Informe Técnico: Infraestructura de Despliegue Automático Arquitectura de CI/CD (Integración y Despliegue Continuo) Personalizado con los requerimientos de larvosys.

1. Resumen General

Se ha implementado un entorno de desarrollo y despliegue basado en **Docker**, que permite a los desarrolladores subir su código mediante **Git** y verlo desplegado automáticamente en un servidor de aplicaciones **WildFly 30**, sin intervención manual del administrador,así también se se encuentra funcional postgres 17 con pgadmin 4 para el uso visual menos complicado.

2. Componentes del Ecosistema (Stack)

Componente	Función	Acceso / Puerto
Gitea	Servidor de Git privado para control de versiones.	http://194.5.152.156:3000
WildFly 30	Servidor de aplicaciones Java (Jakarta EE).	http://194.5.152.156:8080
PostgreSQL	Base de Datos relacional para las aplicaciones.	Puerto 5432
pgAdmin 4	Gestión visual de la base de datos.	http://194.5.152.156:5050

---

3. Auto-Deploy Hook

La pieza clave es un script de **Git Hook (post-receive)** configurado dentro de Gitea.

- **¿Cómo funciona?**: Cada vez que el desarrollador hace un git push, el script detecta automáticamente si hay un archivo .war.
- **Acción**: Extrae el contenido binario del commit y lo deposita en la carpeta /home/deployments/ del VPS.
- **Sincronización**: Esta carpeta está vinculada ("mapeada") directamente con el contenedor de WildFly, logrando un despliegue inmediato.

4. Credenciales y Claves de Acceso

Servicio	URL de Acceso	Usuario	Contraseña
Gitea (Git)	http://194.5.152.156:3000	rodrigo15jul67@hotmail.com Rodrigo	Larvosys-2025GIT
pgAdmin (DB)	http://194.5.152.156:5050	admin@larvosys.com	Larvosys-2025PG
WildFly Console	http://194.5.152.156:9990	Admin richar	Larvosys-2025WF larvosysdev@
Postgres	postgres_db	postgres	Larvosys-2025DB

docker exec -it wildfly_server /opt/jboss/wildfly/bin/add-user.sh admin Larvosys-2025WF --silent

5. Rutas Críticas en el VPS

- **Archivos de configuración:** /root/entorno-dev/ (Aquí están los docker-postgres
- **Carpeta de Despliegue:** /home/deployments/ (Aquí caen los archivos .war).
- **Logs de Gitea:** /root/entorno-dev/gitea_data/

6. Instrucciones para el Desarrollador

Para que el sistema funcione, el desarrollador debe seguir estas reglas:

1. **Crear Repositorio:** Siempre usar la **Plantilla** (admin/plantilla-java) en Gitea para que el Hook se active.
2. **Subir el archivo:** El archivo .war debe ir en la raíz del repositorio o dentro del proyecto.

3. **URL de la App:** Si el archivo se llama sistema.war, la aplicación será accesible en http://194.5.152.156:8080/sistema/.

Ventajas de solución propuesta

- **Automatización:** El desarrollador no pierde tiempo subiendo archivos por FTP o SSH; solo hace `git push`.
- **Aislamiento (Docker):** Si algo falla en WildFly, no daña el resto del servidor. Es fácil de reiniciar y recuperar.
- **Independencia:** el servidor de Git propio (Gitea), ellos son dueños de su código y su infraestructura, sin depender de servicios externos costosos.

7. Interfaz Gráfica de Administración (Escritorio Remoto) Se ha habilitado un entorno de escritorio (GUI) accesible vía **RDP** (Escritorio Remoto de Windows). Esto permite realizar tareas administrativas de archivos y navegación interna de forma visual, facilitando la curva de aprendizaje para el personal no técnico.

He optado por XFCE porque es un entorno de alto rendimiento que no sacrifica la potencia del servidor. Esto les da una interfaz amigable para gestionar sus archivos sin lentitud, manteniendo toda la capacidad del VPS para las aplicaciones Java.

Conclusión

El proyecto se entregó bajo una arquitectura de micro-servicios mediante contenedores. Esto significa que en un solo VPS robusto tienen la potencia de lo que antes requeriría 5 máquinas. He automatizado el flujo de trabajo para que el despliegue sea instantáneo vía Git, reduciendo el margen de error humano y eliminando la necesidad de un administrador de sistemas dedicado para subir archivos cada vez.

Recomendaciones

Actualmente el acceso es por IP y puertos. Se recomienda la implementación de un Proxy Inverso con SSL para que el acceso sea mediante un dominio (larvosys.com) y cuente con cifrado de seguridad HTTPS, lo cual protege las credenciales y el código fuente".

Git

git init
git add .
git remote add origin http://194.5.152.156:3000/Rodrigo/prueba-1.git
git commit -m "comentario"
git push origin master

windows
git config --global http.postBuffer 524288000

jar -cvf birt2.war -C birt2.war/ .

Guía de Comandos: Sistema LarvoSys

Para gestionar el entorno de desarrollo y producción, utiliza el comando maestro `larvosys`. No es necesario acceder a las capas internas del sistema; todo se controla desde esta interfaz.

1. Monitoreo de logs (En tiempo real)

Ideal para debuggear errores de despliegue o verificar transacciones en la base de datos.

- **`larvosys logs wildfly`**: Muestra los logs del servidor de aplicaciones (JBoss/WildFly). Úsalo para ver si tu `.war` o `.ear` se desplegó correctamente.
- **`larvosys logs postgres`**: Muestra los logs de la base de datos. Úsalo para verificar conexiones y errores de SQL.

2. Gestión de Servicios (Reinicios)

Si el servidor se queda "pegado" o necesitas aplicar cambios de configuración profunda.

- **`larvosys restart wildfly`**: Reinicia únicamente el contenedor de WildFly sin afectar la base de datos.
- **`larvosys restart postgres`**: Reinicia el motor de base de datos.
- **`larvosys restart all`**: Realiza un reinicio completo de toda la infraestructura (Gitea, WildFly, Postgres, pgAdmin).

3. Mantenimiento y Limpieza

WildFly a veces deja rastro de despliegues fallidos que impiden subir versiones nuevas.

- **`larvosys clean`**: Borra automáticamente todos los archivos `.failed`, `.deployed` y carpetas temporales de la ruta `/home/deployments`. **Úsalo siempre antes de subir un nuevo archivo de despliegue.**

4. Estado del Sistema

- **`larvosys status`**: Muestra una tabla con los servicios activos, su tiempo de encendido y los puertos en los que están escuchando.
- **`larvosys stats`**: Muestra el consumo real de **CPU y Memoria RAM** de cada servicio. Útil para detectar fugas de memoria en la aplicación.

Flujo de Trabajo Recomendado

Para subir una nueva versión de la aplicación sin errores, el desarrollador debería seguir este orden:

1. **Limpiar el entorno:** `larvosys clean`
2. **Subir el archivo:** Colocar el `.ear` o `.war` en `/home/deployments`.
3. **Monitorear:** Ejecutar `larvosys logs wildfly` para confirmar que el servidor reconoce el archivo y lanza el mensaje `Deployed`.

Manejo de Gittea

1) Requisitos previos (local)

1. Instalar Git:
    - Windows: Git for Windows
    - macOS: `brew install git`
    - Linux: `sudo apt install git` / `sudo dnf install git`
2. Configurar identidad (una sola vez):

```
git config --global user.name "Tu Nombre"
git config --global user.email "tu_correo@dominio.com"
```

2) Crear un repositorio en Gitea (web)

1. Entra a tu Gitea (ej: `https://gitea.tuempresa.com`)
2. **New Repository**
3. Define:
    - **Owner** (tu usuario o una org)
    - **Repository Name**
    - Visibilidad: **Private/Public**
    - Opcional: inicializar con **README**, `.gitignore`, `LICENSE`
4. Crear.

Resultado: Gitea te mostrará la URL para clonar por **HTTPS** y/o **SSH**.

3) Autenticación recomendada: SSH

3.1 Crear llave SSH

```
ssh-keygen -t ed25519 -C "tu_correo@dominio.com"
```

Enter para aceptar ruta por defecto, y (opcional) passphrase.

3.2 Cargar la llave pública en Gitea

1. Copia tu clave pública:

```
cat ~/.ssh/id_ed25519.pub
```

2. En Gitea: **Settings → SSH Keys → Add Key**
3. Pega y guarda.

3.3 Probar conexión

```
ssh -T git@tu-dominio-gitea
```

4) Clonar el repo y primer commit

4.1 Clonar

```
git clone git@tu-dominio-gitea:owner/nombre-repo.git
cd nombre-repo
```

4.2 Crear un archivo y commitear

```
echo "# Mi Proyecto" > README.md
git add README.md
git commit -m "Agrega README inicial"
git push -u origin main
```

> Nota: algunas instalaciones usan `master` en vez de `main`.

5) Flujo básico día a día (cambios normales)

5.1 Traer cambios del servidor

```
git pull
```

5.2 Ver estado

```
git status
git log --oneline --decorate -n 10
```

5.3 Agregar cambios y push

```
git add .
git commit -m "Describe claramente el cambio"
git push
```

6) Flujo recomendado en equipo: ramas + Pull Request

6.1 Crear rama de feature

```
git checkout -b feature/login
```

6.2 Trabajar y subir

```
git add .
git commit -m "Implementa pantalla de login"
git push -u origin feature/login
```

6.3 Abrir Pull Request en Gitea

En la web del repo:

- **Pull Requests → New Pull Request**
- base: `main`
- compare: `feature/login`
- Describe el cambio, añade reviewers, labels, etc.

6.4 Actualizar tu rama con main (si main avanzó)

Opción A (merge):

```
git checkout feature/login
git pull
git merge origin/main
git push
```

Opción B (rebase, más limpio si tu equipo lo usa):

```
git checkout feature/login
git fetch origin
git rebase origin/main
git push --force-with-lease
```

> Regla: **rebase** solo si tu política lo permite y entienden el `--force-with-lease`.

7) Tags y releases (versiones)

7.1 Crear tag

```
git tag -a v1.0.0 -m "Release 1.0.0"
git push origin v1.0.0
```

7.2 Release en Gitea

En la web:

- **Releases → New Release**
- Selecciona el tag `v1.0.0`
- Notas de versión

8) Issues (tareas/bugs) y buenas prácticas

En Gitea:

- **Issues → New Issue**
    - Título claro
    - Pasos para reproducir (si bug)
    - Criterios de aceptación (si feature)
    - Labels: `bug`, `enhancement`, `priority`
    - Milestones (si manejan sprints)

Práctica útil: vincular commits/PRs a issues:

- En mensajes/descr: `Fixes #12` o `Closes #12` (según soporte/config), para autocerrar al merge.

9) Convenciones recomendadas (para que el versionamiento “sirva”)

9.1 Mensajes de commit (formato simple)

- `feat: agrega autenticación`
- `fix: corrige validación de email`
- `docs: actualiza README`
- `refactor: reorganiza servicios`

9.2 Estructura de ramas

- `main`: estable
- `develop` (opcional): integración
- `feature/*`: nuevas funciones
- `fix/*` o `hotfix/*`: correcciones

9.3 Protecciones en Gitea (admin)

- Proteger `main`:
    - Requerir PR
    - Requerir approvals
    - Bloquear force push
    - Checks/CI (si usan Actions/CI)

10) Problemas comunes y soluciones rápidas

“Me pide usuario/contraseña y falla por HTTPS”

- En muchas configs modernas, HTTPS requiere **token** en vez de contraseña.
- Recomendación: **usar SSH** (sección 4).

“Mi rama local no sube”

```
git push -u origin nombre-rama
```

“Conflictos al hacer pull/merge”

1. Abre archivos en conflicto (marcados por Git)
2. Resuelve manualmente
3. Luego:

```
git add .
git commit
git push
```

11) Mini guía de comandos esenciales (cheat sheet)

```
git clone <url>
git status
git add <archivo> | git add .
git commit -m "mensaje"
git pull
git push
git checkout -b feature/x
git checkout main
git merge feature/x
git log --oneline --graph --decorate --all
```