CI/CD con Jenkins

Open Bokeron
openbokeron.uma.es
18/01/2026

Agenda

1. Introducción
2. Instalación inicial
3. Parte práctica
4. CI/CD desde fuera (como dev)
5. Consideraciones y curiosidades
6. Cierre

Somos esta gente

O sea, no somos todos esos de la foto. Somos los frikis del fondo.

Open Bokeron

- Asociación de software libre de la ETSII (UMA).
- Hacemos cosas #HazCosas
- Web: openbokeron.uma.es

➥ Objetivo del taller: entender un correcto flujo de desarrollo, CI/CD y perderle miedo a Jenkins de forma práctica usando un proyecto base (FastAPI + Svelte).

Y yo soy este

- Amin hazme el favor y pon cosas sobre ti
- Toma Marcos, tanto que decías que no te gustaba esta foto
- No sé qué más poner, así que: ¿Cómo hacer una buena bechamel? Las proporciones correctas para obtener una bechamel perfecta, son las siguientes: utiliza el mismo peso de harina que de grasa y la proporción de ingredientes líquidos con ingredientes secos es de 9<:1.

Qué es CI/CD?

- **CI (Integración Continua)**: integrar cambios pequeños frecuentemente y validarlos automáticamente.
- **CD (Entrega/Despliegue Continuo)**: llevar el cambio a un entorno desplegable con un proceso reproducible.
- Se apoya en automatización: pipeline + artefactos versionados + entornos.

Ejecución del taller y consideraciones

- Se os da una **VM Ubuntu** con el repo clonado en local.
- Trabajaremos con:
  - `backend/` (FastAPI + pytest + ruff)
  - `frontend/` (Svelte/Vite + svelte-check)
  - pipelines: `Jenkinsfile.ci` y `Jenkinsfile.cd`

Jenkins vs GitHub Actions

Jenkins

- Muy flexible; se integra con casi cualquier cosa.
- Control total: agentes, credenciales, redes, storage.
- Contras: mantenimiento, plugins, upgrades y seguridad.

GitHub Actions

- Experiencia integrada con GitHub (PRs, checks, permisos).
- Menos ops; runners gestionados (o self-hosted pagando).
- Contras: lock-in, límites de ejecución, dependes de un externo, starvation.

Conceptos de Jenkins

Piezas

- **Controller**: coordina jobs, UI, credenciales y cola.
- **Nodo/Agente**: máquina (VM, contenedor, bare metal) donde *se ejecuta* el trabajo.
- **Executor**: # de trabajos simultáneos que un agente puede correr.
- **Workspace**: carpeta donde Jenkins hace el checkout y trabaja.

Pipeline

- **Job/Pipeline**: definición del flujo (en este taller: `Jenkinsfile.*`).
- **Stage**: fase visible (lint, test, build, deploy…).
- **Step**: acción concreta dentro de un stage.
- **Artefactos**: salidas versionadas (builds, reports) que Jenkins guarda/publica.

➥ En **nuestro caso**: Jenkins corre en **bare metal** y el **controller** y el **agente** son la **misma máquina**; los pipelines ejecutan comandos lanzando **contenedores Docker**.

Arquitectura del repo

- API: `backend/` (FastAPI)
- Front: `frontend/` (Svelte + Vite, base path `/taller/`)
- Orquestación: `docker-compose.yml` (frontend + backend)
- Jenkins:
  - Pipelines: `Jenkinsfile.ci` y `Jenkinsfile.cd`

➥ Importante: el frontend usa un **base path** `/taller/`.

Uf seguro que instalar el Jenkins ese es muy difícil...

¿Seguro?

- `sudo wget -O /etc/apt/keyrings/jenkins-keyring.asc https://pkg.jenkins.io/debian-stable/jenkins.io-2026.key`
- `echo "deb [signed-by=/etc/apt/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null`
- `sudo apt install jenkins`

Configuración inicial y plugins

- Completar el asistente inicial (admin + plugins sugeridos).
- Configurar credenciales.

Job de prueba

- Creamos un job simple para validar:
- Ejemplo de pipeline:

```
pipeline {
  agent any
  stages {
    stage('¿Ya está, no? Ya sé de DevOps') {
      steps {
        sh 'uname -a'
        sh 'docker version'
      }
    }
  }
}
```

No


#5
ene 15
21:04
No Changes
337ms
failed
#4
ene 15
21:02
No Changes
359ms
failed
#3
ene 15
21:01
No Changes
340ms
failed
#2
ene 15
20:59
No Changes
340ms
failed
#1
ene 15
20:58
No Changes
518ms
failed


Jenkin's pipeline is failing again.

Lo siento, os toca trabajar ahora

- Vamos a intentar compilar manualmente el proyecto y a levantarlo
- Para ello debería ser tan sencillo como leerse el README
- Una vez lo levantemos, intentemos automatizarlo para no perder el tiempo en despliegues

CI: qué hace Jenkinsfile.ci?

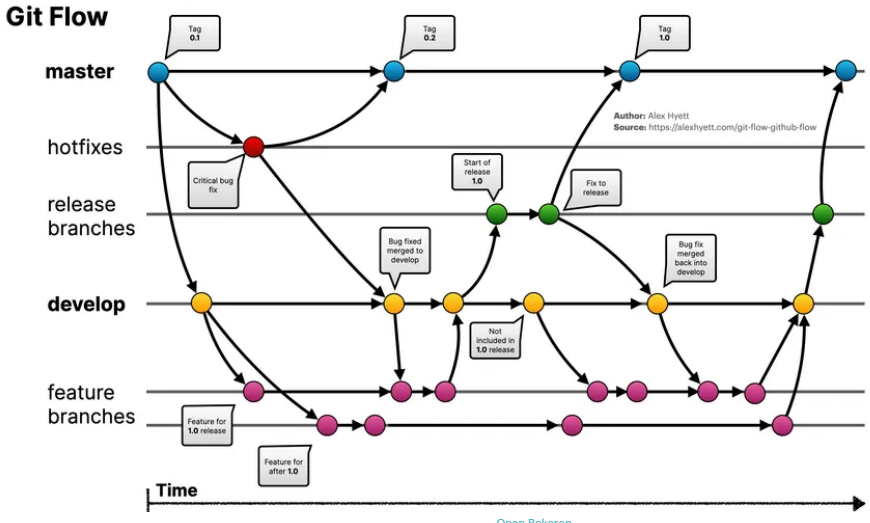
- Stage **Init**: lee autor y hash corto del commit.
- Stage **Backend: lint & test**:
  - usa contenedor `python:3.11-slim`
  - crea venv, instala dev deps
  - ejecuta `ruff` y `pytest`
- Stage **Frontend: check & build**:
  - usa contenedor `node:20-slim`
  - `npm install` + `npm run check` + `npm test` + `npm run build`

CD: qué hace Jenkinsfile.cd?

- Construye imágenes Docker con tags tipo `1.0.$BUILD_NUMBER`.
- Inyecta metadatos en runtime:
  - `APP_VERSION`, `GIT_COMMIT`, `COMMIT_AUTHOR`, `BUILD_NUMBER`
- Despliega con `docker compose up -d` usando `BACKEND_TAG` y `FRONTEND_TAG`.

➥ El rollback es sencillo: si tienes tags, puedes redeployar una versión anterior en segundos. CD genera artefactos (imágenes versionadas) y en nuestro caso, se ocupa del deploy también.

Flujo de trabajo - Git



Flujo de trabajo

- Crear rama: `feature/...`
- Hacer cambio pequeño y con sentido.
- Abrir Merge Request / Pull Request.
- Observar checks:
  - CI corre en cada push
  - el MR/PR muestra estado verde/rojo
- Si falla: arreglar y push de nuevo.
- Merge a main y observar despliegue (CD).

Propuesta de cambio para los asistentes

Cambio sugerido

- Cambiar el título del hero en `frontend/src/App.svelte`. Alguien puso "UwU".
- Nuevo texto sugerido: *Taller CI/CD con Jenkins*.

PR en Gitea

- Haréis la PR con un **usuario genérico** que os damos en nuestro **Gitea**.
- Al abrir la PR:
  - Jenkins lanza la **CI** (y la veréis en la web de Jenkins).
  - Aparece el generalísimo `jenkins-bot`.
- `jenkins-bot` comentará en la PR:
  - si todo ha ido bien: **verde** y a mergear.
  - si la habéis liado: a mirar el log.

Qué observar en Jenkins al hacer el MR/PR

- Log del stage que falla (si falla): ruff, pytest, svelte-check, build…
- En CD: tag de imagen generado (`APP_VERSION`) y deploy.
- En la app:
  - `GET /health` refleja build/commit/autor.
  - `GET /builds` refleja builds recientes via Jenkins REST.

Triggers: webhooks vs polling

Webhooks

- El repositorio ”avisa” a Jenkins.
- Rápido y eficiente.
- El repo le envía toda la información a Jenkins (autor, commit, rama, etc.)
- Eso tiene pinta de ser difícil... Jenkins tiene plugins para prácticamente cualquier cosa.

Polling

- Jenkins pregunta cada X tiempo.
- Absurdamente simple.
- Si abres una PR no es inmediato, debes esperar al polling.

Rollback con imágenes etiquetadas

- En CD generamos tags con `APP_VERSION = 1.0.$BUILD_NUMBER`.
- Si el deploy rompe algo, puedes volver atrás:
  - desplegar `BACKEND_TAG=1.0.41 FRONTEND_TAG=1.0.41` (ejemplo)
  - `docker compose up -d`

Jenkins REST API: ver CI/CD sin entrar al UI

- Jenkins expone una API JSON por job/build.
- En este repo, el backend consulta builds recientes y expone en el frontend las últimas 5 builds del CD.
- Ideas de uso:
  - dashboards del estado de las builds.
  - alertas.

Recap

- CI: automatiza validación (lint/tests/build) en cada cambio.
- CD: produce artefactos versionados y despliega de forma reproducible.
- Jenkins: flexible, potente, pero requiere mimos (seguridad, plugins, mantenimiento). Sin embargo, una vez lo tienes montado, el mantenimiento ya no es tan drama.
- Ahora podrás entrar en los pipelines de tu futuro puesto de trabajo sin miedo a romper nada... ¿Verdad?

Preguntas y siguientes pasos

- Preguntas.
- ¿Ampliamos taller? Se nos ha quedado en el tintero...
  - Registry (Harbor/Docker Hub/GHCR)
  - SonarQube + quality gates
  - Despliegue con Quadlets usando Podman
  - Kubernetes (no)

Encuesta de satisfacción

¡Muchas Gracias!

https://openbokeron.uma.es

@OpenBokeron