JM3 - Aggiornare

- JM3.1 - La pipeline passo per passo
- JM3.2 - Controllare che sia andato tutto bene
- JM3.3 - Tornare indietro

JM3.1 - La pipeline passo per passo

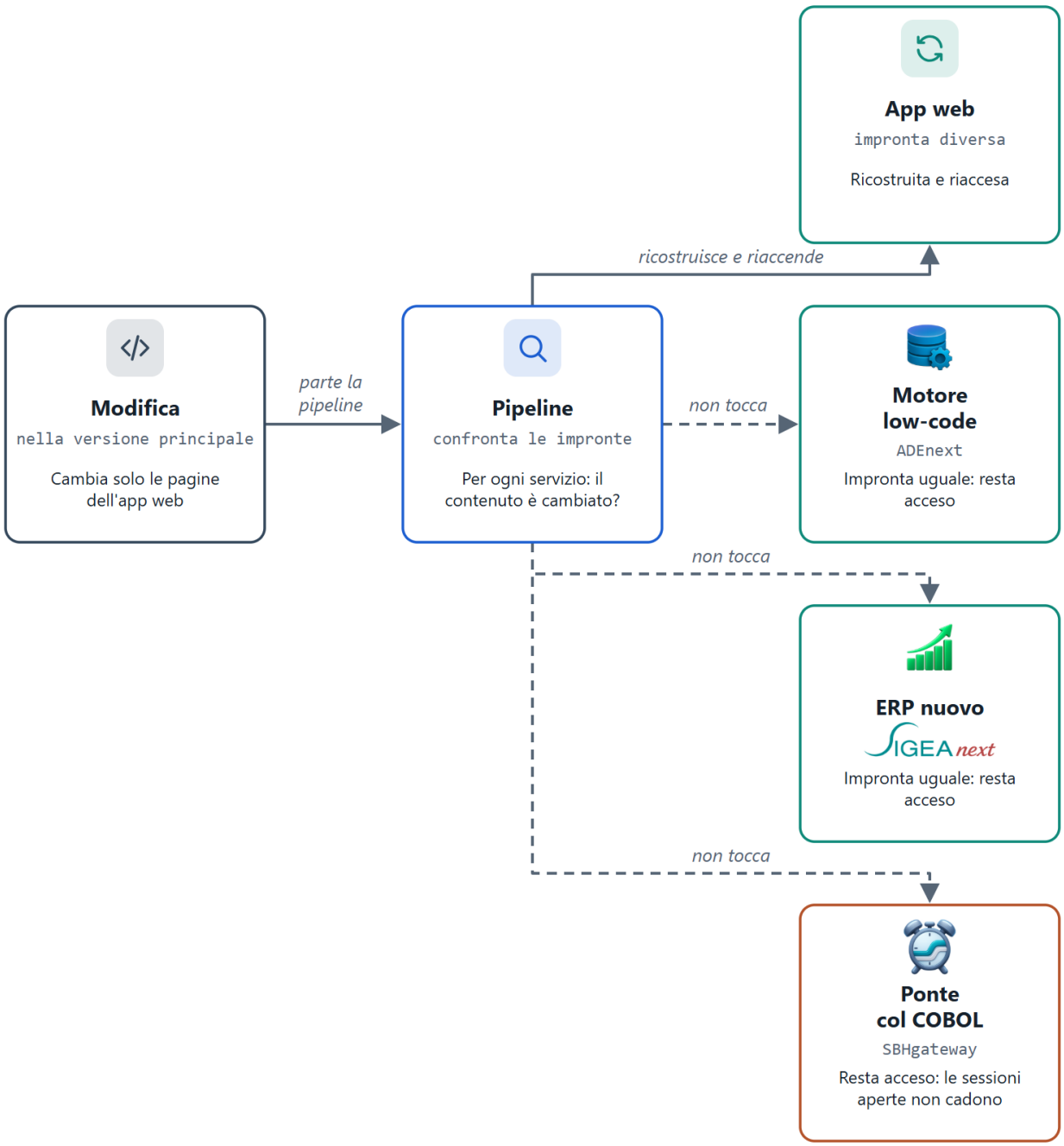
In una frase. Dopo che una modifica entra nella versione principale, la pipeline la prova, ricostruisce solo i servizi cambiati e aspetta il clic di una persona per metterli sul server di prova.

Un aggiornamento fatto a mano dipende da chi lo fa e da quanto è di fretta. La pipeline fa sempre gli stessi passi, e li fa uguali ogni volta. Il passo che cambia davvero ciò che gli utenti vedono resta in mano a una persona: lo decide lei, quando è il momento giusto.

1. **La modifica entra nella versione principale.** Chi sviluppa lavora su un ramo a parte. Quando la modifica è approvata viene unita alla versione principale, e la pipeline parte da sola. Sui rami a parte non parte niente.
2. **Test.** Vengono provate le parti toccate: se cambia ADEnext girano i suoi test, se cambia l'app web girano i controlli delle pagine. Se non cambia nessuna delle due, questo passo non parte.
3. **Build.** Per ogni servizio la pipeline calcola un'**impronta** di ciò che lo compone. Se l'impronta è uguale a quella già presente sul server, il servizio non si ricostruisce. Se è diversa, si prepara la versione nuova. Fin qui sul server non è cambiato niente: gli utenti lavorano ancora con la versione di prima.
4. **Deploy: il pulsante.** La pipeline si ferma e mostra un pulsante. Chi lo preme mette le versioni nuove al posto delle vecchie. Vengono riaccesi solo i servizi con l'impronta cambiata, o con un'impostazione cambiata. Gli altri restano accesi.
5. **I servizi ripartono.** Ogni servizio riacceso, all'avvio, porta il suo archivio alla forma che gli serve e poi si iscrive al Monitor servizi. Ci vuole circa un minuto.

Aggiornare senza ricostruire tutto

Esempio: la modifica tocca solo le pagine dell'app web. Gli altri servizi non si fermano.



Una modifica alle sole pagine: la pipeline ricostruisce l'app web e lascia accesi gli altri servizi

> **Attenzione.** Il clic va dato in un momento tranquillo se l'aggiornamento tocca il Ponte col COBOL: riaccenderlo chiude le sessioni aperte nelle maschere storiche, e il lavoro non confermato si perde. Se il Ponte non è cambiato resta acceso, e nessuno se ne accorge.

Nota. Oggi i test avvisano ma non bloccano: se sono rossi il pulsante compare lo stesso. Prima di premerlo, guarda l'esito dei test. Una schermata della pipeline non è ancora in questa raccolta.

Per chi sviluppa.

• File: `.gitlab-ci.yml`. Stadi: `test`, `helper`, `build`, `deploy`, `reconcile`. Il lavoro di build è `build:images`, quello di deploy è `deploy:simil-prod` (`when: manual`).

• L'impronta è un tag di contenuto (`ADE_NEXT_TAG`, `GPE_NEXT_TAG`, `FRONTEND_TAG`…): l'hash degli alberi git di ciò che entra nell'immagine. Stesso contenuto, stesso tag: niente build, e il deploy non ricrea il contenitore. Dettagli in `devops/deploy/README.md`, paragrafo 3.3.

• I test hanno ancora `allow_failure: true`: il file spiega perché e quando va tolto.

• `reconcile` parte subito dopo l'unione, senza aspettare il clic: dice ad ADEnext quale codice generato è nella versione principale.

• Prima di toccare i servizi, il deploy controlla che nel file delle impostazioni ci siano le password dell'Assistente dei dati: se mancano si ferma e dice quale.

JM3.2 - Controllare che sia andato tutto bene

In una frase. Dopo il clic si aspetta circa un minuto, poi si guarda lo stato dei servizi e si prova una cosa per ogni servizio aggiornato.

Il pulsante verde non basta. Dice che i servizi nuovi sono stati accesi, non che rispondono. Un servizio può partire e fermarsi subito dopo, per esempio perché non trova il suo archivio. Cinque minuti di controllo evitano che se ne accorga un utente il giorno dopo.

1. **Leggi la fine del lavoro di deploy.** L'ultima cosa che scrive è l'elenco dei servizi con il loro stato. Subito dopo l'accensione è normale leggere «sta partendo».
2. **Aspetta circa un minuto.** I servizi riaccesi devono avviarsi e iscriversi al Monitor servizi. Intanto l'app può mostrare «Errore server 503» sulle pagine che li usano. Non è un guasto: passa da solo. I servizi più grossi, come ADEnext e GPEnext, possono metterci un po' di più.
3. **Apri l'app ed entra.** Se la pagina di accesso si apre e ti lascia entrare, l'app web e il Gateway rispondono.
4. **Prova una cosa per ogni servizio aggiornato.**
   - SIGEAnext: apri una tabella con i dati del gestionale.
   - ADEnext: apri una schermata disegnata o un cruscotto.
   - GPEnext: chiedi una stampa in PDF.
   - Ponte col COBOL: apri una maschera storica.
5. **Se una prova non va, guarda chi manca.** Sul server, la pagina del Monitor servizi elenca i servizi iscritti. Chi non c'è non è partito: il motivo è scritto nel suo registro. Vedi il libro JC - Gateway e Monitor servizi, pagina «JC3.1 - Vedere quali servizi sono accesi».
6. **Guarda che il resto non si sia mosso.** I servizi che non erano cambiati devono risultare accesi da prima dell'aggiornamento. Se sono ripartiti anche loro, qualcosa li ha toccati: di solito un'impostazione cambiata.

> **Nota.** Un servizio che si ferma viene riacceso da solo. Per questo un servizio a cui manca un'impostazione non resta spento: continua a ripartire a vuoto, e nell'elenco lo vedi tornare sempre «sta partendo».

> **Attenzione.** L'errore 503 dopo un aggiornamento dura circa un minuto. Se dopo qualche minuto c'è ancora, e sempre sulla stessa funzione, quel servizio non è partito: vedi Se qualcosa non va.

Per chi sviluppa.

• Il lavoro di deploy finisce con `docker compose ps`: stati `starting`, `healthy`, `unhealthy`.

• Ogni servizio ha il suo controllo di stato (`/actuator/health`), che il compose interroga a intervalli. Il tempo concesso per partire va da 40 secondi (Monitor servizi) a 120 (ADEnext, GPEnext): `devops/deploy/docker-compose.deploy.yml`.

• Gateway, SIGEAnext, GPEnext e app web aspettano che il servizio da cui dipendono sia in salute (`depends_on`).

• Le porte interne rispondono solo dal server: il controllo di stato a mano si fa da lì.

• Riavvio automatico: `restart: unless-stopped`.

JM3.3 - Tornare indietro

In una frase. Non c'è un pulsante «annulla»: si torna indietro togliendo la modifica dal codice e facendo ripartire la pipeline, e quello che è già cambiato negli archivi resta.

È meglio saperlo prima. Chi preme il pulsante del deploy deve sapere cosa può disfare e cosa no. Promettere un ritorno automatico che non esiste sarebbe peggio che non averlo.

Quello che si può fare

Non premere. Finché nessuno preme il pulsante, sul server gira la versione di prima. Se i test sono rossi o la modifica non convince, si lascia la pipeline ferma lì. Non succede niente.

Togliere la modifica e rifare il giro. È la strada normale. Chi sviluppa annulla la modifica nel codice, l'annullamento entra nella versione principale e la pipeline riparte. Il servizio torna com'era, e si rimette sul server con lo stesso clic.

Contare sulla versione di prima, per poco. Quando la pipeline prepara una versione nuova, quella vecchia non viene cancellata finché un servizio acceso la sta usando. Dopo il deploy resta sul server ancora per un giro, e sparisce alla pipeline successiva. In quella finestra, se il codice torna identico a prima, l'impronta è la stessa e non serve ricostruire niente. Rimetterla in funzione a mano è un lavoro per chi gestisce il server: non c'è una procedura guidata.

Quello che non si può fare

Riportare indietro gli archivi. Quando un servizio parte, porta il suo archivio alla forma che gli serve: aggiunge una tabella, una colonna. Questi passaggi vanno solo in avanti. Tornare alla versione di prima del servizio non li disfa. Se un passaggio era sbagliato, si corregge con un passaggio nuovo.

Riavere le sessioni chiuse. Se l'aggiornamento ha riacceso il Ponte col COBOL, le sessioni aperte nelle maschere storiche sono state chiuse. Tornare indietro lo riaccende un'altra volta.

> **Attenzione.** Prima di un aggiornamento che cambia la forma di un archivio, serve una copia dell'archivio. La pipeline non la fa.

> **Nota.** Il ritorno alla versione precedente, servizio per servizio e con un comando, è previsto nel piano per l'installazione dai clienti. Quel piano non è ancora realizzato.

>

Per chi sviluppa.

• Pulizia delle immagini: `build:images` toglie i tag diversi da quello attuale con `docker rmi` senza `-f`, che non tocca le immagini in uso. Vedi `devops/deploy/README.md`, paragrafo 3.3.

• Il tag è l'hash del contenuto: un revert che riporta la cartella del servizio allo stato di prima dà lo stesso tag di prima.

• Le migrazioni di ADEnext e GPEnext partono a ogni avvio. In ADEnext una migrazione già applicata e poi modificata non viene rieseguita: si aggiunge una migrazione nuova.

• Gli archivi sono descritti nel libro JL - Archivi. Copie dei volumi su file: `devops/deploy/README.md`, paragrafo 3.4.