PE03 - Proyecto Final: Infraestructura Completa (Vídeo)

Duración estimada: 4-5 horas (desarrollo) + 1 hora (grabación del vídeo)
Dificultad: ⭐⭐⭐⭐ Alta

Objetivo

Diseñar e implementar una infraestructura completa con múltiples servicios utilizando Vagrant, y **grabar un vídeo explicativo** donde demuestres su funcionamiento, expliques tu proceso y compartas lo aprendido. El alumno elegirá entre varias opciones de proyecto y deberá demostrar dominio de todos los conceptos del curso.

Formato de entrega: Vídeo

La entrega principal de este proyecto es un **vídeo grabando tu pantalla mientras explicas en voz alta** todo tu trabajo. No es una presentación de diapositivas: se trata de que muestres tu entorno real, tu código y tus máquinas funcionando.

Requisitos del vídeo

Aspecto	Requisito
Duración	Entre **10 y 20 minutos**
Formato	Grabación de pantalla con audio (tu voz explicando)
Resolución mínima	720p (se debe poder leer el código)
Audio	Claro y entendible. Puedes usar micrófono del portátil o auriculares
Formato de archivo	MP4, MKV o enlace a YouTube/Stream (no listado)

Herramientas de grabación recomendadas (gratuitas)

- **OBS Studio** (Windows/macOS/Linux) — obsproject.com
- **Xbox Game Bar** (Windows) — `Win + G`
- **Grabación de pantalla nativa** (Windows 11) — `Win + Alt + R`
- **ShareX** (Windows) — getsharex.com

> **Consejo**: Antes de grabar el vídeo final, haz una prueba corta de 30 segundos para verificar que se graba bien la pantalla y se escucha tu voz.

Estructura del vídeo

Tu vídeo debe tener las siguientes **4 partes**, en este orden:

Parte 1: Presentación e introducción (2-3 min)

- Di tu **nombre y la opción elegida**
- Explica brevemente **qué es el proyecto** y para qué sirve la infraestructura que has montado
- Muestra el **diagrama de arquitectura** (puede ser un dibujo, un esquema en papel, un diagrama digital o Mermaid renderizado)
- Indica las **IPs, servicios y puertos** que usas

Parte 2: Código y configuración (3-5 min)

- Abre tu **Vagrantfile** y explícalo por partes: qué hace cada bloque, por qué has configurado cada VM así
- Muestra tus **scripts de provisioning** y comenta qué instalan y configuran
- Si usas archivos de configuración externos (nginx.conf, settings.yml, etc.), muéstralos y explica su propósito
- Comenta cómo está **organizado el proyecto** (estructura de carpetas)

Parte 3: Demostración en vivo (4-8 min)

Esta es la parte más importante. Debes demostrar que todo funciona:

- Ejecuta `vagrant status` para mostrar las VMs
- Muestra el **servicio principal funcionando** desde el navegador o la terminal
- Entra por SSH a las máquinas (`vagrant ssh`) y demuestra que los servicios están corriendo
- Muestra la **comunicación entre VMs** (ping, curl, conexión a base de datos, etc.)
- Si tu proyecto tiene balanceo de carga, demuestra que las peticiones se reparten
- Si tiene API, haz peticiones reales y muestra las respuestas

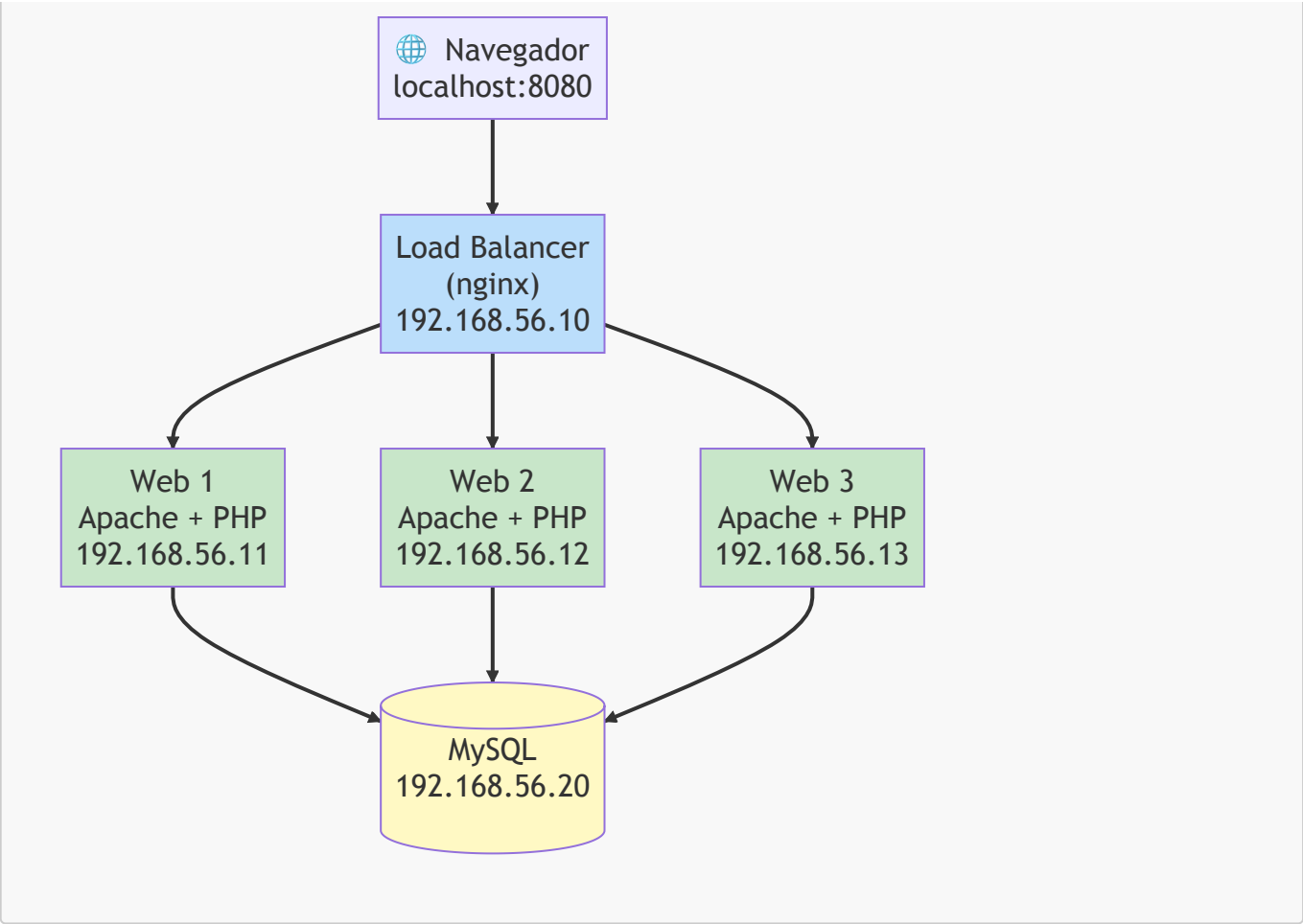
Parte 4: Reflexión final (1-2 min)

- ¿Qué ha sido lo **más difícil** del proyecto?
- ¿Qué **problemas** has encontrado y cómo los has resuelto?
- Si tuvieras más tiempo, ¿qué **mejorarías**?

Opciones de proyecto

Elige **una** de las siguientes opciones. Lee la descripción completa de cada una para decidir cuál te interesa más.

Opción A: Cluster web con balanceo de carga



¿En qué consiste?

Se trata de montar una infraestructura web de **alta disponibilidad**. El usuario accede a una web a través de un **balanceador de carga** (nginx o HAProxy) que reparte las peticiones entre varios servidores web idénticos. Todos los servidores web se conectan a una única base de datos compartida.

Este es el tipo de arquitectura que usan empresas reales para que sus webs aguanten mucho tráfico y no se caigan si un servidor falla.

Máquinas virtuales necesarias (5 VMs)

VM	Hostname	IP	RAM	Función
Load Balancer	`lb`	192.168.56.10	512 MB	Recibe todas las peticiones y las reparte entre los webs
Web 1	`web1`	192.168.56.11	1024 MB	Servidor Apache + PHP. Sirve la aplicación
Web 2	`web2`	192.168.56.12	1024 MB	Servidor Apache + PHP. Idéntico al anterior
Web 3	`web3`	192.168.56.13	1024 MB	Servidor Apache + PHP. Idéntico al anterior

VM	Hostname	IP	RAM	Función
Base de datos	`db`	192.168.56.20	2048 MB	MySQL. Almacena los datos de la aplicación

Qué se espera que funcione

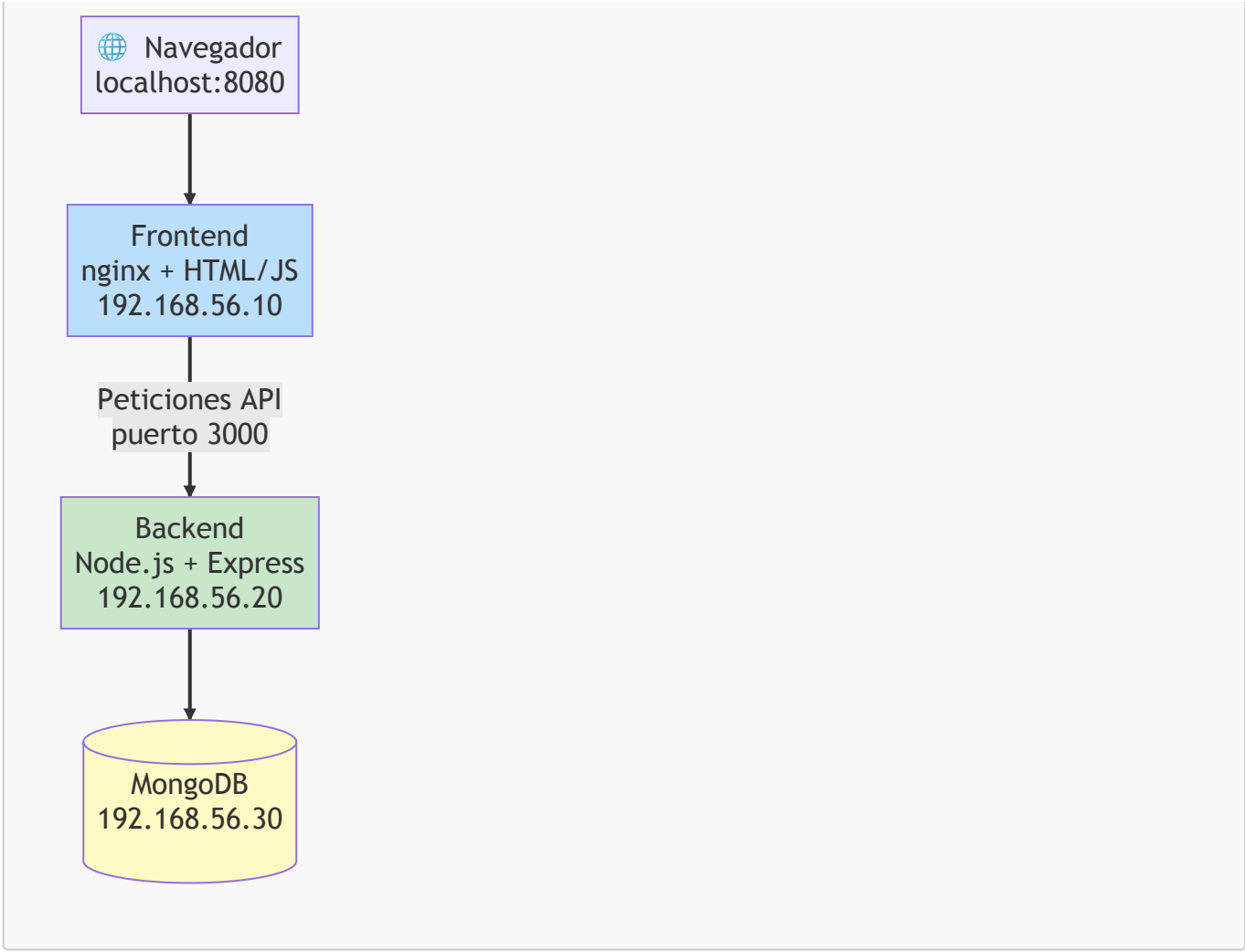
1. Al acceder a `http://localhost:8080` se ve una **página web** servida por Apache
2. Al recargar la página varias veces, el contenido indica que **la petición va a un servidor web distinto** cada vez (round-robin). Por ejemplo, la página muestra "Servido por: web1", "Servido por: web2", etc.
3. La aplicación PHP se **conecta a MySQL** en la máquina `db` y realiza alguna consulta (mostrar datos de una tabla, un contador de visitas, etc.)
4. Si apagas una de las webs (`vagrant halt web2`), la aplicación **sigue funcionando** porque el balanceador envía las peticiones a las otras dos

Qué debes demostrar en el vídeo

- Hacer `curl` varias veces seguidas a `http://localhost:8080` y ver cómo cambia el servidor que responde
- Conectarte por SSH a un web y verificar que Apache y PHP están funcionando
- Conectarte por SSH a `db` y mostrar la base de datos con `mysql -u usuario -p -e "SHOW DATABASES;"`
- Mostrar que si apagas un web, el servicio sigue funcionando

---

Opción B: Stack MEAN/MERN (aplicación JavaScript fullstack)



¿En qué consiste?

Se trata de montar una **aplicación web moderna** con arquitectura de tres capas separadas: un servidor de frontend que sirve la interfaz visual (HTML/CSS/JavaScript), un servidor de backend con una API REST (Node.js + Express), y una base de datos NoSQL (MongoDB). Cada capa vive en su propia máquina virtual.

Es la arquitectura típica de aplicaciones modernas como las que se crean con React, Angular o Vue en el frontend.

Máquinas virtuales necesarias (3 VMs)

VM	Hostname	IP	RAM	Función
Frontend	`frontend`	192.168.56.10	512 MB	nginx sirviendo archivos HTML/CSS/JS estáticos. La interfaz del usuario
Backend	`backend`	192.168.56.20	1024 MB	Node.js + Express. API REST que recibe peticiones del frontend
MongoDB	`mongo`	192.168.56.30	1024 MB	Base de datos NoSQL. Almacena la información en formato JSON (documentos)

Qué se espera que funcione

1. Al acceder a `http://localhost:8080` se ve una **página web** (interfaz frontend) con un diseño sencillo
2. El frontend hace **peticiones HTTP a la API** del backend (por ejemplo, `fetch("http://192.168.56.20:3000/api/items")`) y muestra los datos recibidos en la página
3. La API del backend responde con datos en formato **JSON** y los lee/escribe de MongoDB
4. Se puede usar `curl http://192.168.56.20:3000/api/items` desde la terminal para probar la API directamente

Ejemplo mínimo de la API

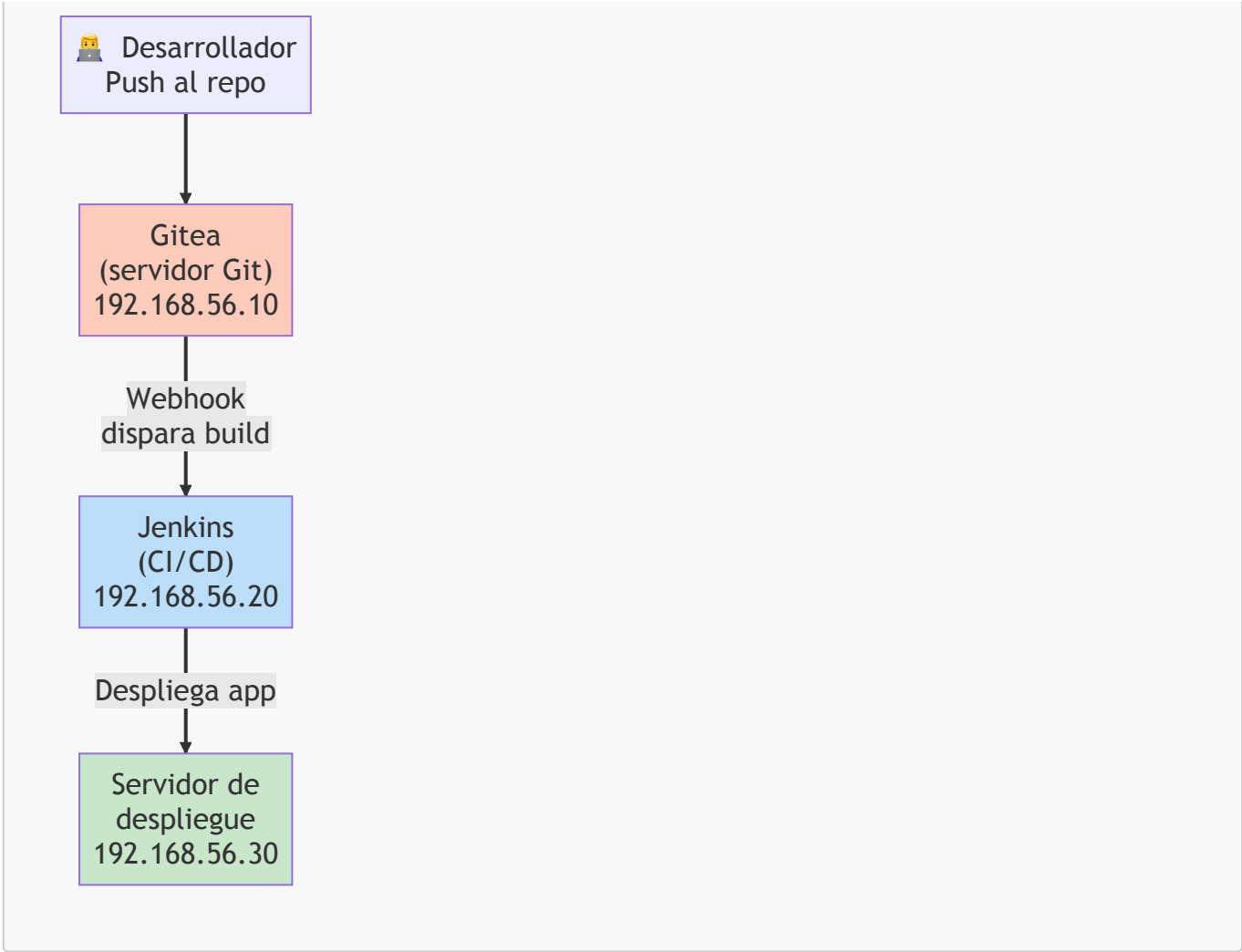
No hace falta que la aplicación sea compleja. Un ejemplo sencillo sería:

Método	Ruta	Descripción
`GET`	`/api/items`	Devuelve una lista de elementos
`POST`	`/api/items`	Crea un nuevo elemento
`DELETE`	`/api/items/:id`	Elimina un elemento

Qué debes demostrar en el vídeo

- Abrir el **navegador** y mostrar el frontend funcionando
- Hacer peticiones con `curl` a la API y ver las respuestas JSON
- Conectarte a MongoDB (`mongosh`) y mostrar la base de datos con los documentos
- Crear un dato desde el frontend y verificar que aparece en MongoDB

Opción C: Entorno de integración continua (CI/CD)



¿En qué consiste?

Se trata de montar un **entorno DevOps básico** con un servidor Git propio (Gitea, que es una alternativa ligera a GitLab), un servidor de integración continua (Jenkins) y un servidor donde se despliega la aplicación. Cuando alguien sube código al repositorio Git, Jenkins detecta el cambio y automáticamente construye y despliega la aplicación en el servidor de producción.

Este tipo de entorno es lo que usan los equipos de desarrollo profesionales para automatizar el despliegue de sus aplicaciones.

Máquinas virtuales necesarias (3 VMs)

VM	Hostname	IP	RAM	Función
Gitea	`gitea`	192.168.56.10	1024 MB	Servidor Git con interfaz web. Aquí se suben los repositorios
Jenkins	`jenkins`	192.168.56.20	2048 MB	Servidor CI/CD. Ejecuta los pipelines automáticamente
Deploy	`deploy`	192.168.56.30	1024 MB	Servidor de producción. Donde se despliega la aplicación final

> **Nota**: Se recomienda Gitea en lugar de GitLab porque GitLab requiere muchísima RAM (mínimo 4 GB solo para GitLab) y los portátiles de clase no lo van a soportar bien. Gitea hace lo mismo pero es mucho más ligero.

Qué se espera que funcione

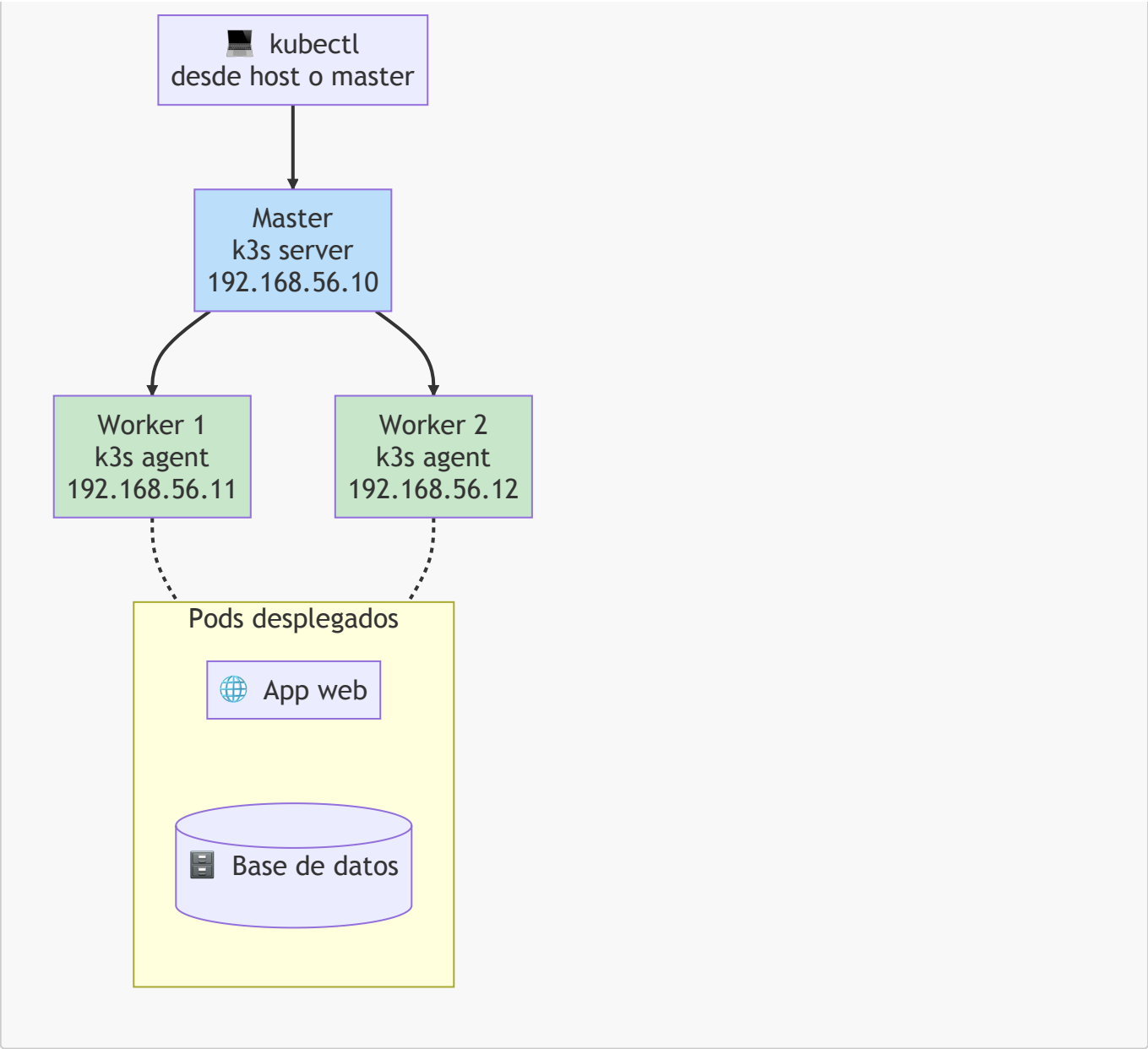
1. Acceder a **Gitea** en `http://192.168.56.10:3000` y ver la interfaz web del servidor Git con al menos un repositorio creado
2. Acceder a **Jenkins** en `http://192.168.56.20:8080` y ver el panel de control con al menos un pipeline/job configurado
3. El pipeline de Jenkins, al ejecutarse, hace algo visible en el servidor de despliegue: por ejemplo, despliega una página web, ejecuta un script, o copia archivos
4. El servidor de **despliegue** tiene un servicio corriendo (por ejemplo, nginx mostrando la aplicación desplegada)

Qué debes demostrar en el vídeo

- Abrir **Gitea** en el navegador, mostrar el repositorio y hacer un `git push` desde la terminal
- Abrir **Jenkins**, mostrar el pipeline y ejecutarlo (o mostrar que se ejecuta automáticamente al hacer push)
- Ver los **logs del pipeline** en Jenkins
- Acceder al servidor de despliegue y mostrar que la aplicación se ha desplegado correctamente
- Si funciona el webhook automático (Gitea → Jenkins), mostrarlo es un gran plus

---

Opción D: Cluster Kubernetes con k3s (avanzado)



¿En qué consiste?

Se trata de montar un **cluster Kubernetes** usando k3s (una versión ligera de Kubernetes) con un nodo master y dos nodos worker. Dentro del cluster se despliega una aplicación usando manifiestos de Kubernetes (Deployments, Services, etc.).

Kubernetes es la tecnología estándar del sector para orquestar contenedores en producción. Este proyecto es el más avanzado y ambicioso de todas las opciones.

> **Importante**: Esta opción es para alumnos que buscan un reto extra. Kubernetes tiene una curva de aprendizaje pronunciada. No se penalizará si no funciona al 100%, pero se valorará mucho el esfuerzo y la comprensión de los conceptos.

Máquinas virtuales necesarias (3 VMs)

VM	Hostname	IP	RAM	Función
Master	`master`	192.168.56.10	2048 MB	Nodo master de k3s. Ejecuta el API server y el plano de control

VM	Hostname	IP	RAM	Función
Worker 1	`worker1`	192.168.56.11	1024 MB	Nodo worker. Ejecuta los pods (contenedores de aplicación)
Worker 2	`worker2`	192.168.56.12	1024 MB	Nodo worker. Igual que el anterior para tener redundancia

Qué se espera que funcione

1. El cluster k3s está operativo: `kubectl get nodes` muestra los 3 nodos en estado **Ready**
2. Hay al menos un **Deployment** ejecutándose (por ejemplo, nginx o una app web sencilla)
3. Hay un **Service** que expone la aplicación y se puede acceder desde el host
4. Se puede ver con `kubectl get pods` que los pods están corriendo y repartidos entre los workers

Qué debes demostrar en el vídeo

- Ejecutar `kubectl get nodes` y mostrar los 3 nodos en estado Ready
- Ejecutar `kubectl get pods -o wide` y mostrar los pods corriendo en distintos workers
- Acceder a la **aplicación desplegada** desde el navegador o con `curl`
- Eliminar un pod (`kubectl delete pod ...`) y mostrar cómo Kubernetes lo **recrea automáticamente**
- Mostrar los manifiestos YAML usados (Deployment, Service) y explicar qué hacen

---

Opción E: Proyecto libre

Si tienes una idea propia que te motive más que las opciones anteriores, puedes proponer tu propio proyecto. **Debes obtener la aprobación del profesor antes de empezar.**

Requisitos mínimos para un proyecto libre

- **Mínimo 3 VMs** con servicios interconectados
- Provisioning **completamente automatizado** (un solo `vagrant up` configura todo)
- Red privada con **comunicación real** entre las máquinas
- Algún **servicio accesible** desde el host (web, API, panel de administración, etc.)

Ejemplos de proyectos libres válidos

- **Servidor de correo electrónico**: Postfix + Dovecot + Roundcube + base de datos
- **Plataforma de monitorización**: Prometheus + Grafana + Node Exporter en varios servidores
- **Sistema ERP/CRM**: Odoo/ERPNext con servidor web y base de datos separados
- **Servidor de juegos**: Servidor dedicado de Minecraft/Terraria con panel web de administración y base de datos de usuarios
- **Plataforma educativa**: Moodle con servidor web, base de datos y servidor de archivos separados

---

Requisitos obligatorios (todos los proyectos)

Independientemente de la opción elegida, tu proyecto debe cumplir estos requisitos:

1. Infraestructura (2.5 puntos)

- [ ] Mínimo 3 VMs funcionando y comunicadas
- [ ] Red privada configurada correctamente
- [ ] Port forwarding para acceder a los servicios desde el host
- [ ] Recursos asignados de forma razonable (RAM, CPU)

2. Provisioning automatizado (2.5 puntos)

- [ ] Al ejecutar `vagrant up` todo se instala y configura automáticamente
- [ ] Scripts organizados (separados por servicio, no un script gigante)
- [ ] Scripts comentados para que se entienda qué hacen
- [ ] Sin intervención manual: no debe hacer falta entrar por SSH a configurar nada

3. Vídeo explicativo (3 puntos)

- [ ] Duración entre 10 y 20 minutos
- [ ] Se graba la pantalla y se escucha tu voz explicando
- [ ] Se ve el código (Vagrantfile y scripts) y lo explicas
- [ ] Se demuestra el funcionamiento real (no capturas estáticas)
- [ ] Se demuestra la comunicación entre máquinas
- [ ] Incluye la reflexión final (dificultades y aprendizajes)

4. Funcionamiento real (2 puntos)

- [ ] El servicio principal es accesible (desde navegador o terminal)
- [ ] Las VMs se comunican entre sí correctamente
- [ ] Los datos persisten (lo que se guarda en la BD se puede consultar)

Puntos extra (hasta 2 puntos adicionales)

Extra	Puntos	Descripción
Ansible	+1.0	Usar Ansible en lugar de (o además de) Shell para el provisioning
Variables externas	+0.5	Configuración en un archivo YAML/JSON separado, no hardcoded
SSL/HTTPS	+0.5	Certificados configurados (pueden ser autofirmados)
Monitorización	+0.5	Algún tipo de health-check o monitorización (Prometheus, script propio, etc.)
README excelente	+0.5	README.md completo con diagrama, instrucciones y troubleshooting
Calidad del vídeo	+0.5	Vídeo especialmente bien explicado, claro y bien estructurado

Estructura del proyecto

```
PE03-Proyecto/
├── README.md                # Descripción del proyecto
├── Vagrantfile              # Configuración principal
├── scripts/                 # Scripts de provisioning
│   ├── common.sh            # Configuración común a todas las VMs
│   ├── servicio1.sh         # Script del servicio 1
│   ├── servicio2.sh         # Script del servicio 2
│   └── ...
├── config-files/            # Archivos de configuración (en caso de que
sean necesarios)
│   ├── nginx.conf
│   └── ...
└── ansible/                 # (Opcional) Si usas Ansible
    ├── playbook.yml
    └── roles/
```

Criterios de evaluación detallados

Rúbrica

Criterio	Suspenso (0-4)	Aprobado (5-6)	Notable (7-8)	Sobresaliente (9-10)
Infraestructura	No arranca o falta VMs	VMs arrancan pero incompletas	Todo funciona correctamente	Config. óptima con extras
Provisioning	Manual o no funciona	Semi-automático, hay pasos manuales	Automático completo	Idempotente, modular y limpio
Vídeo - Código	No muestra el código	Muestra sin explicar	Explica correctamente	Explicación clara y detallada
Vídeo - Demo	No demuestra nada	Demo parcial, cosas no funcionan	Demo completa, todo funciona	Demo impecable con casos extra
Vídeo - Expresión	No se entiende	Se entiende pero poco claro	Buena explicación	Excelente comunicación
Reflexión	No hay reflexión	Muy superficial	Reflexión con contenido	Gran análisis de problemas y soluciones

---

Entrega

Qué hay que entregar

1. **El vídeo** (obligatorio):
    - Archivo MP4/MKV subido a la plataforma del centro, **o**
    - Enlace a YouTube/Microsoft Stream como vídeo **no listado** (no público, no privado)
2. **El código** (obligatorio):
    - Repositorio Git (GitHub/GitLab) con todo el código, **o**
    - Archivo ZIP con el proyecto completo

Fecha límite

[Lunes 2 de marzo a las 08:00]

Consejos para grabar un buen vídeo

1. **Prepara un guión**: No hace falta que lo leas palabra por palabra, pero ten claro el orden de lo que vas a mostrar. Apunta en un papel las 4 partes y los puntos que quieres cubrir
2. **Levanta las VMs antes de grabar**: No grabes el `vagrant up` entero (tarda demasiado). Ten las VMs ya corriendo y empieza mostrando `vagrant status`
3. **Practica la demo una vez antes**: Asegúrate de que todo funciona antes de darle al botón de grabar
4. **Habla con naturalidad**: No hace falta que sea perfecto. Si te equivocas, sigue adelante o graba de nuevo solo ese trozo
5. **Aumenta el tamaño de la fuente**: En la terminal y en el editor, pon la fuente grande para que se lea bien al grabar (16-18 pt mínimo)
6. **No grabes más de 20 minutos**: Si te pasas, recorta. La brevedad se valora

---

FAQ

P: ¿Puedo usar Docker dentro de las VMs?
R: Sí, en caso necesario y que queráis, no es obligatorio.

P: ¿Puedo trabajar en grupo?
R: No, es trabajo individual.

P: ¿Puedo cambiar de opción a mitad del proyecto?
R: Sí.

P: ¿Qué pasa si no me funciona todo al 100%?
R: Entrega lo que tengas. En el vídeo, explica qué funciona, qué no y por qué crees que falla. Se valora la honestidad y el análisis.

P: ¿Tengo que mostrar la cara en el vídeo?
R: No, solo se necesita grabación de pantalla y tu voz.

P: ¿Puedo editar el vídeo?
R: Sí, puedes cortar silencios, añadir títulos entre secciones, etc. Pero no es obligatorio.

P: ¿Cuánta RAM necesito en mi PC para estos proyectos?
R: Recomendable 16 GB. Con 8 GB es justo pero posible si cierras todo lo demás. Si tienes problemas de recursos, baja la RAM asignada a cada VM.