



Universidad de las Fuerzas Armadas ESPE
Departamento de Ciencias de la Computación
Ingeniería de Software

Docente: Geovanny Cudco

Aplicaciones Distribuidas

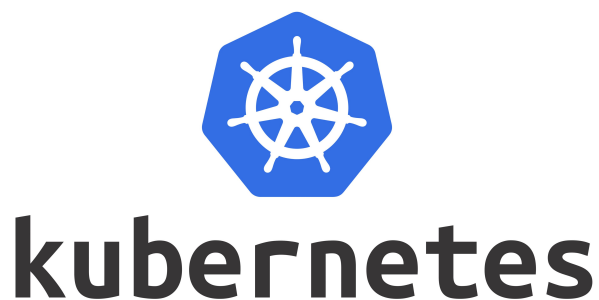
Guía de instalación de Kubernetes en Windows 11 con Minikube

1. Objetivo de la práctica

Instalar y configurar un clúster local de Kubernetes en un equipo con Windows 11 utilizando Minikube y kubectl, con el fin de crear un entorno de pruebas que permita implementar, gestionar y monitorear contenedores de forma orquestada.

2. Contexto e importancia

Figura 1: Logo de Kubernetes.



Kubernetes se ha convertido en el estándar de facto para la orquestación de contenedores en entornos modernos de desarrollo y despliegue de aplicaciones. Su dominio es

fundamental para profesionales en DevOps, cloud computing y desarrollo de microservicios. Esta práctica permite a los estudiantes o desarrolladores aprender los fundamentos de Kubernetes sin necesidad de contar con infraestructura en la nube, aprovechando herramientas como Minikube que simulan un clúster real en el entorno local de Windows 11.

3. Requisitos previos

1. Sistema operativo: Windows 11 Home, Pro o Enterprise
2. Virtualización habilitada en la BIOS (VT-x o AMD-V)
3. Un hipervisor como:
 - WSL 2 (recomendado)
 - Docker
 - Hyper-V (solo en ediciones Pro o Enterprise)
4. Acceso a PowerShell como administrador

4. Pasos para la instalación

4.1. Instalar Minukube

Minikube es Kubernetes local y se centra en facilitar el aprendizaje y el desarrollo para Kubernetes. Es importante tener instalado un contenedor Docker (o similar compatible) o un entorno de máquina virtual.

Nota: Para más información visiste: <https://minikube.sigs.k8s.io/docs/start>

4.1.1. Requisitos mínimos

- 2 CPU o más
- 2 GB de memoria libre
- 20 GB de espacio libre en disco
- Conexión a Internet
- Administrador de contenedores o máquinas virtuales, como: Docker , QEMU , Hyperkit , Hyper-V , KVM , Parallels , Podman , VirtualBox o VMware Fusion/-Workstation

4.1.2. Instalación vía comandos:

Mediante Windows Package Manager

```
winget install Kubernetes.minikube
```

Mediante Chocolatey Package Manager

```
choco install minikube
```

Iniciar el cluster

Desde una terminal con acceso de administrador (pero sin iniciar sesión como root), ejecute:

```
minikube start
```

Figura 2: Inicio del Cluster.

```
PS C:\WINDOWS\system32> minikube start
* minikube v1.36.0 en Microsoft Windows 11 Pro 10.0.26100.4652 Build 26100.4652
* Controlador docker seleccionado automáticamente. Otras opciones: hyperv, ssh
* Using Docker Desktop driver with root privileges
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.47 ...
```

Nota. La salida exitosa de este comando debe arrojar el texto **Done! kubectl is now configured to use “minikube” cluster and “default” namespace by default**

4.1.3. Instalar kubectl (cliente de Kubernetes)

Mediante Windows Package Manager

```
winget install -e --id Kubernetes.kubectl
```

Mediante Chocolatey Package Manager

```
choco install kubernetes-cli -y
```

Figura 3: Instalación de Kubectl.

```
PS C:\WINDOWS\system32> winget install -e --id Kubernetes.kubectl
Found Kubernetes CLI [Kubernetes.kubectl] Version 1.33.3
This application is licensed to you by its owner.
Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.
Downloading https://dl.k8s.io/release/v1.33.3/bin/windows/amd64/kubectl.exe
58.8 MB / 58.8 MB
Successfully verified installer hash
Starting package install...
Command line alias added: "kubectl"
Path environment variable modified; restart your shell to use the new value.
Successfully installed
```

Nota. La salida exitosa de este comando debe arrojar el texto **Successfully installed**

4.1.4. Verificar las instalaciones

```
minikube version
```

```
kubectl version --client
```

Figura 4: Comprobación de las instalaciones.

```
PS C:\WINDOWS\system32> minikube version
minikube version: v1.36.0
commit: f8f52f5de11fc6ad8244afac475e1d0f96841df1-dirty
PS C:\WINDOWS\system32> kubectl version --client
Client Version: v1.32.2
Kustomize Version: v5.5.0
```

5. Iniciar el clúster con Minikube

Iniciar con Docker Desktop o WSL 2:

```
minikube start --driver=docker
```

Figura 5: Iniciar el Cluster con el Driver de Docker.

```
PS C:\WINDOWS\system32> minikube start --driver=docker
* minikube v1.36.0 en Microsoft Windows 11 Pro 10.0.26100.4652 Build 26100.4652
* Using the docker driver based on existing profile
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.47 ...
* Updating the running docker "minikube" container ...
! Failing to connect to https://registry.k8s.io/ from inside the minikube container
* To pull new external images, you may need to configure a proxy: https://minikube.sigs.k8s.io/docs/
/proxy/
* Preparando Kubernetes v1.33.1 en Docker 28.1.1...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Complementos habilitados: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
PS C:\WINDOWS\system32>
```

Verificar el funcionamiento del cluster:

```
kubectl get nodes
```

Figura 6: Salida del comando verificar el funcionamiento del cluster.

```
PS C:\WINDOWS\system32> kubectl get nodes
NAME          STATUS    ROLES          AGE   VERSION
minikube      Ready    control-plane   20m   v1.33.1
PS C:\WINDOWS\system32>
```

Acceder al Dashboard web:

```
minikube dashboard
```

Figura 7: Inicio del Dashboard de Kubernetes.

```
PS C:\WINDOWS\system32> minikube dashboard
* Habilitando dashboard
  - Using image docker.io/kubernetesui/dashboard:v2.7.0
  - Using image docker.io/kubernetesui/metrics-scraper:v1.0.8
* Some dashboard features require the metrics-server addon. To enable all features please run:

    minikube addons enable metrics-server

* Verifying dashboard health ...
* Launching proxy ...
* Verifying proxy health ...
* Opening http://127.0.0.1:54821/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/ in your default browser...
```

Nota. Al finalizar la ejecución de este comando se abrirá el Dashboard en el navegador.

6. Deploy

Crear una implementación de muestra y exponerla en el puerto 8080:

```
kubectl create deployment hello-world --image=kicbase/echo-server:1.0
kubectl expose deployment hello-world --type=NodePort --port=8080
```

Esperar un momento, su implementación aparecerá pronto, posteriormente ingrese:

```
kubectl get services hello-world
```

Figura 8: Verificación del servicio.

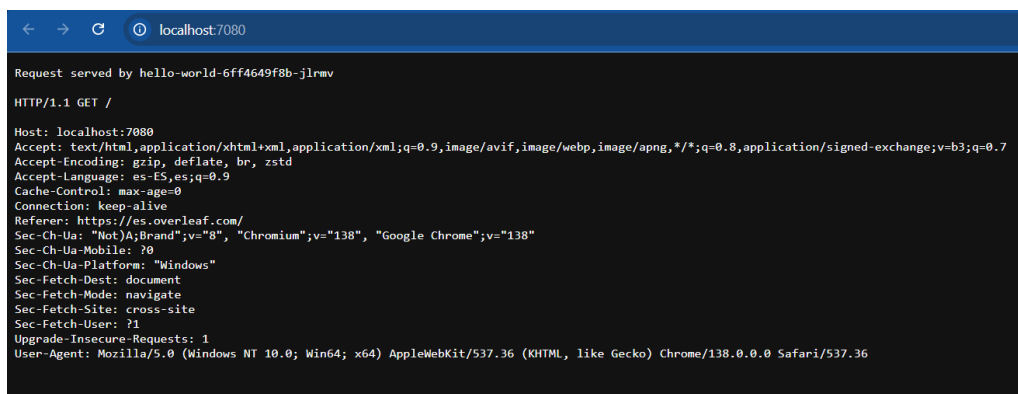
```
PS C:\WINDOWS\system32> kubectl get services hello-world
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
hello-world  NodePort    10.107.92.105 <none>        8080:32699/TCP   7s
PS C:\WINDOWS\system32>
```

Exponer la aplicación en un puerto específico mediante **port-forward**

```
kubectl port-forward service/hello-world 7080:8080
```

Comprobar el funcionamiento en: <http://localhost:7080>

Figura 9: Consulta en el navegador.



```
<  →  ↻  localhost:7080

Request served by hello-world-6ff4649f8b-jlrmv

HTTP/1.1 GET /

Host: localhost:7080
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: es-ES,es;q=0.9
Cache-Control: max-age=0
Connection: keep-alive
Referer: https://es.overleaf.com/
Sec-Ch-Ua: "Not)A;Brand";v="8", "Chromium";v="138", "Google Chrome";v="138"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: cross-site
Sec-Fetch-User: ?1
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36
```

Limpiar el cluster

```
minikube delete --all
```

Despliegue de aplicaciones

En Kubernetes, un Service es un objeto que define cómo acceder a uno o varios Pods (contenedores) a través de la red. Existen varios tipos de Service que cubren distintas necesidades de exposición:

Tabla 1: Tipos de Service en Kubernetes

Tipo de Service	¿Qué hace?	Uso típico
ClusterIP	Crea una IP virtual interna del clúster. Solo accesible desde dentro del mismo clúster.	Comunicaciones internas entre microservicios.
NodePort	Asigna un puerto estático en cada nodo (en rango 30000–32767) y reenvía el tráfico que llega a ese puerto al Service.	Exponer una aplicación fuera del clúster sin infra extra; ideal para desarrollo con Minikube.
LoadBalancer	Solicita a un proveedor de nube un balanceador de carga externo que apunte al Service.	Producción en plataformas cloud (AWS, GCP, Azure).
ExternalName	Mapea el Service a un nombre DNS externo; no crea proxy ni balanceo dentro de Kubernetes.	Integrar servicios externos (p.ej. una base de datos gestionada fuera del clúster).

Requisitos

Tabla 2: Componentes y sus funciones en el entorno Kubernetes

Componente	Función
Docker	Construir imágenes de contenedor.
kubectl	CLI para interactuar con el clúster Kubernetes.
Minikube	Ejecuta un clúster Kubernetes local en tu máquina.
Virtualización	Hypervisor como VirtualBox, Hyper-V o Docker driver.

Práctica 1

Despliegue de una SPA desarrollada con ReactJs, para esta practica se necesita una imagen alojada en <https://hub.docker.com/>, en este ejemplo se trabaja con <https://hub.docker.com/repository/docker/agcudco/ejemplo-prime-crud/general>

```
minikube start --driver=docker
```

Habilitar **Ingress Controller** para gestionar rutas HTTP y HTTPS más flexibles.

```
minikube addons enable ingress
minikube dashboard
```

Verificar que el controlador está corriendo

```
kubectl get pods -n ingress-nginx
kubectl get svc -n ingress-nginx
```

Crear el Namespace

En Kubernetes, un Namespace es una forma de aislar recursos dentro de un mismo clúster. Piensa en él como en un “espacio de nombres” lógico donde agrupar objetos (Pods, Services, Deployments, ConfigMaps, etc.) que comparten alguna característica, equipo o finalidad.

```
# 1namespace.yml

apiVersion: v1
kind: Namespace
metadata:
  name: prime-crud-ns
```

Crear el Deployment

En Kubernetes, un Deployment es un objeto de alto nivel que gestiona la creación y actualización de uno o varios Pods (conjuntos de contenedores) de forma declarativa. Su función principal es garantizar que el estado real de tus aplicaciones coincida siempre con el estado deseado que tú defines.

Funciones clave de un Deployment

1. **Despliegues declarativos.** Defines cuántas réplicas de un Pod quieres (por ejemplo, 3 réplicas) y Kubernetes se encarga de crearlas, eliminarlas o reemplazarlas para mantener ese número.
2. **Actualizaciones progresivas (Rolling Updates).** Permite actualizar la versión de la aplicación sin tiempo de inactividad: crea progresivamente Pods nuevos con la versión actualizada y elimina los antiguos conforme los nuevos estén listos.
3. **Reversión automática (Rollback).** Si durante una actualización algo falla (por ejemplo, los nuevos Pods no arrancan correctamente), Kubernetes puede revertir automáticamente al Deployment anterior.
4. **Escalado fácil.** Cambiando simplemente el campo replicas, puedes aumentar o disminuir el número de Pods en ejecución.

```
# 2deployment.yml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: prime-crud
  namespace: prime-crud-ns
  labels:
    app: prime-crud
spec:
  replicas: 2
  selector:
    matchLabels:
      app: prime-crud
  template:
    metadata:
      labels:
        app: prime-crud
    spec:
      containers:
        - name: app
          image: agcudco/ejemplo-prime-crud:latest # <-- imagen pública
          ports:
            - containerPort: 80
          resources:
            requests:
              cpu: 50m
              memory: 64Mi
            limits:
              cpu: 200m
              memory: 128Mi
```

Crear el Service

En Kubernetes, un Service es un objeto que proporciona una forma estable y fiable de acceder a uno o varios Pods, independientemente de su ciclo de vida o de la IP que tengan en cada momento. Piensa en él como una “capa de abstracción” que:

1. Descubre automáticamente los Pods que cumplen un selector de etiquetas.
2. Balancea el tráfico entre ellos.
3. Mantiene una IP virtual (ClusterIP) y, opcionalmente, puertos y rutas, para que no tengas que preocuparte de las IP dinámicas de los Pods.

¿Para qué se usa un Service?

- Exponer una aplicación dentro del clúster (por ejemplo, que otros microservicios puedan invocarla).
- Exponer una aplicación al exterior (internet o red local) usando NodePort, Load-Balancer o Ingress.
- Balanceo de carga sencillo y nativo de Kubernetes.
- Descubrimiento de servicios: otros Pods pueden encontrar el Service mediante DNS interno (nombre-del-service.namespace.svc.cluster.local).

Componentes clave

- Selector: etiquetas que indican qué Pods “pertenecen” al Service.
- ClusterIP: la IP interna estable que el Service asigna.
- Port mapping: puertos que el Service abre y a qué puerto del Pod se redirigen.

```
# 3servicio.yml

apiVersion: v1
kind: Service
metadata:
  name: prime-crud-svc
  namespace: prime-crud-ns
spec:
  selector:
    app: prime-crud
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  type: ClusterIP
```

Crear el Ingress

Un Ingress en Kubernetes es un recurso de nivel superior que gestiona el acceso HTTP y HTTPS externo a uno o varios Services dentro de tu clúster. Actúa como un controlador de rutas (reverse proxy) que, basándose en reglas de host y de ruta, dirige el tráfico entrante al Service correspondiente.

¿Por qué usar Ingress?

1. **Un único punto de entrada.** En lugar de exponer cada Service con NodePort o LoadBalancer, un Ingress centraliza el acceso externo bajo un único dominio y configuración TLS.
2. **Enrutamiento basado en Host/Ruta.** Puede definir reglas como:
 - `api.miapp.com` → Service A
 - `miapp.com/blog` → Service B
 - `miapp.com/shop` → Service C
3. **Terminación TLS.** Gestiona certificados TLS en el Ingress, de modo que el cifrado se descifra allí y el tráfico interno puede ser HTTP puro.
4. **Middleware y anotaciones.** Con controladores como NGINX, Traefik o Contour puedes aplicar límites de velocidad, autenticación, reescritura de URLs y mucho más vía anotaciones.

```
# 4ingress.yml

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: prime-crud-ingress
  namespace: prime-crud-ns
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
    - host: prime-crud.local # cambiar por un dominio real
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: prime-crud-svc
                port:
                  number: 80
```

Finalizada la creación de los archivos .yaml ingresar en un terminal:

```
kubectl apply -f 1namespace.yml
kubectl apply -f 2deployment.yml
kubectl apply -f 3servicio.yml
```

```
kubectl apply -f 4ingress.yml
```

#verificar el funcionamiento

```
kubectl get pods -n prime-crud-ns -w
```

Crear un dominio local en el archivo etc/hosts

En Windows (CMD admin)

```
minikube ip
```

copia la IP

```
notepad C:\Windows\System32\drivers\etc\hosts
```

añade la línea

```
# <IP> prime-crud.local
```

Verificar el despliegue en el Dashboard de Kubernetes y acceder a la ruta <http://prime-crud.local>

En caso de que el servicio no se haya desplegado, verificar su funcionamiento mediante:

Comprobar que el namespace existe

```
kubectl get ns prime-crud-ns
```

Ver los pods y sus estados

```
kubectl get pods -n prime-crud-ns
```

Ver el deployment

```
kubectl get deploy -n prime-crud-ns
```

Ver el servicio

```
kubectl get svc -n prime-crud-ns
```

Ver el ingress

```
kubectl get ingress -n prime-crud-ns
```

Ver todo de un vistazo

```
kubectl get all -n prime-crud-ns
```

En caso de tener problemas con el **Ingress/DNS/hosts** se recomienda realizar un **port-forward**

```
kubectl port-forward -n prime-crud-ns svc/prime-crud-svc 8080:80
```

Posteriormente acceder a <http://localhost:8080>

Práctica 2

En la práctica se emplean tres elementos: la base de datos, el backend y el frontend.

1. **La base de datos** se levanta en el yamel denominado 1basededatos.yml, y emplea un volumen persistente.
2. **Backend** es un proyecto en Node JS, una api rest que conecta a una base de datos NO SQL de mongoDB.
3. **FrontEnd** es un proyecto en Vue 2, emplea vuetify y consume la api rest del backend

Importante: Para exponer la apps se utiliza un ingress por lo que debe habilitar dicho addons en el minikube.

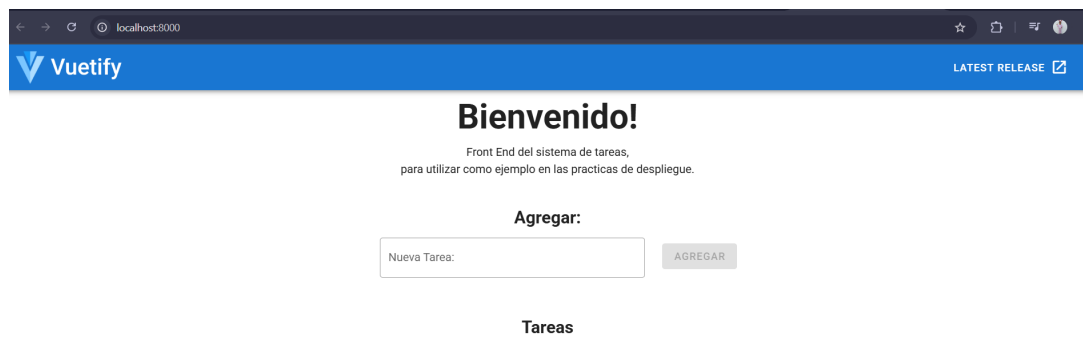
Descargue los archivos desde el repositorio:

<https://github.com/agcudco/taller-kubernetes.git>

En caso de tener problemas con el Ingress, realizar un port-forward posteriormente ingresar a <http://localhost:8000/>

```
kubectll port-forward -n ejemplo svc/frontend 8000:80
```

Figura 10: Despliegue en local.



Actividad Propuesta

Realice el despliegue del proyecto AppPublicaciones mediante un Cluster Local de kubernetes y que sea visible desde la url <http://app-publicaciones.local>