

Práctica calificada 5 CC3S2

Instrucciones de desarrollo y formato de entrega

Objetivo general

Completar los ejercicios avanzados utilizando Ansible, Vagrant y Docker, siguiendo buenas prácticas de desarrollo y automatización. Deberás estructurar tu proyecto de manera organizada, utilizar control de versiones con Git y compartir tu trabajo a través de un repositorio en GitHub. Además, prepararás un archivo comprimido con todo el código y documentación necesaria.

Estructura del proyecto

Organiza tu proyecto con la siguiente estructura de directorios:

```
 proyecto/
    ├── ansible/
    |   ├── ejercicio1/
    |   ├── ejercicio2/
    |   ├── ejercicio3/
    |   ├── ejercicio4/
    |   ├── ejercicio5/
    |   ├── ejercicio6/
    |   ├── ejercicio7/
    |   ├── ejercicio8/
    |   ├── ejercicio9/
    |   ├── ejercicio10/
    ├── handlers/
    ├── templates/
    ├── vagrant/
    |   └── Vagrantfile
    ├── site.yml
    ├── README.md
    └── docs/
        └── reportes y documentación adicional
```

- **ansible/**: Contiene subcarpetas para cada ejercicio con las tareas específicas.
- **handlers/**: Almacena los manejadores utilizados en los ejercicios.
- **templates/**: Contiene las plantillas (.j2) utilizadas en los playbooks.
- **vagrant/**: Incluye el Vagrantfile para levantar la máquina virtual.
- **site.yml**: Playbook principal de Ansible que importa las tareas de los ejercicios.
- **README.md**: Archivo con instrucciones y descripción del proyecto.
- **docs/**: Carpeta para documentación adicional, reportes y capturas de pantalla.

Instrucciones de desarrollo

1. Configuración de la carpeta del repositorio Git

2. Configurar el archivo .gitignore:

Crea un archivo .gitignore para excluir archivos innecesarios, como archivos temporales, claves privadas y archivos de máquina virtual generados.

Contenido sugerido para .gitignore:

```
markdown  
Copy code  
*.log  
*.pyc  
__pycache__/  
.vagrant/  
.env  
*.box  
*.iso  
*.vdi
```

3. Realizar commits regulares:

A medida que avances en el desarrollo, realiza commits frecuentes con mensajes claros que describan los cambios realizados.

4. Publicación en GitHub

Desarrollo de los ejercicios

Ejercicio 1 al 6:

- Completa cada ejercicio en su respectiva carpeta dentro de ansible/.
- Asegúrate de mantener el playbook site.yml actualizado, importando las tareas correspondientes.
- Utiliza buenas prácticas de Ansible, como la reutilización de código, variables y plantillas.
- Documenta cualquier configuración especial o consideración en el archivo README.md.

Ejercicios adicionales con docker :

- Crea una carpeta docker/ en la raíz del proyecto para almacenar archivos relacionados con Docker y Kubernetes.
- Sigue las instrucciones de cada ejercicio, asegurándote de documentar los pasos y decisiones tomadas.

- Si utilizas Docker Compose, Kubernetes o herramientas adicionales, incluye los archivos de configuración correspondientes en las subcarpetas adecuadas (docker/compose/, docker/k8s/, etc.).

Documentación y reportes

README.md:

- Proporciona una descripción general del proyecto y los ejercicios realizados.
- Incluye instrucciones sobre cómo reproducir el entorno, ejecutar los playbooks y cualquier requisito previo.
- Documenta cualquier problema encontrado y cómo lo solucionaste.

Documentación adicional:

- En la carpeta docs/, incluye cualquier documento adicional que sea relevante, como:
 - o Reportes de escaneos de seguridad.
 - o Capturas de pantalla de aplicaciones funcionando.
 - o Notas sobre pruebas realizadas.
 - o Diagrama de la arquitectura implementada.

5. Preparación del archivo comprimido para entrega

Verificar el contenido: Asegúrate de que el proyecto esté completo y que todos los archivos necesarios estén incluidos.

Excluir archivos innecesarios: Antes de comprimir, verifica que no estás incluyendo archivos pesados o innecesarios (por ejemplo, imágenes de máquinas virtuales).

Comprimir el proyecto y verificar el archivo comprimido:

Asegúrate de que el archivo comprimido se creó correctamente y que incluye todos los archivos y carpetas necesarios.

Nota: Este archivo debe estar en la plataforma del curso (UNI VIRTUAL) de lo contrario se revisará sobre 10 el examen.

Formato de entrega

Archivos a entregar

Repositorio de GitHub:

- Proporciona el enlace al repositorio público o privado (en este último caso, asegúrate de dar acceso al instructor o evaluador).
- El repositorio debe estar actualizado con todos los commits y reflejar el historial de desarrollo.

Archivo comprimido:

- Entrega el archivo comprimido (.tar.gz o .zip) que contiene todo el proyecto.
- Asegúrate de que el archivo no exceda los límites de tamaño establecidos por el instructor o plataforma de entrega.

Instrucciones para la entrega

- **Fecha límite:** Asegúrate de conocer la fecha y hora límite para la entrega del proyecto.
- **Plataforma de entrega:** Si hay una plataforma específica (por ejemplo, un aula virtual), sigue las instrucciones para subir el archivo y proporcionar el enlace al repositorio.
- **Información de contacto:** Incluye en el README.md tu nombre completo, correo electrónico y cualquier otra información solicitada por el instructor.

Consideraciones finales

- **Calidad del código:** Sigue buenas prácticas de codificación y estilo. Asegúrate de que tu código sea legible y esté bien organizado.
- **Pruebas:** Verifica que todos los playbooks y scripts funcionan correctamente y que puedes levantar el entorno desde cero siguiendo tus propias instrucciones.
- **Seguridad:** No incluyas información sensible en el repositorio o en los archivos de entrega (por ejemplo, contraseñas, claves privadas, tokens de acceso).
- **Originalidad:** Asegúrate de que tu trabajo es original y cumple con las políticas de integridad académica.

Estructura del proyecto

Archivo principal: site.yml (playbook de Ansible).

Carpetas:

- ansible/: Contiene subcarpetas para cada ejercicio (ejercicio1, ejercicio2, ..., ejercicio6).
- handlers/: Almacena los manejadores utilizados en los ejercicios.
- vagrant/: Contiene el Vagrantfile para levantar la máquina virtual.

Ejercicio 1: Configuración básica del sistema (1 punto)

Descripción

Configura una máquina virtual Ubuntu utilizando Vagrant y Ansible. En este ejercicio, deberás:

- Crear la estructura de directorios mencionada.
- En ansible/ejercicio1/, implementar tareas para:

- Actualizar el sistema operativo.
- Configurar locales y zona horaria.
- Crear usuarios y grupos con permisos específicos.

Instrucciones

1. **Crear la estructura de directorios.**
2. **Crear el Vagrantfile en vagrant/**. Configura la máquina virtual con:
 - a. Ubuntu 20.04.
 - b. 2 GB de RAM.
 - c. Red privada con IP estática.
 - d. Provisión con Ansible usando el playbook site.yml.
3. **Escribir el playbook site.yml.** Incluye la importación de las tareas de ansible/ejercicio1/main.yml.
4. **Crear las tareas en ansible/ejercicio1/main.yml.** Implementa tareas para:
 - a. Actualizar los paquetes del sistema.
 - b. Configurar la zona horaria y locales.
 - c. Crear un grupo llamado admin.
 - d. Crear un usuario devuser perteneciente al grupo admin con contraseña cifrada.

Resultados esperados

- El sistema operativo actualizado y configurado.
- Usuario devuser y grupo admin creados.

Ejercicio 2: Implementación de servicios web con seguridad básica (2 puntos)

Descripción

Aumenta la complejidad al desplegar un servidor web seguro. Deberás:

- Instalar y configurar Nginx.
- Generar certificados SSL autofirmados para habilitar HTTPS.
- Configurar reglas básicas de firewall (ufw) para permitir solo tráfico SSH, HTTP y HTTPS.

Instrucciones

1. **Actualizar site.yml.** Agrega la importación de ansible/ejercicio2/main.yml.
2. **Crear las tareas en ansible/ejercicio2/main.yml.** Implementa tareas para:
 - a. Instalar Nginx.
 - b. Generar certificados SSL autofirmados.
 - c. Configurar Nginx para utilizar SSL.
 - d. Habilitar y configurar el firewall UFW para permitir tráfico en los puertos necesarios.

3. **Crear el manejador en handlers/main.yml.** Define un manejador para reiniciar Nginx cuando sea necesario.
4. **Crear la plantilla de configuración de Nginx.** Crea una plantilla que configure Nginx para utilizar los certificados SSL y servir en HTTPS.

Resultados esperados

- Nginx instalado y sirviendo en HTTPS.
- Acceso al sitio web mediante <https://localhost>.
- Firewall configurado adecuadamente.

Ejercicio 3: Despliegue de aplicación web con balanceador de carga (2 puntos)

Descripción

Despliega una aplicación web en múltiples instancias y configura un balanceador de carga.

- Implementar una aplicación Flask en varios puertos.
- Configurar Nginx como balanceador de carga para distribuir el tráfico entre las instancias de la aplicación.

Instrucciones

1. **Actualizar site.yml.** Agrega la importación de ansible/ejercicio3/main.yml.
2. **Crear las tareas en ansible/ejercicio3/main.yml.** Implementa tareas para:
 - a. Instalar dependencias de Python.
 - b. Instalar Flask y Gunicorn.
 - c. Crear directorio y copiar la aplicación Flask.
 - d. Crear servicios systemd para cada instancia de la aplicación en puertos diferentes.
 - e. Iniciar y habilitar los servicios de la aplicación.
 - f. Configurar Nginx como balanceador de carga.
3. **Crear las plantillas necesarias.**
 - a. Plantilla de la aplicación Flask.
 - b. Plantilla para los archivos de servicio systemd.
 - c. Plantilla de configuración de Nginx para balanceo de carga.

Resultados esperados

- Aplicación Flask corriendo en múltiples instancias en puertos diferentes.
- Nginx balanceando el tráfico entre las instancias.
- Acceso al servicio a través de <http://localhost>.

Ejercicio 4: Monitoreo y alertas (2 puntos)

Descripción

Implementa soluciones de monitoreo y alertas:

- Instalar y configurar Prometheus y Grafana.
- Configurar alertas básicas en Prometheus.
- Integrar métricas de la aplicación Flask.

Instrucciones

1. **Actualizar site.yml.** Agrega la importación de ansible/ejercicio4/main.yml.
2. **Crear las tareas en ansible/ejercicio4/main.yml.** Implementa tareas para:
 - a. Instalar Prometheus y Node Exporter.
 - b. Configurar Prometheus para recopilar métricas de los servicios.
 - c. Instalar Grafana y configurar su servicio.
 - d. Configurar alertas en Prometheus.
3. **Crear el manejador para reiniciar Prometheus.**
4. **Modificar la aplicación Flask para exponer métricas.** Actualiza la aplicación para que exponga métricas compatibles con Prometheus.
5. **Asegurarte de instalar las dependencias necesarias.**

Resultados esperados

- Prometheus recolectando métricas de las instancias de Flask.
- Grafana mostrando dashboards de monitoreo.
- Alertas configuradas y funcionando.

Ejercicio 5: Implementación de alta disponibilidad y recuperación de desastres (2 puntos)

Descripción

Mejora la infraestructura para soportar fallos y desastres:

- Configurar replicación de datos (por ejemplo, si usas una base de datos).
- Implementar backups automáticos.
- Configurar failover automático para la aplicación web.

Instrucciones

1. **Actualizar site.yml.** Agrega la importación de ansible/ejercicio5/main.yml.
2. **Crear las tareas en ansible/ejercicio5/main.yml.** Implementa tareas para:
 - a. Configurar una base de datos con replicación (ejemplo: PostgreSQL).
 - b. Implementar scripts de backup y restauración automáticos.

- c. Configurar un mecanismo de failover (por ejemplo, Keepalived) para proporcionar alta disponibilidad.
3. **Crear los scripts necesarios.**
 - a. Scripts de backup y restauración.
 - b. Archivos de configuración para el failover.

Resultados esperados

- Datos replicados y seguros.
- Backups automáticos funcionando.
- Failover automático en caso de fallo de una instancia.

Ejercicio 6: Seguridad avanzada y cumplimiento (2 puntos)

Descripción

Aumenta la seguridad del sistema y asegura el cumplimiento de políticas:

- Implementar políticas de seguridad con Ansible.
- Configurar y endurecer AppArmor o SELinux.
- Escanear el sistema en busca de vulnerabilidades.

Instrucciones

1. **Actualizar site.yml.** Agrega la importación de ansible/ejercicio6/main.yml.
2. **Crear las tareas en ansible/ejercicio6/main.yml.** Implementa tareas para:
 - a. Instalar herramientas de análisis de seguridad (por ejemplo, Lynis).
 - b. Ejecutar escaneos de seguridad y generar informes.
 - c. Endurecer la configuración de servicios clave (por ejemplo, Nginx, SSH).
 - d. Asegurar permisos y propiedades de archivos críticos.
 - e. Configurar políticas de seguridad del sistema (por ejemplo, restricciones de usuarios, políticas de contraseñas).
3. **Crear los manejadores necesarios.**
 - a. Manejadores para reiniciar servicios tras cambios en configuraciones de seguridad.
4. **Crear archivos de configuración y políticas.**
 - a. Archivos de configuración para AppArmor o SELinux.
 - b. Plantillas para configuraciones endurecidas.

Resultados esperados

- Sistema más seguro y resistente a ataques.
- Cumplimiento de políticas y estándares de seguridad.
- Reportes de escaneos de vulnerabilidades disponibles.

Ejercicio 7: Integración de pruebas automatizadas con Pytest (2 puntos)

Descripción

Añade pruebas automatizadas para la aplicación Flask utilizando Pytest.

- Escribe pruebas unitarias y de integración para las rutas y funcionalidades de la aplicación.
- Configura un entorno de pruebas aislado.
- Integra las pruebas en el proceso de despliegue automatizado con Ansible.

Instrucciones

1. **Actualizar site.yml:** Agrega la importación de ansible/ejercicio7/main.yml.
2. **Crear las tareas en ansible/ejercicio7/main.yml** Implementa tareas para:
 - a. Instalar Pytest y cualquier dependencia necesaria.
 - b. Copiar los archivos de pruebas al directorio adecuado.
 - c. Ejecutar las pruebas automáticamente después de desplegar la aplicación.
 - d. Configurar reportes de pruebas y manejo de resultados (por ejemplo, guardando los resultados en un directorio específico o enviando notificaciones si las pruebas fallan).
3. **Escribir los casos de prueba**
 - a. Define casos de prueba que cubran las diferentes rutas y funcionalidades de la aplicación Flask.
 - b. Asegúrate de cubrir escenarios de éxito y manejo de errores.
4. **Integrar Pytest con Ansible**
 - a. Configura tareas en el playbook para ejecutar las pruebas y manejar los resultados.
 - b. Implementa lógica para detener el despliegue si las pruebas fallan.

Resultados esperados

- Pruebas automatizadas que validan la funcionalidad de la aplicación.
- Integración de las pruebas en el proceso de despliegue continuo.
- Reportes de pruebas disponibles para revisión.

Ejercicio 8: Contenerización de la aplicación con Docker (2 puntos)

Descripción

Empaque la aplicación Flask en un contenedor Docker.

- Escribe un Dockerfile que construya la imagen de la aplicación.
- Configura un registro de contenedores local o utiliza un registro público.
- Modifica el despliegue para usar contenedores en lugar de servicios systemd.

Instrucciones

- 1. Crear el Dockerfile**
 - a. Define las etapas necesarias para construir la imagen de la aplicación.
 - b. Incluye instalación de dependencias y configuración de la aplicación.
- 2. Actualizar site.yml** Agrega la importación de ansible/ejercicio8/main.yml.
- 3. Crear las tareas en ansible/ejercicio8/main.yml** Implementa tareas para:
 - a. Instalar Docker en el sistema.
 - b. Construir la imagen de Docker utilizando el Dockerfile.
 - c. Ejecutar contenedores de la aplicación en los puertos necesarios.
 - d. Configurar Nginx para balancear carga entre los contenedores.
- 4. Gestionar imágenes y contenedores**
 - a. Implementa tareas para gestionar versiones de imágenes.
 - b. Configura limpieza automática de imágenes y contenedores antiguos.
- 5. Integrar con el proceso de despliegue**
 - a. Asegúrate de que el despliegue con Ansible ahora utiliza contenedores Docker en lugar de servicios systemd.

Resultados esperados

- Aplicación Flask empaquetada en una imagen Docker.
- Despliegue de la aplicación utilizando contenedores Docker.
- Nginx balanceando carga entre los contenedores en ejecución.

Ejercicio 9: Orquestación de contenedores con Docker Compose (2 puntos)

Descripción

Utiliza Docker Compose para orquestar los servicios de la aplicación.

- Define un archivo docker-compose.yml que describa la aplicación, base de datos y otros servicios.
- Modifica el despliegue para utilizar Docker Compose con Ansible.

Instrucciones

- 1. Crear el archivo docker-compose.yml**
 - a. Describe los servicios necesarios: aplicación Flask, base de datos, etc.
 - b. Configura redes y volúmenes según sea necesario.
- 2. Actualizar site.yml** Agrega la importación de ansible/ejercicio9/main.yml.
- 3. Crear las tareas en ansible/ejercicio9/main.yml** Implementa tareas para:
 - a. Instalar Docker Compose en el sistema.
 - b. Copiar el archivo docker-compose.yml al servidor.
 - c. Ejecutar docker-compose up para levantar los servicios.

- d. Manejar actualizaciones y reinicios de los servicios.

4. Gestionar la configuración

- a. Utiliza variables y plantillas en Ansible para parametrizar el archivo docker-compose.yml.

Resultados esperados

- Servicios de la aplicación orquestados con Docker Compose.
- Despliegue simplificado y gestión de múltiples contenedores.
- Posibilidad de escalar servicios fácilmente.

Ejercicio 10: Implementación de Integración Continua con Pytest y Docker

Descripción

Integra las pruebas automatizadas y la contenerización en un pipeline de Integración Continua (CI).

- Configura una herramienta de CI como GitHub Actions.
- Automatiza la ejecución de pruebas con Pytest y la construcción de imágenes Docker.
- Despliega automáticamente la aplicación si las pruebas pasan.

Instrucciones

1. Configurar la herramienta de CI

- a. Crea un archivo de configuración para la herramienta seleccionada.
- b. Define los stages/pasos del pipeline: pruebas, construcción, despliegue.

2. Integrar con Ansible y Docker

- a. Configura el pipeline para que ejecute los playbooks de Ansible.
- b. Asegura que las credenciales y accesos necesarios estén configurados de manera segura.

3. Automatizar el proceso

- a. Establece triggers para que el pipeline se ejecute en eventos específicos (por ejemplo, push a una rama determinada).

4. Manejar notificaciones y reportes

- a. Configura la herramienta de CI para notificar resultados de las pruebas y despliegues.
- b. Asegura que los logs y reportes estén disponibles para revisión.

Resultados esperados

- Pipeline de CI que automatiza pruebas, construcción y despliegue.
- Mayor eficiencia y confiabilidad en el proceso de desarrollo y despliegue.
- Visibilidad del estado del proyecto y rápida detección de problemas.

Indicaciones generales

- **Handlers:** En la carpeta handlers/, define los manejadores necesarios para reiniciar servicios cuando haya cambios en configuraciones.
- **Variables y plantillas:** Utiliza variables y plantillas (templates/) para hacer tu playbook más dinámico y reutilizable.
- **Roles (opcional):** Considera utilizar roles de Ansible para organizar mejor tus tareas y reutilizar código.
- **Documentación:** Documenta tus tareas y playbooks para facilitar su comprensión y mantenimiento.