XPipe – Architettura e Standard Tecnici

# Architettura XPipe: Panoramica Completa

## 1. Introduzione

XPipe è una piattaforma PaaS modulare e altamente scalabile, progettata per fornire servizi DevOps, orchestrazione, automazione e gestione dei workload cloud attraverso un'infrastruttura basata su Kubernetes e una solida architettura a microservizi.

## 2. Monorepo e Domain Driven Design (DDD)

La struttura principale del codice è organizzata come un \*\*monorepo\*\* seguendo principi di \*\*Domain Driven Design\*\*:

### 2.1 Struttura a Tre Livelli

- \*\*Livello 0: Business Domain\*\* – rappresenta l’area logica di competenza (es. `Core`, `DevOps`, `CMDB`, `Security`, `Docs`, `School`)

- \*\*Livello 1: Role\*\* – identifica la funzione o categoria (es. `Module`, `API`, `Base`, `Job`)

- \*\*Livello 2: Virtual Repository\*\* – rappresenta l’unità logica atomica (es. `Auth`, `Policies`, `Pandoc`, `Courses`)

Esempi:

- `Core -> Module -> Auth`

- `DevOps -> API -> Pipelines`

- `School -> Module -> Courses`

## 3. Naming Convention e Slug

Ogni componente viene identificato tramite uno \*\*slug semantico e posizionale\*\*, conforme a:

[product]-[domain]-[role]-[name]

- Il nome è obbligatoriamente in lowercase, concatenato senza separatori interni

- `product` è opzionale (es. `xpipe`, `school`)

- `role` può avere lunghezza fissa a 3 caratteri per standardizzazione (`api`, `mdl`, `bas`)

- Esempio completo: `xpipe-core-api-usersessionhandler`

## 4. Github Actions: Workflow e Job

Il sistema CI/CD si basa su \*\*Github Actions\*\* con 3 tipologie principali di file:

### 4.1 Tipologie

- \*\*Workflow specifici\*\*: per ciascun componente (`core-ci-ecommerce-void.yml`)

- \*\*Job riutilizzabili\*\*: prefisso `job-` (es. `job-ci-build-container.yml`)

- \*\*Template dinamici\*\*: prefisso `template-` per orchestrazione esecuzioni (es. `template-no-infra-eks.yml`)

### 4.2 Esecuzione Pipeline Dinamiche

- Il backend di esecuzione è Github stesso

- I template possono essere clonati in repo dinamici o eseguiti da un repository singleton

- Input via `workflow\_dispatch` per ogni esecuzione con `env` posix-style uppercased

## 5. Pipeline Input e Ambiente

Ogni pipeline può leggere file `.env` personalizzati da:

[BusinessDomain]/[Role]/[RepoSlug]/config/[RepoSlug].env

Esempio:

Core/API/core-api-policies/config/core-api-policies.env

## 6. Considerazioni Finali

- Design altamente modulare e scalabile

- Naming posizionale per chiarezza, controllo e interoperabilità

- Supporto a più prodotti derivati (es. `school`, `books`)

- Standardizzazione che agevola troubleshooting, audit e automazione

\*\*Autore\*\*: Luca Pitzoi – Founder @ Sindria