Tecnologie PoC

Gruppo 19 - aLittleByte

Team: Bellet Eleonora, Belluz Diego, Gastaldello Angelica, Kaneda Fernandes Vinicius Yuri, Maule Angela, Oloni Alex, Soligo Leonardo

email: alittlebyte.swe@gmail.com

Contesto e obiettivo della PoC

Capitolato: C5
Proponente: *Eggon*

Il capitolato richiede l'evoluzione di una **piattaforma HR/documentale** tramite moduli intelligenti integrabili nel prodotto esistente.

Funzionalità principali del capitolato

- **AI Assistant Generativo**: supporto alla creazione di comunicazioni HR a partire da prompt, con personalizzazione di tono/stile e valutazione del risultato.
- **AI Co-Pilot per Consulenti del Lavoro**: gestione di documenti caricati nella dashboard, con OCR, riconoscimento dei destinatari, split dei PDF massivi ed estrazione dei dati utili.

Obiettivo tecnico della PoC

Validare un flusso end-to-end in cui tutte le componenti scelte cooperano in modo **riproducibile in locale** ed **osservabile** mantenendo una struttura coerente con successive integrazioni cloud reali.

aLittleByte

Ambiente locale e provisioning AWS-like


Docker


Make

Composer


LocalStack


Terraform


SSM


Secrets

L'obiettivo dell'ambiente locale è che ogni persona del team lavori sulla stessa identica configurazione, vicina a quella di produzione, senza dover creare account cloud o installare servizi a mano.

Container e comandi

Le immagini dell'applicazione e del web server sono costruite con **Docker**, con le dipendenze PHP gestite da **Composer**, e avviate da **Docker Compose** insieme a database e servizi di supporto su reti separate. I comandi ricorrenti passano da un unico **Make**file, così le procedure restano uguali per tutti.

Cloud emulato in locale

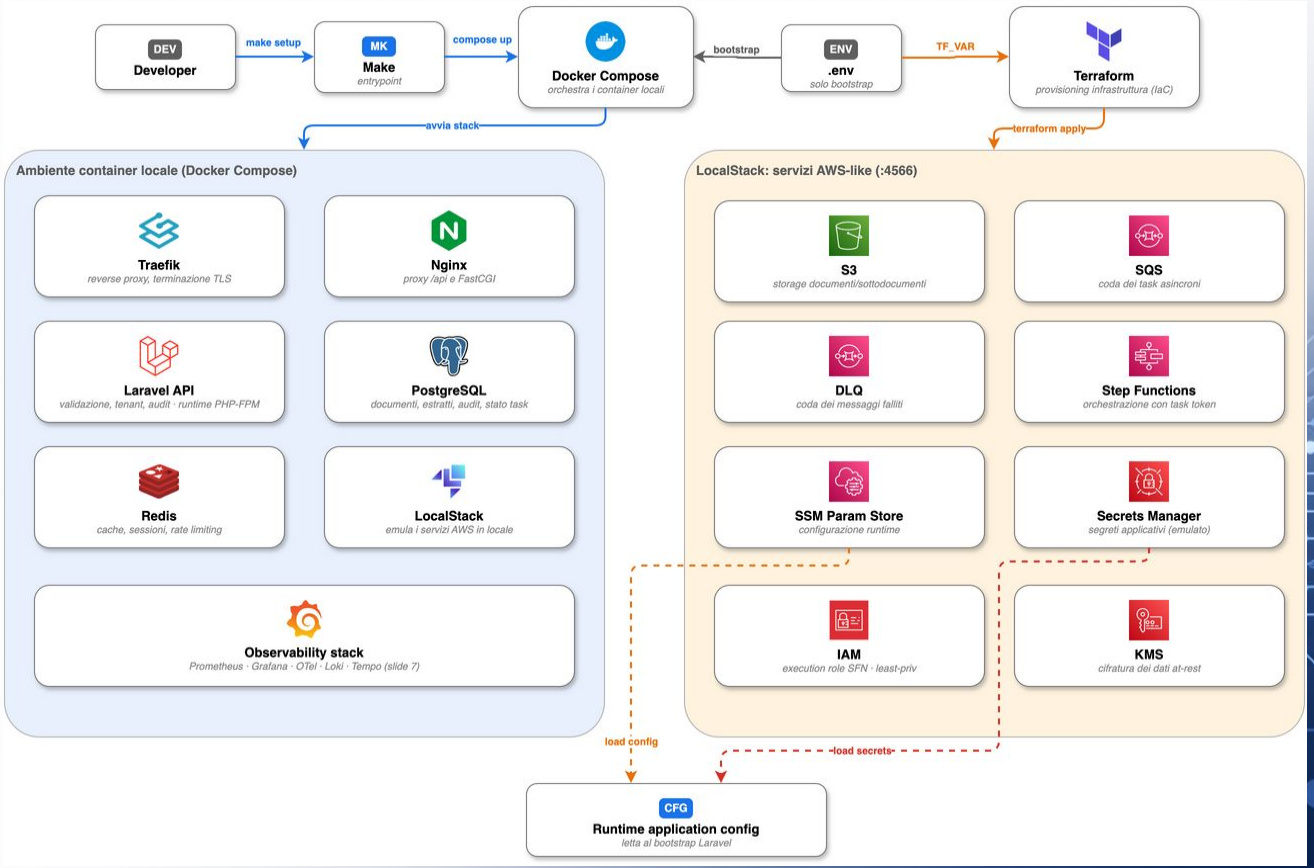
In sviluppo non usiamo davvero i servizi AWS: **LocalStack** li riproduce in locale, dallo storage alle code alla gestione dei segreti. In questo modo proviamo il flusso senza dipendere continuamente da risorse cloud attive o credenziali reali

Provisioning e segreti

L'infrastruttura è descritta con **Terraform** invece che con script manuali, quindi è versionata e ricreabile a comando. La configurazione non sensibile vive in **SSM** e i segreti in **Secrets Manager**, fuori dal codice.

aLittleByte

Dettaglio: Ambiente locale e provisioning AWS-like



aLittleByte

Frontend SPA e contratto API


Angular 21


TypeScript 5.9


Angular CLI


OpenAPI 3.1


Orval


Redocly


Jest


RxJS


Reactive Forms


lucide

Il frontend è una applicazione a pagina singola che non conosce a memoria gli endpoint del backend: si appoggia a un contratto condiviso, così frontend e backend rimangono allineati durante l'evoluzione del contratto API.

Interfaccia

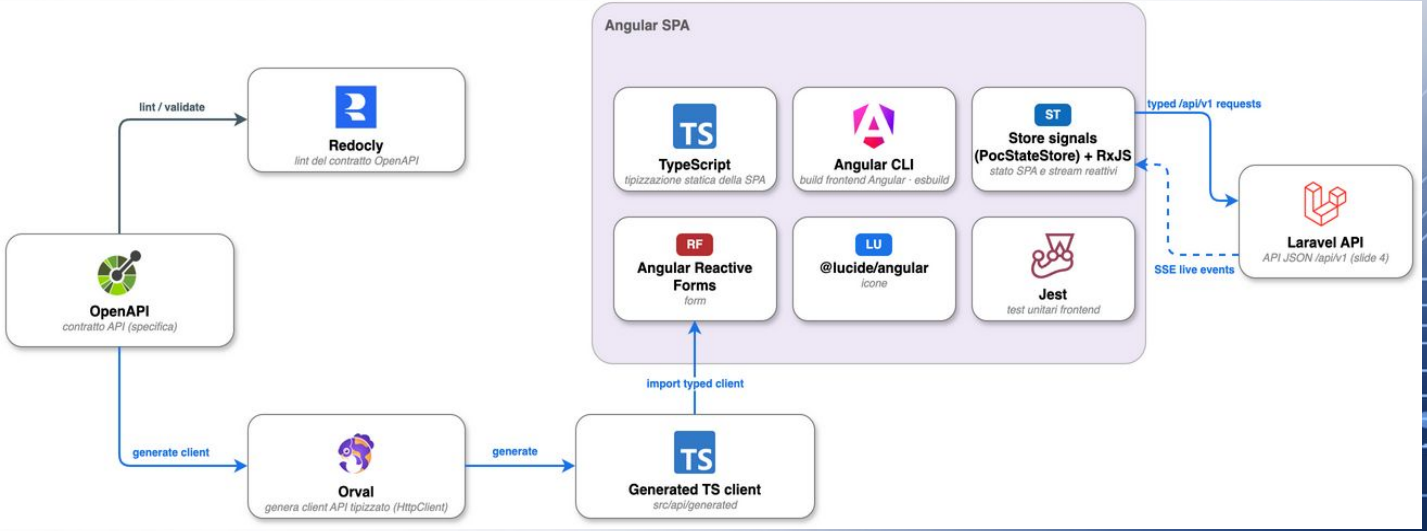
È costruita con **Angular 21** e **TypeScript 5.9**, con **Angular CLI** per sviluppo e build veloci; le icone vengono da **lucide**, i form da **Reactive Forms** e lo stato con signals + **RxJS**. Abbiamo scelto **Angular** per la struttura integrata e opinionata (DI, routing, reactive forms, signals), che dà coerenza e manutenibilità nel tempo.

Contratto e test

Il contratto **OpenAPI 3.1** è la fonte di verità da cui **Orval** genera il client e che **Redocly** valida, così non scriviamo le chiamate a mano. I test girano con **Jest** e gli aggiornamenti dal server arrivano via **SSE**.

aLittleByte

Dettaglio: Frontend SPA e contratto API



aLittleByte

Edge, runtime e backend API


Traefik


Nginx


PHP-FPM


PHP 8.4


Laravel 12


S3

Una richiesta HTTPS attraversa alcuni livelli prima di arrivare al codice, e ognuno ha un compito ben preciso.

Edge e web server

Traefik sta davanti a tutto: termina il TLS, smista le richieste in base all'host ed espone metriche già pronte. Dietro, **Nginx** inoltra le chiamate /api a PHP-FPM, mentre la SPA statica è servita da un'emulazione CDN locale che la legge dal bucket S3 (LocalStack), in produzione questo ruolo sarebbe svolto da AWS CloudFront.

Runtime PHP

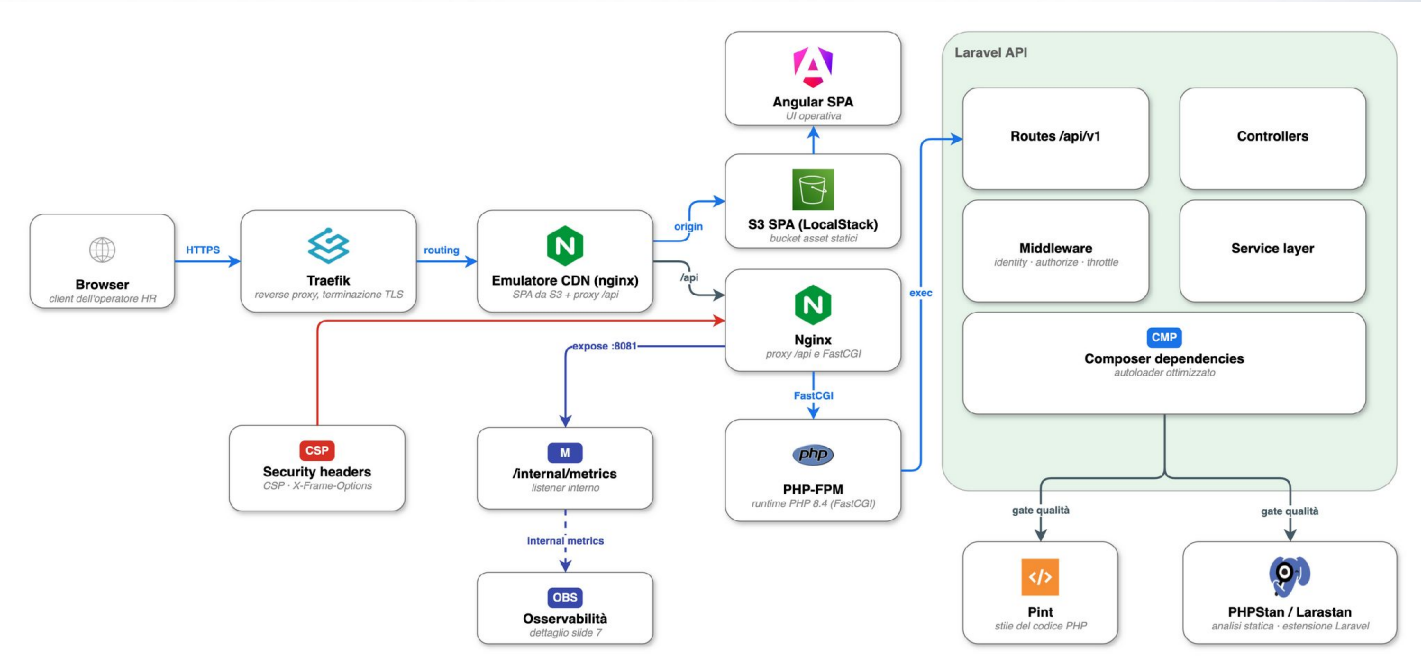
Il codice gira su **PHP 8.4** servito da **PHP-FPM**. Abbiamo scelto questo runtime classico invece di **FrankenPHP** o **Laravel Octane**, che tengono l'applicazione in memoria tra le richieste: più rapidi sulla carta, ma con processi sempre attivi da sorvegliare e un comportamento meno prevedibile per una PoC.

Backend

L'API JSON è realizzata con **Laravel 12**, scelto per la maturità del suo ecosistema rispetto ad alternative come Symfony o a framework di altri linguaggi.

aLittleByte

Dettaglio: Edge, runtime e backend API



aLittleByte

Dati, storage e protezione


PostgreSQL 16


Redis 7


AWS S3


KMS


IAM

I dati della PoC hanno nature diverse, perciò usiamo strumenti diversi invece di forzare tutto su un'unica tecnologia.

Dati applicativi

PostgreSQL 16 conserva i dati strutturati e, grazie a JSONB, anche i payload semi strutturati senza cambiare database. Lo abbiamo preferito a MySQL e a soluzioni documentali come MongoDB per affidabilità e flessibilità.

Stato volatile

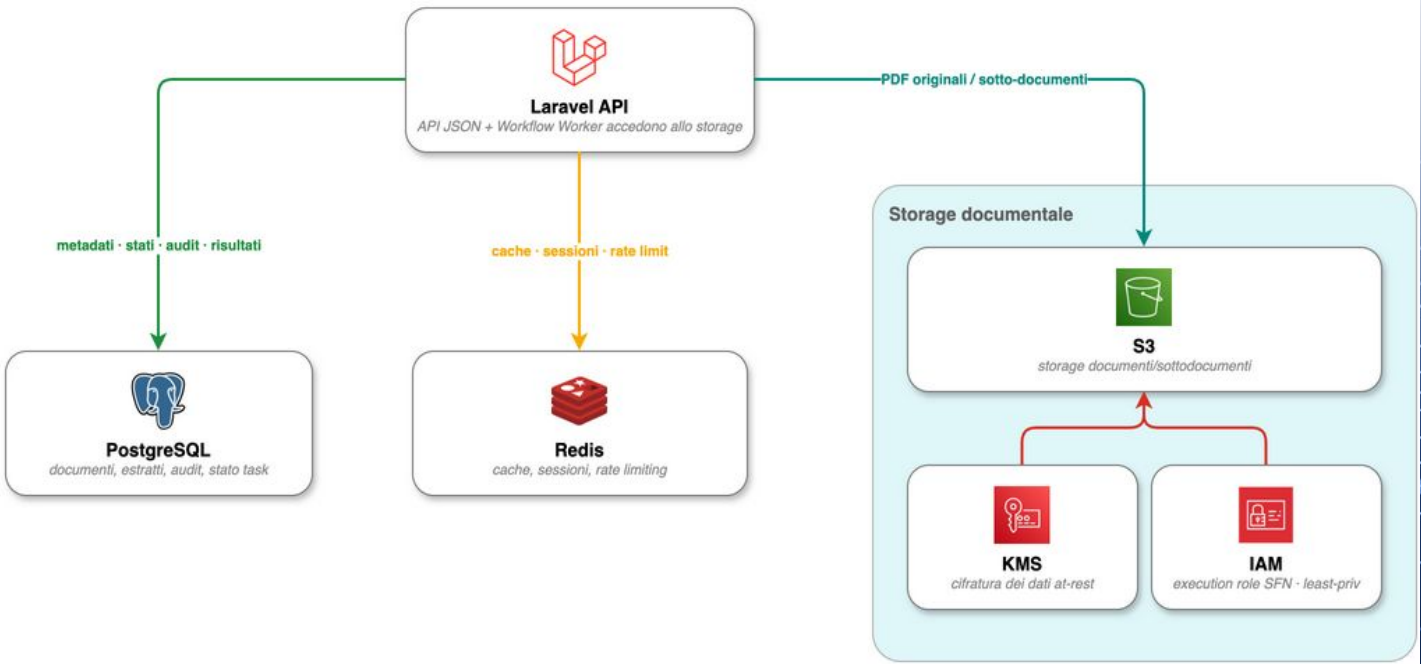
Redis 7 tiene ciò che è temporaneo: cache, sessioni e limiti di frequenza, sempre con una scadenza. Resta stato volatile e non viene usato come coda della pipeline, ruolo affidato a un servizio dedicato.

Documenti e protezione

I documenti del flusso AI/OCR sono archiviati su **AWS S3** (in locale lo stesso ruolo è coperto da LocalStack). Gli oggetti sono cifrati con **KMS** e l'accesso è regolato da policy **IAM**.

aLittleByte

Dettaglio: Dati, storage e protezione



aLittleByte

Pipeline documentale asincrona e integrazione AI/OCR

L'elaborazione dei documenti non avviene durante la richiesta dell'utente ma in modo asincrono, perché coinvolge passi lenti come OCR e generazione con i modelli.

Orchestrazione

Step Functions coordina i passaggi gestendo tentativi, timeout e stato. È un orchestratore e non una coda: lo abbiamo preferito a soluzioni self hosted come Temporal o Camunda per non doverle gestire noi.

Coda e worker

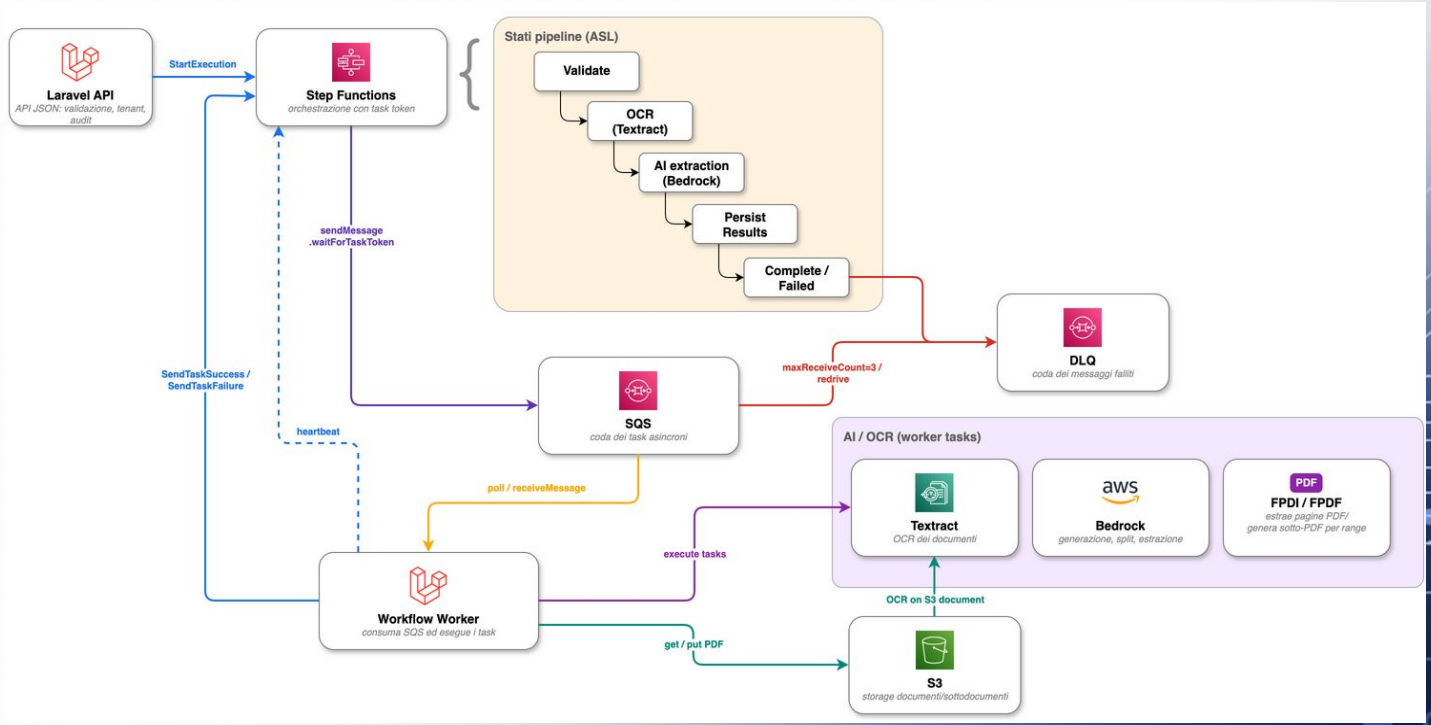
I task viaggiano su **SQS**, scelto al posto di una coda su Redis per la durabilità e per la gestione automatica dei messaggi falliti. Un **Worker** dedicato li consuma dalla coda e risponde all'orchestratore con un token di callback, restando idempotente perché lo stesso messaggio può arrivare più volte.

AI e OCR

L'estrazione e la generazione si appoggiano a servizi gestiti: **Bedrock** per i modelli e **Textract** per l'OCR, che legge i file da AWS S3. Sono stati preferiti ad alternative come OpenAI o Tesseract per integrazione e gestione.

aLittleByte

Dettaglio: Pipeline documentale asincrona e integrazione AI/OCR



aLittleByte

Osservabilità


OTel


Prometheus


Tempo


Loki


Alloy


Grafana


Alertmanager

Per capire cosa succede nel sistema raccogliamo metriche, tracce e log da un unico punto e li leggiamo in un'unica interfaccia, con allarmi legati al comportamento dell'applicazione.

Strumentazione

L'applicazione è strumentata con **OpenTelemetry** (**OTel**), uno standard indipendente dal fornitore: cambiando backend di osservabilità non toccheremmo il codice, e questo evita il vincolo a prodotti come Datadog o New Relic.

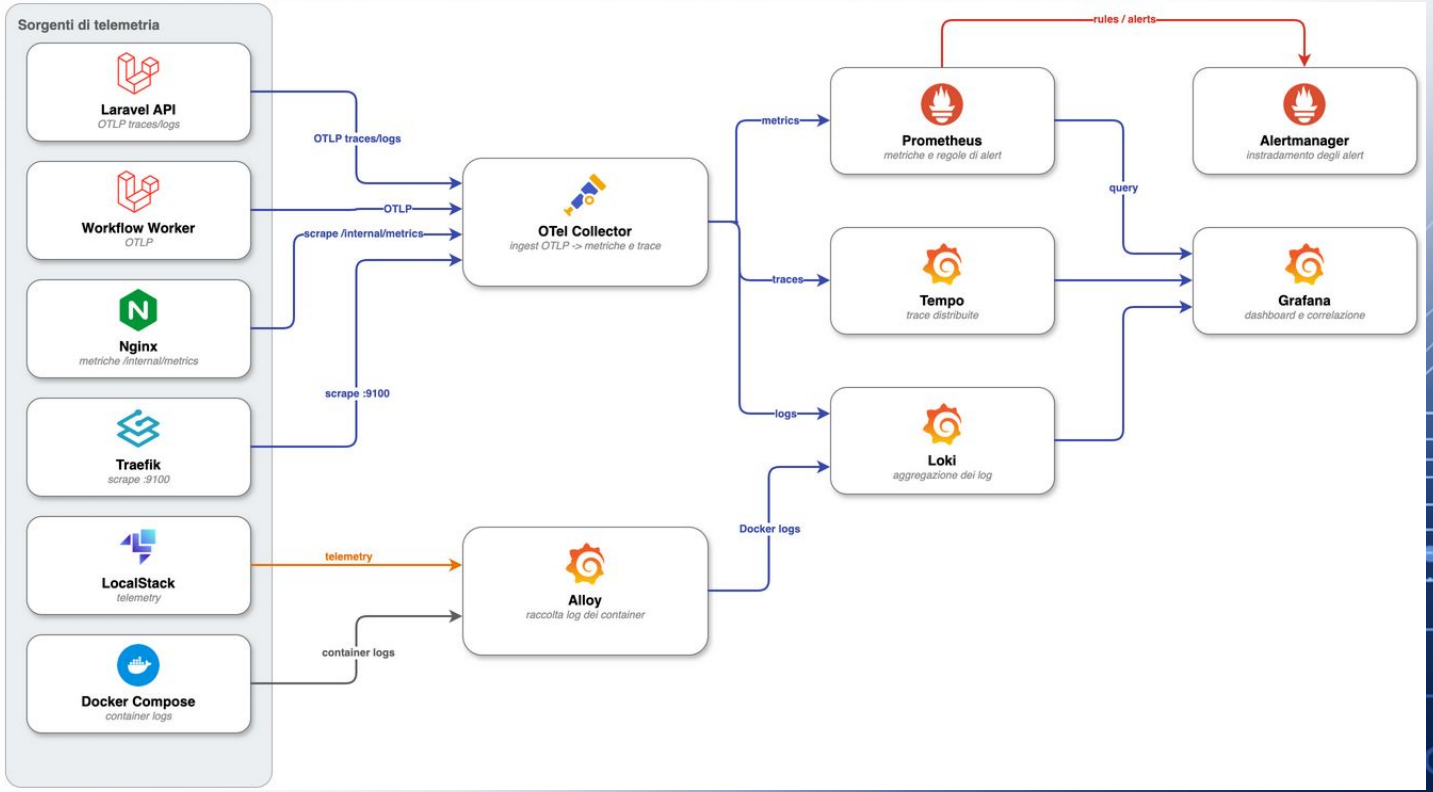
Raccolta e correlazione

Un **Collector** centrale smista i dati verso **Prometheus** per le metriche, **Tempo** per le tracce e **Loki** per i log, con **Alloy** che raccoglie i log dai container. **Grafana** li mette insieme in cruscotti correlati, invece di tenere stack separati come ELK o Jaeger.

Allarmi

Alertmanager genera gli avvisi a partire da regole definite su Prometheus, legate a eventi concreti del dominio più che a soglie generiche.

Dettaglio: Osservabilità



aLittleByte

CI e quality gate


GitHub Actions


GHCR


Docker Hub


Trivy


npm audit


Pint


PHPStan


Pest


axe


pa11y


Playwright


ESLint

Ogni modifica spinta sul repository ricostruisce le immagini e deve superare una serie di controlli automatici prima di poter essere pubblicata.

Controlli

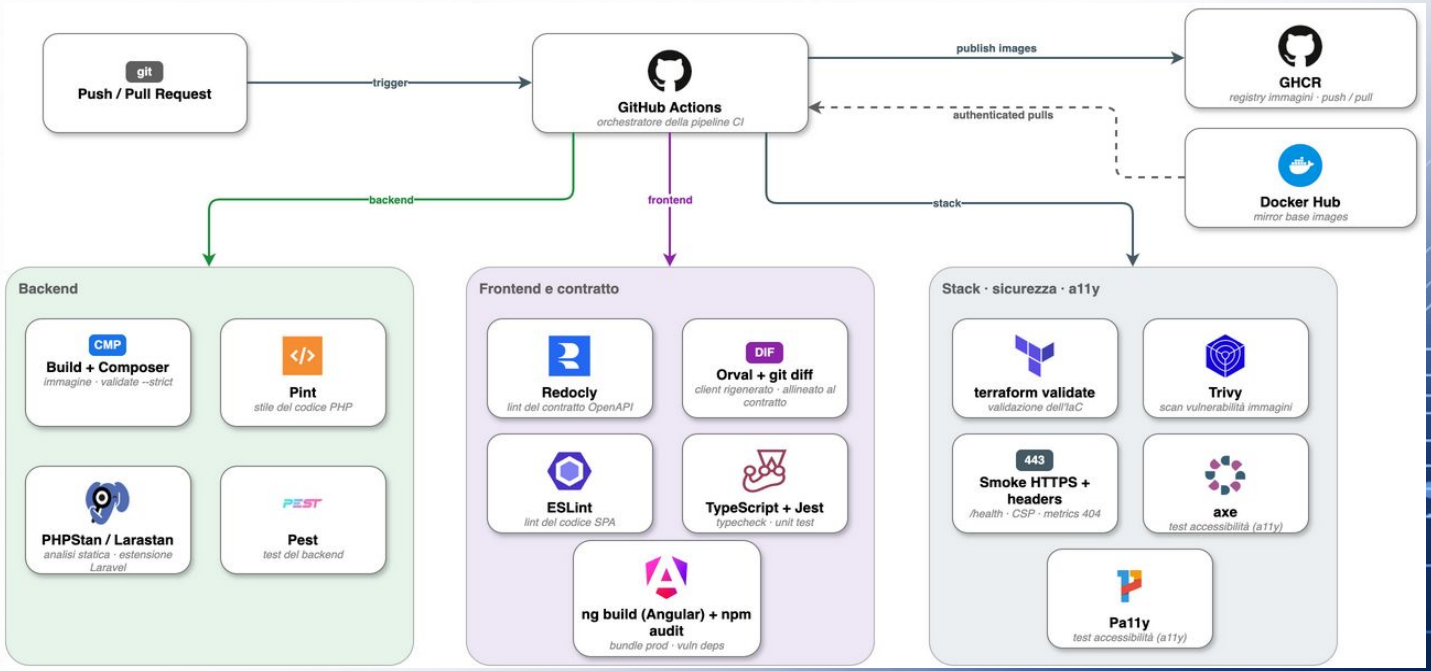
La pipeline gira su **GitHub Actions**, al posto di GitLab CI o Jenkins. Sul frontend ESLint verifica il codice e Jest i test unitari; sul backend **Pint** controlla lo stile, **PHPStan** l'analisi statica e **Pest** i test; la sicurezza passa da **Trivy** e **npm audit**, l'accessibilità da **axe**, **pa11y** e **Playwright**.

Pubblicazione

Le immagini vanno su **GHCR** solo se tutti i controlli passano. **Docker Hub** non è il registro di pubblicazione: lo usiamo soltanto per scaricare immagini autenticate ed evitare i limiti di download.

aLittleByte

Dettaglio: CI e quality gate



aLittleByte