# Base comune Moonitoring

## Architettura, responsabilità e procedura di rilascio sullo stage

Guida completa per Jacopo e Lorenzo · 10 novembre 2026

Include introduzione e capitoli 01-12. Distingue configurazione applicata, prove concluse, codice verificato localmente e proposte operative ancora da attivare.

Versione condivisa: guida nella documentazione aziendale. I collegamenti rispettano gli accessi già previsti; non puntano al computer dell’autore.

## Risultato e perimetro

Access e Farm hanno un percorso di rilascio verificato sullo stage. L’approvazione protetta e i test delle migrazioni sono descritti con il loro effettivo livello di verifica. Soglie, ricezione degli allarmi e restore hanno criteri concreti; gli esiti futuri non vengono anticipati.

# Indice completo

Base comune Moonitoring 3

01 - Architettura e responsabilità 5

02 - Dallo sviluppo al rilascio 7

03 - Modello comune per un componente 9

04 - Accessi: chi può fare cosa 11

05 - Stato e chiusura della base iniziale 13

06 - Test per accettare la base comune 16

07 - Fonti e limiti delle evidenze 19

08 - Schema SQL e rilascio dei componenti 21

09 - Collaudo del kit e del rilascio Access - 9 novembre 2026 23

10 - Recovery isolato: retry reale verificato 25

11 - Secondo sistema: onboarding Farm 27

12 - Soglie e prova di ripristino dello stage 29

# Base comune Moonitoring

Guida condivisa per Jacopo e Lorenzo · aggiornata al 10 novembre 2026.

Scarica il PDF completo, con introduzione e capitoli 01-12.

Questa introduzione e le dodici schede sono la sorgente completa versionata della guida. Le sintesi dei collaudi sono condivisibili; le ricevute tecniche complete restano nell'archivio operatore, senza pubblicare state o credenziali.

**La base cloud esiste: rilascio Access e recupero SQL-free sono collaudati. Farm ha completato rollout, verifiche SQL e prova privata Platform→Farm con dinieghi e cleanup.** Il controllo dell'ordine viene esplicitato nella CLI di approvazione; restano i controlli operativi descritti nel piano. Console e il dettaglio di tutti i futuri sistemi non sono prerequisiti.

## Come funziona

**Google Cloud** ospita risorse e applicazioni. **Terraform** descrive e configura infrastruttura e permessi. **GitHub** conserva il codice. **Cloud Build** lo controlla e costruisce le immagini. **Cloud Deploy** distribuisce gli artefatti approvati sullo stage, eseguendo le migrazioni previste.

Nelle prove di rete, **Platform→Farm** indica il percorso dal progetto `moonitoring-platform-stage` al progetto `moonitoring-farm-stage`. Non esiste un servizio chiamato Platform da distribuire: i nomi e i livelli sono distinti.

L'autonomia riguarda le **release ordinarie** dei componenti già registrati. Jacopo e Lorenzo coordinano il primo onboarding e le nuove risorse/autorizzazioni. Lo sviluppatore mantiene Dockerfile, script, test, manifest e migrazioni; le ricette Cloud Build e i controlli privilegiati restano comuni.

Per un nuovo componente si dichiarano una volta destinazione, identità, risorse e accessi. I successivi aggiornamenti del codice riusano la pipeline. Terraform cambia quando servono nuove risorse o autorizzazioni, non per ogni versione applicativa.

## Cosa definiamo adesso

| Adesso, come base comune | Durante lo sviluppo del servizio |
|---|---|
| Confini dei sistemi già noti e risorse condivise | Componenti effettivi, API, schema e migrazioni |
| Modello breve per dichiarare un componente | Compilazione del modello con i suoi bisogni reali |
| Profili riusabili di esecuzione, accesso e rilascio | Risorse e permessi specifici prima del primo deploy |
| Procedura e test comuni | Test del comportamento applicativo |

Non occorre completare oggi le schede Farm/Releases o distribuire tutti i servizi. La configurazione autorevole rimane nei cataloghi Terraform e nei manifest; la scheda spiega le decisioni, senza duplicare tutto il codice.

## Punto attuale e ultimi passi

Sono già predisposti progetti, rete, Cloud SQL, database e identità iniziali. Access ha provato backend, migrazione e telemetria. I moduli comuni esistono, ma alcuni profili hanno soltanto test locali.

**Aggiornamento Farm del 10 novembre:** le modifiche applicative e infrastrutturali sono su `main`; il publisher comune ha prodotto le due immagini. La baseline del database è stata applicata conservando la vecchia fixture e verificando il catalogo Access invariato; l'amministratore SQL temporaneo è stato rimosso. Login privato e isolamento SQL sono verificati prima e dopo le migrazioni; il primo rollout è riuscito. La finestra IAM è chiusa con grant sui workload nominati. I cinque casi S2S e il cleanup sono riusciti; metriche e una traccia correlata sono ricevute. Il migratore rieseguito non ha trovato migrazioni mancanti, con i limiti di evidenza indicati nella scheda. Stato e prossima sequenza.

**Aggiornamento operativo del 9 novembre:** il kit Access ha completato rilascio ordinario, migrazione prima del backend, ripresa senza duplicazioni e verifica di richieste/telemetria. È riuscito anche il retry reale di una fixture senza SQL, con lo stesso candidato approvato; sonde e cleanup sono conclusi. Il nuovo comando di approvazione verifica ordine e conflitti prima del POST; i suoi test sono distinti dai rilasci storici. Lorenzo verifica le proprie azioni al normale primo uso: una sessione dedicata non è un prerequisito. Rilasci Access · Recovery isolato.

La **base comune versione 1** è a questo punto:

1. **Standard allineato:** mantenere coerenti modello, profili supportati ed esempi verificabili.
2. **Kit e procedura pronti:** rilascio ordinario e retry SQL-free sono verificati, con cleanup concluso. Il nuovo comando esplicito di approvazione è implementato e testato; il kit aggiornato è verificato offline. Il primo uso reale del nuovo comando sarà registrato al prossimo rilascio, senza ripetere appositamente i deploy già riusciti. Ogni operatore verifica le proprie azioni nel normale utilizzo.
3. **Secondo sistema verificato:** Farm ha superato rollout, accessi SQL, S2S e cleanup; il kit successivo è pronto. Le nuove API riusano questo profilo, verificando i propri grant e il primo rilascio. Non serve completare il prodotto Farm. Secondo sistema.
4. **Da completare per la chiusura operativa:** attivare e provare gli allarmi, verificare limiti e ricezione notifiche, eseguire il restore isolato della scheda 12. Il recupero v1 è definito come fix/revert compatibile su main, preservando le migrazioni applicate; retry e ripristino del database restano operazioni diverse.

La consegna iniziale rende autonomo il rilascio dei profili API accettati. Per SQL comprende il percorso **backend + migratore separato**: il backend modifica i dati, il migratore modifica la struttura. Un nuovo sistema con questa combinazione può riusare il percorso, dopo la propria registrazione e verifica.

**Prima di attivare un secondo componente che scrive nello stesso schema**, per esempio un consumer di Access, serve invece estendere e collaudare la delivery. Il limite riguarda la nostra procedura, non la capacità di PostgreSQL di gestire più scrittori. Le regole per schema e componenti SQL spiegano il lavoro aggiuntivo.

Readonly, eventi e Job restano attività da completare per profilo, nella sezione separata delle estensioni; non bloccano i quattro passi della consegna iniziale. I primi deploy controllati servono proprio a collaudare nuove capacità. Console/staff web e produzione restano consegne distinte. “Versione 1” stabilizza un contratto estendibile, non certifica ogni requisito futuro.

## Percorso di lettura

| Documento | Risponde a |
|---|---|
| 01 - Architettura | Dove sono le risorse e chi le gestisce? |
| 02 - Sviluppo e rilascio | Come arriva un commit sullo stage? |
| 03 - Modello del servizio | Che cosa compiliamo e quando? |
| 04 - Accessi | Chi può fare cosa? |
| 05 - Stato e prossimi passi | Che cosa è pronto e quale lavoro manca? |
| 06 - Collaudi | Come dimostriamo che la base è pronta? |
| 07 - Fonti | Quali prove e documenti sostengono queste informazioni? |
| 08 - Schema e componenti SQL | Chi modifica le tabelle e come rilasciamo più componenti sullo stesso schema? |
| 09 - Collaudo del 9 novembre | Che cosa ha funzionato davvero con il nuovo kit e quali prove restano aperte? |
| 10 - Recovery isolato | Come abbiamo verificato il retry di un deploy fallito e quali casi non copre? |
| 11 - Onboarding Farm | Quale componente riusiamo e cosa serve prima di metterlo online? |
| 12 - Soglie e ripristino | Quali allarmi proponiamo e come proveremo il restore senza toccare lo stage corrente? |

Questa guida distingue implementazione, applicazione e collaudo. Le prove Access restano separate dal recovery e da Farm. La doc ufficiale registra il recovery nella PR #93 e la preparazione Farm nella PR #94, entrambe unite. Il registro ufficiale conserva anche l'esito Farm; questa guida collega quei risultati alla procedura comune e alle decisioni successive, senza trasformare un aggiornamento documentale in un nuovo deploy. Le fonti distinguono commit fissati e copie locali.

# 01 - Architettura e responsabilità

Guida iniziale · Stato

## I nomi che non vanno confusi

| Nome | Significato |
|---|---|
| **Moonitoring Platform** | L'ecosistema intero: sistemi, interfacce e regole comuni. Non è una singola repository o un progetto Google Cloud. |
| **Area dei sistemi comuni / Platform** | Raggruppamento tecnico usato per spiegare Access, distribuzione client e altri servizi trasversali. Ogni sistema conserva responsabilità, dati e API proprie. |
| `moonitoring-platform-stage` | Progetto Google Cloud concreto che ospita alcuni runtime comuni. Il nome non gli assegna tutte le risorse della piattaforma: rete, DB e CI sono in progetti dedicati. |
| **Repository applicativa** | Sorgenti di un sistema o di un'interfaccia. Può produrre più componenti, come backend e migratore. Non coincide automaticamente con un progetto cloud. |
| **Modulo Terraform** | Codice riusabile che configura un gruppo di risorse. Un progetto può usare diversi moduli; lo stesso modulo serve più sistemi. |

Quindi aggiungere un servizio ad Access o Farm non richiede automaticamente un nuovo progetto, database o modulo Terraform. La scelta segue isolamento, risorse e responsabilità; la registrazione esplicita conserva il confine fra sistemi.

## Dove sono le risorse

La parte predisposta è sotto l'organizzazione aziendale `moonitoring.com`. Le cartelle raggruppano ambienti e responsabilità; i progetti delimitano risorse, IAM e attribuzione dei costi. Uno stesso account di fatturazione può finanziare più progetti.

```
moonitoring.com
├─ 10-shared
│  ├─ moonitoring-infra-seed       state Terraform e identità delle fondamenta
│  └─ moonitoring-cicd             sorgenti GitHub, Build, Deploy e artefatti CI
└─ 30-stage
   ├─ moonitoring-net-stage       rete condivisa e collegamento privato SQL
   ├─ moonitoring-db-stage        una istanza PostgreSQL: sql-stage
   ├─ moonitoring-platform-stage  runtime dei sistemi comuni
   └─ moonitoring-farm-stage      runtime e risorse del prodotto Farm
```

È uno schema logico semplificato: omette cartelle intermedie e altri progetti aziendali. Conserviamo gli ID esistenti. Sviluppo locale con Docker; la produzione non viene attivata da questo piano.

`sql-stage` contiene i database `access`, `releases`, `news`, `ai_gateway`, `farm` e `moochat`; `platform` storico è conservato. I database separano dati e privilegi, ma condividono la capacità della stessa istanza.

Il progetto `moonitoring-platform-stage` ospita Access e ospiterà gli altri sistemi comuni. `moonitoring-farm-stage` ha anche bucket raw/dataset e code predisposte. Access e il backend+migratore Farm hanno una delivery collaudata; Farm ha provato anche il percorso privato fra progetti. Database e identità degli altri sistemi non ne provano l'esecuzione. Nei collaudi, **Platform→Farm** abbrevia soltanto questi due progetti Google Cloud.

Il secondo sistema collaudato è Farm, con backend e migratore reali. Build, DNS privato, migrazione, backend, separazione SQL e S2S sono verificati; metriche e una traccia applicativa sono ricevute. La riesecuzione del migratore ha confermato il no-op. Stato Farm.

## Che cosa gestisce Terraform

Terraform descrive progetti, rete, database, archivi, code, identità, IAM e registrazioni delle pipeline. `plan` mostra le modifiche previste; `apply` le esegue. Bootstrap e stage hanno state remoti separati nel bucket delle fondamenta: lo state associa la configurazione alle risorse reali ed è protetto da accessi limitati e locking, che impedisce scritture concorrenti sullo stesso state. Non è l'archivio dei dati applicativi né il kit da dare a ogni sviluppatore.

Le release usano tre output selezionati: `delivery`, `runtime` e `operator`. Descrivono destinazioni e limiti della procedura. Vengono esportati e versionati, senza condividere l'intero state.

## Che cosa gestisce il servizio

Una repository può contenere backend, migratore e altri componenti. Ogni componente attivato deve avere identità, risorse e modalità di rilascio esplicite. Un Service Cloud Run riceve richieste; un Job esegue un lavoro finito. Un worker può essere uno dei due secondo il suo funzionamento, non è necessariamente una macchina sempre accesa.

Il sistema è responsabile del proprio modello dati. Schema e migrazioni sono versionati nel repository applicativo, ma vengono applicati dal **Job migratore con un'identità distinta**. Backend e worker leggono o aggiornano i dati autorizzati, senza privilegi per modificare la struttura e senza eseguire migrazioni all'avvio. La separazione riguarda esecuzione e permessi: non richiede una repository distinta o modifiche manuali al database. Schema e rilasci SQL.

Il manifest descrive come eseguire il container: immagine, identità, rete, variabili e limiti. Cloud Deploy gestisce le revisioni applicative; Terraform prepara risorse e permessi. I due strumenti non devono modificare contemporaneamente la stessa specifica del workload.

Common e Design sono dipendenze di codice. Console è l'interfaccia interna dello staff, senza database proprio: usa le API dei sistemi. Access gestisce identità e autorizzazioni applicative; i servizi si chiamano direttamente con identità e policy previste, senza passare tutti attraverso Console o Access.

Per il percorso GitHub → cloud vedere 02; per le diverse autorizzazioni 04. Riferimenti verificati in 07.

# 02 - Dallo sviluppo al rilascio

Guida iniziale · Architettura

**L'autonomia riguarda i rilasci ordinari dei componenti già registrati.** Jacopo e Lorenzo possono sviluppare servizi diversi e riusare la stessa procedura. Il primo onboarding e le variazioni di risorse o permessi restano un lavoro coordinato fra loro; non diventano configurazioni cloud nascoste nel servizio.

## Chi prepara cosa

| Responsabile | Cosa mantiene |
|---|---|
| Sviluppatore del servizio | Codice, Dockerfile, script di build/test, dipendenze fissate, manifest, hook e migrazioni. Prova il servizio localmente e dichiara risorse/accessi necessari. |
| Jacopo e Lorenzo, per l'onboarding | Scelta del profilo già supportato, cataloghi Terraform, identità e grant minimi, baseline SQL, prima creazione, contratti applicati e kit consegnato. |
| Infrastruttura comune | Ricette Cloud Build, verifica degli artefatti, strumenti di release/approvazione/recupero e moduli Terraform riutilizzabili. |
| Operatore e approvatore abilitati | Eseguono la procedura con il proprio account, verificano il candidato, approvano esplicitamente e conservano le ricevute. I gruppi possono avere oggi le stesse persone; i ruoli restano distinti. |

GitHub conserva le modifiche revisionate. Cloud Build esegue i controlli e pubblica immagini identificate dal digest; Artifact Registry le conserva. Cloud Deploy rende e distribuisce i manifest dopo l'approvazione, con la migrazione separata quando prevista. Un `cloudbuild.yaml` applicativo non può sostituire il percorso privilegiato comune o assegnarsi nuovi permessi.

Terraform prepara infrastruttura e accessi; Cloud Deploy gestisce le revisioni applicative. Non devono gestire contemporaneamente la stessa specifica Service/Job. I root bootstrap e stage hanno state distinti, non distribuiti agli sviluppatori.

## Prima registrazione e release ordinarie

**Una volta per il componente:** concordare repository/profilo, destinazione, identità, risorse e capacità. Applicare i piani necessari, preparare SQL, adattare i manifest ai contratti e completare il primo rilascio. Dopo la prima creazione si chiude la finestra IAM e si consegna il kit per i workload esistenti. Onboarding.

**A ogni release ordinaria:** codice e test nella repository del servizio, modifica revisionata su `main`, poi lo stesso kit e la stessa pipeline. Nessun apply Terraform se non cambiano risorse, autorizzazioni o profilo.

1. Selezionare lo SHA revisionato di `main` **remoto corrente**. La CLI lo ricontrolla prima di pubblicare e caricare il candidato; se `main` avanza, predisporre un run coerente senza riscrivere gli input congelati.
2. Avviare il publisher e approvare esplicitamente la build Cloud Build.
3. Preparare candidato e bundle immutabili; approvare separatamente la build che verifica gli artefatti, crea la release e rende i manifest.
4. Ispezionare il rollout e approvarlo con il comando dedicato sotto.
5. Verificare rollout, Service Ready, migrazione, accessi e segnali previsti.

Access e Farm usano **migratore separato → successo → backend**. Il Job applica solo le versioni mancanti; un errore impedisce il nuovo backend, senza spegnere automaticamente quello già attivo. Un Job generico invece non viene eseguito implicitamente a ogni release: registrazione, avvio e scheduling sono distinti.

## Kit e approvazione manuale

Il kit contiene i tre contratti **delivery, runtime, operator** e gli strumenti corrispondenti. Si verifica da un checkout fidato con l'hash ricevuto separatamente; run, ricevute, credenziali e ambiente Python restano fuori dal kit. Preparazione e verifica.

`stage_release.py init/advance/status` prepara e legge la release; non approva implicitamente build o rollout. Gli eventi automatici sono spenti nel percorso configurato. Per l'approvazione Cloud Deploy, l'approvatore usa esplicitamente:

```
python -B tools/stage_approval.py inspect --run-dir <run> --account <email-aziendale> --output <snapshot.json>
python -B tools/stage_approval.py approve --run-dir <run> --account <email-aziendale> --snapshot <snapshot.json> --coordina
ted-with-team
```

L'approvatore usa la propria identità Google, distinta dall'operatore che ha preparato il run; non si riscrive l'account nel ledger. IAM decide se può approvare. `inspect` e `approve` devono usare lo stesso account. La CLI rifiuta configurazioni SDK con impersonazione o sostituzione delle credenziali: l'approvatore resta identificato con il proprio account aziendale. Il tool verifica candidato e render, stato `PENDING_APPROVAL/NEEDS_APPROVAL`, snapshot recente e inventario dello stesso target nella stessa pipeline. Rifiuta rollout attivi o incerti e candidati più recenti non scartati; **più recente significa rollout Cloud Deploy**, non un commit Git o un bundle solo preparato. Rilegge lo stato prima della richiesta e registra l'intento per riconciliare una risposta persa. Cloud Build conserva il proprio gate esterno.

Sono controlli preventivi, non un lock distribuito o una transazione tra operatori. I permessi IAM consentono ancora altri canali, compresa la console: il tool non impedisce quel bypass. Il team mantiene un solo rollout in esecuzione per componente/ambiente e concorda quale candidato usare. `--coordinated-with-team` attesta tale accordo; non dimostra che l'altro operatore sia tecnicamente bloccato.

Un aggiornamento del kit serve per usare nuovi strumenti o contratti. Non sostituire i JSON di un run già iniziato né attribuire ai vecchi kit comandi che non contengono.

## Ripresa, retry e ritorno al comportamento precedente

- **Ripresa:** conservare il run e usare `status/advance`; un avvio ambiguo si riconcilia prima di `adopt`. Non cancellare il ledger per chiedere una seconda build.
- **Retry dello stesso rollout:** `stage_recovery.py inspect/retry` mantiene gli stessi artefatti e ricontrolla target, job fallito e assenza di conflitti. Richiede il riscontro dell'Operation/Execution precedente e l'accordo del team. Il tempo trascorso o una lista vuota non provano che il lavoro sia fermo.
- **Ripristino applicativo nella v1:** una nuova fix o revert del codice su `main`, compatibile con lo schema corrente e **senza eliminare le migrazioni già applicate**, passa dalla normale pipeline. Non si seleziona uno SHA vecchio e non si usa il rollback Cloud Deploy a una vecchia release. Nessuna down-migration o restore dell'istanza condivisa è implicito nel rilascio.

SeaORM rifiuta una versione applicata nel database che il migratore non conosce. È una protezione già presente; i test di regressione la rendono esplicita. Perciò un revert dell'intera repository che cancella vecchie migrazioni non è una strategia valida. Schema e compatibilità.

Se il primo deploy fallisce prima di creare il workload, Jacopo e Lorenzo riconciliano risorse e prima-creazione IAM. Un `404/403` non autorizza un retry cieco. Se esiste solo il Job migratore, gli servono i grant nominali prima del retry. Un nuovo kit vale per i run successivi; la storia del run originale resta immutata.

Le prove esistenti coprono release Access/Farm, retry isolato SQL-free, SQL e S2S. Il nuovo guard di approvazione ha test propri e mantiene i limiti di concorrenza sopra indicati; non richiede di rompere nuovamente i servizi online. Runbook · Collaudi.

# 03 - Modello comune per un componente

Indice · Sviluppo e rilascio

**Ora fissiamo lo standard; dettagliamo ogni componente mentre lo sviluppiamo e prima del suo primo deploy.** Non occorre completare Farm, Releases, Console o tutte le schede future per dividere il lavoro.

Lo sviluppatore propone risorse e accessi, Dockerfile/script/test e manifest. Jacopo e Lorenzo completano insieme la registrazione cloud e consegnano il kit; poi le release ordinarie non richiedono di ripetere l'onboarding.

Un sistema può contenere backend, migratore e worker: la scheda descrive un componente eseguibile e le sue dipendenze. Una libreria, invece, non riceve automaticamente un servizio cloud.

## Quando definire le informazioni

| Momento | Informazioni necessarie |
|---|---|
| **Ora: standard condiviso** | Distinzione sistema/componente, profili ammessi, separazione runtime/migratore, accessi espliciti e criteri comuni di collaudo. I valori sconosciuti restano «da definire». |
| **Durante lo sviluppo** | Repository/directory, referente, Service o Job, dati utilizzati, chiamanti/destinatari, migrazioni, capacità cloud e limiti necessari. Verificare presto eventuali profili non supportati. |
| **Prima del primo deploy** | Nomi e identità effettivi, risorse/permessi applicati, baseline SQL se necessaria, manifest senza segnaposto, contratti coerenti, immagini verificate e piano di collaudo/recupero. Nessuna ambiguità sui privilegi. |

## Scheda da copiare

Questo è un **promemoria Markdown descrittivo**, non un nuovo formato eseguibile o un file letto da Terraform. Riportare riferimenti ai cataloghi: non duplicare manualmente tutti i loro valori.

```
Sistema / componente / referente:
Repository e directory:
Profilo build / delivery:
Destinazione e identità: riferimento al catalogo Terraform
Dati e capacità: DB/ruolo, bucket, segreti per riferimento, eventi
Chiamanti e destinatari: rete, identità, operazioni consentite
Schema richiesto, compatibilità e referente delle migrazioni (se SQL):
Altre dipendenze:
Limiti, prove attese e recupero:
Stato: da definire / predisposto / collaudato, con evidenza
```

Non inserire password, chiavi o token. IAM e autorizzazioni applicative rimangono distinti: accessi.

## Dove diventano configurazione

- Catalogo stage: `system_catalog` e risorse/capacità dei componenti.
- Configurazione bootstrap: `application_repositories` per sorgenti/build; `application_deliveries` per le nuove delivery, secondo il modulo comune.
- Configurazione stage: `runtime_deliveries`, enrollment/baseline SQL ed eventuali eventi richiesti.

I contratti delivery/runtime/operator sono **output generati** da confrontare e consegnare all'operatore, non nuovi JSON da scrivere a mano. Un nuovo componente richiede registrazioni revisionate; le successive release ordinarie riusano quelle risorse.

## Profili ed esempi

| Profilo delivery | Limite attuale |
|---|---|
| `service-with-migration` | Un backend SQL più un migratore separato: provato con Access e Farm; riusabile per altri sistemi con questa combinazione. |
| `service` | Service senza SQL; provato con la fixture recovery. Ogni nuovo componente verifica i propri grant e comportamento. |
| `service-read-only` | Service SQL readonly con baseline dedicata; prove cloud aggiuntive mancanti. |
| `job` | Job finito senza SQL; esecuzione e scheduling richiedono prove distinte. |

**Componente SQL aggiuntivo sullo stesso schema:** il profilo runtime con scrittura, senza migratore proprio, non è ancora supportato dalla delivery. Il validatore lo blocca; non basta aggiungerlo al catalogo. Prima di attivarlo servono un prerequisito di schema verificato, ordine e recupero dei rilasci, mantenendo un'unica responsabilità per le migrazioni del sistema. È un'estensione futura esplicita, non un limite di PostgreSQL né un requisito per ogni nuovo backend. Dettaglio e criteri.

I profili build sono `rust-system`, `rust-library`, `dependency` e `rust-slint-app`: non coprono automaticamente qualsiasi linguaggio.

**Access, esempio reale:** `access-backend`, Platform stage, DB `access`, runtime `sa-accounts`; `access-migrate` usa `sa-access-migrate`. Build/delivery, kit e rilasci sono collaudati; nuovi controlli della CLI hanno test separati e non cambiano retroattivamente quelle ricevute.

**Job, esempio illustrativo non registrato:** un controllo finito legge un bucket autorizzato, senza SQL. Vanno definiti identità, oggetti leggibili, durata/retry ed esito; aggiungere Scheduler soltanto se serve. Non assegnargli ora nomi cloud o permessi inventati.

Seguono stato e prossimi passi, collaudi e fonti.

# 04 - Accessi: chi può fare cosa

Indice · Architettura · Sviluppo e rilascio

**Il catalogo assegna a ogni componente gli accessi dichiarati; i grant personali, ereditati o legacy vanno verificati separatamente.** Condividere progetto, rete o sistema non autorizza tutte le comunicazioni. Gli operatori amministrano il cloud; gli utenti delle applicazioni usano ruoli e dati gestiti dal prodotto.

| Tipo | Chi e dove | Controllo e stato essenziale |
|---|---|---|
| **Operatori cloud** | Gruppi aziendali → risorse Google | Terraform assegna ruoli ai gruppi; i gestori aggiungono persone. Grant applicati. Ogni operatore verifica le azioni assegnate nel normale primo uso, distinguendo gruppi e privilegi personali: nessuna sessione dedicata di Lorenzo blocca il lavoro. |
| **Build e deploy** | Identità CI distinte → sorgenti, immagini e workload | Checker, publisher, preparazione, render e deploy hanno compiti separati. Rilascio Access e retry isolato SQL-free sono verificati entro il perimetro delle ricevute. Il publisher non approva il proprio rollout. |
| **Database** | Runtime/migratore → database del sistema | Servono rete privata, login IAM e privilegi PostgreSQL. Runtime senza DDL o ruolo owner; migratore separato. Access è verificato. Farm ha superato le verifiche delle due identità prima e dopo la migrazione, con runtime senza DDL/ledger e altri sei database negati a entrambe. Readonly esplicito. |
| **Servizio → servizio** | Service account chiamante → API destinataria | Token Google, audience corretta, invoker nominale e rete compatibile; il destinatario controlla operazioni e dati. Platform→Farm verificato su health/readiness: autorizzato 200, senza grant/token 403, audience errata 401, dopo revoca 403; sonda e grant rimossi. Il grant Console→Access non ne sostituisce il collaudo. |
| **Archivi, segreti, firma** | Componente → bucket, secret o chiave dichiarati | Lettura, creazione, cancellazione e firma sono capacità diverse. Access ha grant specifici; ogni nuovo componente richiede quelli necessari, senza ereditare tutto dal sistema. |
| **Eventi e Job** | Publisher/SA del trigger → topic, Service o Job | Pub/Sub usa push autenticato; Scheduler usa OIDC per HTTP e OAuth per avviare Job. Moduli predisposti, eventi generici inattivi; verificare anche esito dell'Execution, duplicati e retry. |
| **Staff nelle app** | Persona → ingresso IAP → applicazione | IAP ammette il gruppo; sessione e capacità applicative restano separate. Ingresso staff non attivo. Console è un'integrazione, non un prerequisito della piattaforma. |
| **Client e agent** | Utente/installazione → API prodotto | Token Access e autorizzazioni per organizzazione. Dal 10 novembre la doc prevede IAP anche sui percorsi client dello stage; in produzione quei percorsi non richiedono account Google dei clienti. Edge e sessioni restano da implementare/collaudare. |

`infra-admins` amministra shared e ambienti attivati. `stage-release-admins` approva rilasci; `stage-observers-admins` legge metadata, log e metriche, senza ricevere accesso **diretto** ai dati SQL o ai valori Secret Manager. I log e le configurazioni leggibili possono però contenere informazioni scritte dalle applicazioni: i ruoli IAM non le oscurano. Occorre mantenere i log sanitizzati e i valori segreti fuori dalle configurazioni visibili.

I gruppi produzione restano riservati; non è previsto un gruppo amministratori stage separato. Le iscrizioni non sincronizzano i ruoli applicativi. I permessi per ripetere una fase fallita o tornare a una release precedente vanno verificati

e collaudati separatamente: l'approvazione del rilascio non ne dimostra la disponibilità. Vedere la procedura di rilascio e recupero.

Il piano di recupero, **applicato**, aggiunge agli operatori lettura di release/rollout/JobRun e `retryJob` sulle pipeline dichiarate, più letture sui Service/Job già esistenti e dichiarati. Non aggiunge approvazione, creazione di release, modifica SQL, rollback o lettura delle Operations dell'intero progetto. Letture e retry reale sono provati con Jacopo; questo non certifica retroattivamente Lorenzo né la sufficienza dei soli gruppi rispetto ad altri privilegi personali. Il suo primo uso ordinario fornirà la verifica delle azioni effettive, senza richiedere una sessione dedicata. Recovery e limiti.

La separazione runtime/migratore è una regola già prevista, da mantenere per ogni nuovo componente. Un consumer aggiuntivo può ricevere scrittura sui dati necessari, **non la capacità di modificare tabelle o assumere il ruolo del migratore**. Gli accessi SQL da soli non ne abilitano la delivery: serve anche il percorso descritto nella scheda SQL.

**Limiti reali:** l'approvazione Cloud Build e alcuni accessi alle sorgenti hanno ambito progetto CI; il writer delle immagini ha ambito repository Artifact Registry, non singola immagine. Non promettiamo isolamento maggiore.

Per l'ingress ristretto attuale abbiamo applicato DNS privato `run.app` sulla Shared VPC con Private Google Access; il riscontro DNS è positivo. Il percorso completo è stato provato con un **Job sonda** in Platform verso Farm, senza aspettare Console. L'endpoint `run.app` rimane abilitato: la risoluzione DNS cambia il percorso di rete, non l'URL chiamato o l'audience del token. L'accesso rimane soggetto a ingress e IAM.

Il collaudo iniziale ha usato una sonda temporanea in Platform e il backend Farm reale, con `private-ranges-only` e Shared VPC coerente sia sul chiamante sia sulle revisioni destinatarie. Non richiede il prodotto Farm completo. Sono verificati il successo con l'identità autorizzata e i dinieghi prima del grant, senza token, con audience errata e dopo revoca; cinque Job e grant temporaneo sono rimossi. Una prova soltanto dentro Platform o la sola lettura DNS non certificano questo percorso. Collaudo e limiti.

È una scelta di rete, non un requisito universale di IAM. La sonda prova il percorso e i permessi cloud; le autorizzazioni sulle singole operazioni e sui dati richiedono anche test del servizio.

Durante lo sviluppo basta dichiarare identità, destinazioni e operazioni nel modello del servizio. Terraform prepara grant concreti quando servono; il codice applica il proprio RBAC. Prossimi passi · Collaudi · Fonti e limiti delle evidenze.

## Apertura Access: profilo specifico, ancora separato

La prova Farm ha chiuso la comunicazione privata S2S. Non apre i percorsi esterni di Access e non certifica Console o client. La policy documentale del 10 novembre mette IAP davanti allo staff e, **solo nello stage**, anche ai percorsi dei client. Dopo IAP restano i token e i permessi del prodotto. In produzione i percorsi client non usano IAP; i clienti non diventano account aziendali Google.

Prima di aprire Access serve il suo profilo di ingresso: backend/percorsi esposti, identità ammessa, audience e sessione coerenti, nessun aggiramento dell'ingress interno o confusione con RPC fra servizi. Il modulo `staff_edge` attuale non prova che l'intero edge client-stage sia già pronto. L'onboarding di un altro servizio privato può procedere senza aspettare questa apertura.

# 05 - Stato e chiusura della base iniziale

Guida iniziale · Test di accettazione

## Cosa abbiamo fatto

| Parte | Risultato sostenuto dalle evidenze |
|---|---|
| Fondamenta | Progetti, rete, SQL, archivi, code di base e identità predisposti; state bootstrap/stage separati |
| Access | Backend e migratore distribuiti, IAM di prima creazione ristretto, tracce e metriche ricevute |
| Farm, aggiornamento 10 novembre | Rollout, SQL, cinque casi S2S e cleanup riusciti; finestra IAM chiusa. Metriche e una traccia ricevute, no-op migratore verificato con i limiti documentati. Dettaglio |
| Pipeline comune | Rilascio ordinario Access e retry reale SQL-free verificati; stesso candidato/artefatti dopo il guasto, vecchio errore conservato e richiesta duplicata rifiutata. Cleanup completato, piani finali senza differenze |
| Accessi team | Gruppi e grant configurati e riletti; ogni operatore verifica le azioni assegnate nel normale primo uso. Nessuna sessione dedicata Lorenzo richiesta per procedere |
| Profili ulteriori | Profili di accesso SQL, eventi, Job ed edge predisposti in parte nel codice; mancano collaudi cloud. La delivery di ulteriori scrittori dello stesso schema richiede anche un'estensione. |

Access e il backend minimo Farm sono online nello stage privato; il secondo sistema ha superato il collaudo di delivery e accessi previsto. È registrata anche la fixture temporanea `access-recovery` con pipeline separata, usata per collaudare il recupero e poi rimossa: non è un nuovo prodotto. Gli altri sistemi completano l'enrollment durante lo sviluppo. Console ha preparazione e identità, ma non è online. Evidenze e limiti.

L'allineamento infra è unito nelle PR #26 e #27; la doc è allineata con la PR #90 e i risultati pubblicati con la PR #91, commit `8519f8e43160`. Sono passati **82 test Python, 63 test Terraform bootstrap e 93 test stage**. Sono state applicate con successo **2 aggiunte bootstrap e 4 stage, tutte IAM**, senza aggiornamenti o cancellazioni. I piani successivi a `d5904b` non mostrano differenze. Il kit reale è verificato e le letture Service/Job/Execution di Jacopo funzionano senza impersonazione. Fonti e ricevute. Nessuna prova effettiva dell'account di Lorenzo o chiusura dell'intera base è ancora dichiarata.

Access PR #30 è unita a `99f7c780`: 14 verifiche offline, 98 test Rust e Compose riusciti; ambiente locale di prova ripulito. Il publisher cloud `31b73c94-d3d6-49d7-913d-8c15ba992019` e la preparazione `dfc9b103-8920-4a1e-a107-cd78bcfbf3e6` sono riusciti. Riaprire il comando ha ritrovato il publisher originale senza crearne un altro.

Il nuovo rollout `access-stage-0001` della release `access-0b88f714f6c8e3f668343ac04ccdeedb7f83f582a3b9f04d` è **riuscito**: migratore `access-migrate-k57km` concluso alle **17:47:36 UTC**, prima del deploy del backend alle **17:47:39 UTC**. Nessuna migrazione mancante, grant SQL stage rieseguiti; revisione `access-backend-mw9jv7nr` Ready, immagine/identità verificate e 100% del traffico. La nuova sonda ha completato 360 richieste HTTP 200; tracce e metriche sono ricevute, Job/identità/grant temporanei rimossi. Collaudo completo e limiti.

Un secondo rilascio dello stesso commit è riuscito con revisione `access-backend-mw91e435`. Il tentativo controllato di interrompere il predeploy si è fermato prima di inviare cancellazioni: è quindi un'altra release ordinaria, non un retry positivo. La seconda revisione ha completato richieste, telemetria e cleanup; la sonda ha richiesto una ripetizione dopo un primo 403 di causa non accertata. La scheda 09 separa i due run: il retry Cloud Deploy non è stato eseguito in quei run; la successiva prova positiva è nella scheda 10.

## Consegna iniziale e completamento dei profili

**La consegna iniziale rende autonomo il rilascio delle API:** standard verificato, procedura Access, onboarding del backend e del migratore Farm già disponibili, accessi SQL/S2S pertinenti e operatività minima. Farm verifica un secondo sistema reale; una sonda temporanea in Platform ne prova separatamente l'accesso fra progetti. Non occorre completare il prodotto Farm o Console.

Per SQL chiudiamo il percorso **un backend + un migratore separato per sistema**, già provato con Access. Il runtime non modifica la struttura; il migratore applica le modifiche versionate e la nuova revisione del backend viene distribuita solo dopo il successo del predeploy. Un altro sistema con questo profilo richiede la propria registrazione e verifica, non l'estensione per scrittori aggiuntivi.

**Prima di attivare un secondo componente scrivente sullo stesso schema**, come un consumer di Access, dobbiamo aggiungere il profilo runtime SQL senza migratore indipendente, verificare lo schema richiesto e definire ordine/recupero dei rilasci. Poi va collaudato. È un limite del contratto di delivery attuale, non del database; non impone un nuovo orchestratore o il rilascio simultaneo di tutti i componenti. Decisioni e lavoro residuo SQL.

**Il completamento progressivo dei profili resta nel piano:** readonly, Pub/Sub push, Scheduler HTTP, Job senza SQL e componenti SQL aggiuntivi. Non blocca l'uso ordinario dei profili già accettati; ciascuna estensione va chiusa prima dell'uso ordinario che la richiede.

Web staff, client finali e produzione sono consegne separate. Nuove combinazioni, come Job applicativi con SQL o altre toolchain, richiedono un'estensione esplicita; non sono automaticamente supportate.

## Stato dei quattro passi e lavoro residuo

| Passaggio | Cosa fare concretamente | Finito quando |
|---|---|---|
| **1. Standard comune** | Allineamento eseguito fra modello, documentazione e strumenti: mantenerlo quando si estende un profilo. | Campi, validazioni e limiti sono espliciti; non servono schede complete di tutti i sistemi. |
| **2. Rilascio autonomo** | Flusso ordinario, telemetria, ripresa e retry SQL-free provati. Comando di approvazione implementato, testato e incluso nel kit aggiornato, verificato offline. Registrare il suo primo uso reale al prossimo rilascio. | Prove del flusso e del recupero registrate; il guard rifiuta i conflitti coperti dai test. Azioni individuali verificate al normale primo uso, senza promettere un lock distribuito. |
| **3. Nuovo componente e accessi** | **Completato nel perimetro Farm:** rollout, SQL, cinque casi S2S, cleanup e kit successivo. Metriche/traccia e no-op verificati con i limiti riportati. Stato concreto. | Primo rilascio Farm e migrazione riusciti, poi chiamata tra progetti provata con identità/audience non autorizzate negate. La prova di rete rimane distinta dalla riuscita del deploy. |
| **4. Consegna operativa** | **Residuo concreto:** implementare le soglie proposte, provare ricezione allarmi e restore isolato. Fix/revert compatibile su main è definito; i test verificano il rifiuto di migrazioni applicate sconosciute, senza alterare il ledger. | Notifiche ricevute e restore verificato con tempi, punto recuperato e cleanup. Le soglie e gli obiettivi della scheda 12 diventano risultati misurati. |

La consegna iniziale chiude **tutti e quattro i passi**. Access e Farm già online consentono di usare il profilo accettato mentre si completano i controlli operativi residui. **Il completamento dei prodotti Farm, Releases e Console non è una dipendenza.** Sonde e grant temporanei sono rimossi nei collaudi registrati; il backend Farm è un servizio reale e resta distinto dalle fixture da eliminare.

Possiamo proseguire senza attendere una sessione dedicata di Lorenzo. Le sue azioni effettive si verificano nel normale primo uso, senza sostituire questa evidenza con le prove effettuate usando Jacopo. Il collaudo Platform→Farm ha una propria ricevuta positiva; il recovery Access rimane una prova distinta.

## Estensioni da completare quando richieste

Readonly, Pub/Sub push, Scheduler HTTP e Job senza SQL hanno un'accettazione separata per profilo. Per ulteriori scrittori dello stesso schema occorre prima implementare la delivery runtime SQL, il prerequisito di schema e il recupero per componente, poi collaudarli. Job applicativi con SQL e nuove toolchain richiedono un'estensione esplicita.

Queste attività non sono un passaggio intermedio obbligatorio della consegna iniziale: diventano necessarie prima dell'uso ordinario di un servizio che le richiede. Console/staff web, accessi dei clienti e produzione mantengono integrazioni e collaudi propri.

## Dettagli operativi da chiudere

- **Kit operatore:** il pacchetto Access è già preparato con `operator_kit.py` da infra `d5904b` e verificato nella copia operatore. Ogni destinatario deve verificarlo da checkout fidato con l'hash ricevuto separatamente. Ogni run conserva input e ricevute immutabili fuori dal kit; non si condivide lo state. La verifica del pacchetto non prova i privilegi del destinatario.
- **Ripresa e retry:** `advance` non ritenta il rollout. `stage_recovery.py inspect/retry` usa fotografia del target e riscontro attribuito della precedente Operation/Execution o del mancato avvio provato. Se l'esito è ancora attivo o incerto, non ritenta; un'incertezza non risolvibile con gli accessi disponibili richiede un amministratore autorizzato. Si ritenta l'esatto job/fase Cloud Deploy con artefatti invariati, senza ignorare la migrazione fallita. La sintassi e il formato

`--reconciliation` sono nel runbook infra. Letture, dinieghi del retry non ammesso e retry positivo dopo un errore di permesso sulla fixture SQL-free sono provati; non è una prova del recupero SQL. Ricevuta dedicata.

- **Ripristino applicativo v1:** fix o revert del codice su main compatibile con lo schema corrente, conservando le migrazioni applicate; normale pipeline e approvazioni. Non è previsto rollback Cloud Deploy verso una vecchia release né restore SQL implicito. Il guard SeaORM contro versioni applicate sconosciute viene provato con regressioni dedicate, senza introdurre guasti nel cloud condiviso. Dettaglio in 02 e 08.
- **Rilasci dello stesso componente:** la CLI dedicata ispeziona il target e ricontrolla un candidato prima dell'approvazione; il retry mantiene i propri controlli. I test rifiutano snapshot scaduti/cambiati, rollout attivi o incerti e candidati Cloud Deploy superati. Non sono un lock distribuito né impediscono approvazioni dalla console: il team concorda ancora un solo rollout in esecuzione per componente/ambiente. I sistemi indipendenti possono procedere separatamente.
- **Prima creazione:** se l'oggetto esiste anche senza essere Ready, applicare grant nominali, chiudere la finestra e aggiornare `exists`. Questo cambia il contratto: esportare un nuovo kit per le successive esecuzioni CLI, lasciando intatti input e registro del vecchio run. Il retry del rollout esistente è una procedura separata, non un aggiornamento dei suoi JSON.

## Cosa chiudiamo e cosa resta successivo

La versione iniziale consegna una procedura usabile e un perimetro verificato. Quando si sviluppa un servizio, si completa la sua configurazione e si prova la prima release; un nuovo bisogno può richiedere un'estensione. Un profilo può entrare in collaudo controllato prima di essere dichiarato disponibile.

Jacopo e Lorenzo concordano onboarding, risorse e accessi; ciascuno mantiene codice, Dockerfile, script, test e manifest dei servizi che sviluppa. Le release ordinarie riusano kit e ricette comuni. Entrambi validano le interfacce condivise. Console/staff web, accessi dei clienti, prodotto dati completo e produzione hanno integrazioni proprie ancora da completare. Non vengono dichiarati pronti dalla consegna iniziale API; gli ulteriori profili mantengono la propria accettazione.

Le soglie iniziali, RPO/RTO proposti e la prova isolata sono nella scheda 12. Non sono nuovi allarmi applicati o un restore già eseguito.

# 06 - Test per accettare la base comune

Guida iniziale · Sequenza di lavoro

**I test verificano il meccanismo comune, poi l'integrazione del servizio.** Usiamo Access, il backend+migratore Farm già sviluppati e fixture isolate per i guasti. Validazioni, digest e casi unitari si provano localmente; IAM, rete, Cloud SQL e trigger richiedono verifiche cloud rappresentative. Un apply riuscito non dimostra da solo il funzionamento.

Sono ammessi primi deploy controllati con fixture isolate per accettare nuovi profili. Prima del collaudo il profilo non è disponibile per l'uso ordinario: questo non impedisce la prova necessaria a renderlo disponibile.

## Checklist della consegna iniziale

| Area | Test e risultato richiesto |
|---|---|
| **Modello** | Esempi validi accettati; identità, risorse o capability incoerenti rifiutate. Distinguere profilo disponibile da esempio solo documentale. |
| **Rilascio Access** | Nuova build sullo SHA ammesso → preparazione → approvazione Deploy → migrazione → backend Ready. Ritrovare un rollout già riuscito non vale come nuovo test. |
| **Operatori** | Una prova completa del flusso comune; ogni persona verifica le proprie azioni nel normale primo uso. Nessuna sessione dedicata di Lorenzo è un prerequisito. Grant personali ampi vanno distinti dalla sufficienza dei soli gruppi. |
| **Ripresa e recupero** | Interrompere la CLI senza duplicare build; riconciliare un avvio ambiguo. Prima del retry leggere Operation/Execution precedente: se attiva o incerta, non lanciarne un'altra. Provare il recupero su fixture, senza ignorare migrazioni fallite. |
| **Ordine dei rilasci** | Test automatici del comando di approvazione: stesso target/pipeline, candidato corretto, snapshot recente e invariato, rifiuto di rollout attivi/incerti o più nuovi non scartati. Ispezione prima del normale uso; accordo del team resta necessario, perché controllo e POST non sono atomici e la console resta un altro canale. Nessuna necessità di rompere nuovamente lo stage. |
| **Prima creazione** | Workload nuovo o creato ma non Ready: grant nominali applicati, finestra IAM chiusa, contratti aggiornati e recupero tracciato. |
| **SQL iniziale** | Runtime limitato ai dati autorizzati: niente DDL, ownership o assunzione del ruolo migratore. Migratore separato: applica le mancanti, no-op se già presenti, errore blocca il nuovo backend; compatibilità con la revisione precedente. Provare inoltre il rifiuto di una migrazione applicata sconosciuta al codice, senza modificare il ledger. Riusare le prove valide; integrare quelle mancanti. |
| **S2S e risorse** | Dopo il primo rilascio dello skeleton Farm, un Job sonda in Platform prova il percorso Shared VPC verso il backend: chiamata autorizzata riuscita, identità/audience errate negate. Configurare e verificare separatamente l'autorizzazione dell'operazione Catalog se scelta. Le prove SQL precedono il rilascio; un errore DNS non prova un diniego IAM. |
| **Operatività** | Traccia/metrica nuova, allarme ricevuto, limiti SQL/istanze e spesa controllati; un restore rappresentativo su copia isolata. Usare il percorso v1 di 02: fix/revert compatibile su main, migrazioni preservate e normali gate; test di compatibilità e del migratore. Il retry non vale come prova di rollback. Non ripetere il restore per ogni nuova API. |

Il collaudo S2S usa `private-ranges-only`, DNS privato `run.app` già applicato e Private Google Access, con le revisioni destinatarie collegate alla Shared VPC. La prova precedente `ALL_TRAFFIC` o una chiamata solo dentro Platform non coprono questa configurazione. Conservare URL `run.app` e audience coerenti; registrare risoluzione DNS, identità ed esiti positivi/negativi. Il destinatario è il backend Farm reale; soltanto la sonda Platform e il suo grant sono temporanei. Non occorrono Console o le ulteriori funzionalità Farm. Procedura della sonda · Requisiti Google per la Shared VPC.

## Estensioni da accettare quando richieste

- **Eventi e Job:** push autenticato, duplicati, retry, ack durevole e dead-letter; Job con Execution ID, completamento/fallimento e doppia invocazione. L'avvio accettato da Scheduler non prova il risultato. L'uso di un Job sonda non certifica la schedulazione applicativa.
- **Readonly:** il runner SQL prova i privilegi senza Dev ORM, ma non la delivery `service-read-only`; per accettare quest'ultima serve anche un Service di prova.

Queste prove non bloccano i profili API già accettati. I workload e i grant temporanei delle fixture vengono rimossi; le definizioni e gli artefatti protetti possono restare inattivi per audit, senza indebolire le protezioni contro la cancellazione.

## Prima di attivare ulteriori componenti sullo stesso schema

Queste prove accettano la futura estensione della delivery SQL; non bloccano la consegna del percorso backend + migratore:

- Due identità runtime possono scrivere i dati concessi; entrambe restano senza DDL o accesso allo schema di un altro sistema.
- Il nuovo componente resta bloccato se manca lo schema compatibile richiesto; dopo il successo del migratore può essere distribuito senza una propria migrazione indipendente.
- Estendere le prove di ordinamento già previste nella base iniziale alla delivery di più componenti: due tentativi di migrazione non applicano la stessa modifica in concorrenza. Il lock delle migrazioni non sostituisce transazioni e idempotenza delle scritture applicative.
- Una patch compatibile aggiorna un solo componente. Durante un cambiamento di schema le versioni ancora attive restano funzionanti.
- Un errore di migrazione blocca i dipendenti; un errore successivo del consumer si recupera per componente, senza presumere un rollback globale o distruttivo dello schema.

Usare fixture isolate e registrare quale profilo è stato accettato. La correttezza delle operazioni concorrenti sui dati richiede anche i test applicativi del sistema.

## Prove già disponibili e limiti

**Farm, 10 novembre:** publisher riuscito, DNS privato applicato e riletto, baseline SQL applicata e riletta; metadati della fixture precedente e catalogo Access invariati, credenziali/risorse amministrative temporanee rimosse. Rollout e migrazioni riusciti; identità SQL verificate prima e dopo, con sei database estranei negati a entrambe e runtime senza DDL/ledger. Finestra IAM chiusa. Cinque casi S2S e cleanup riusciti; metriche e una traccia correlata ricevute. La seconda traccia richiesta non è stata osservata. No-op del migratore verificato con l'identità operatore: non equivale a un nuovo rollout Cloud Deploy dopo la chiusura IAM. Le righe della vecchia fixture non sono state lette o certificate. Stato e ricevute Farm.

La prova del 7 novembre comprendeva rollout, migrazione no-op e 360 richieste private health/ready, con tracce e metriche. Il nuovo collaudo del 9 novembre, distinto da quella prova, è riassunto sotto. I casi migrazioni mancanti/errore hanno prove locali dedicate. Questo non prova login staff, tutti gli endpoint o carico di cento allevamenti.

L'estensione kit/recupero di infra `d5904b` ha superato **82 test Python**, più **63 test bootstrap e 93 stage**. Le sei aggiunte IAM sono applicate, i piani successivi non mostrano differenze e il kit reale Access è verificato. Il nuovo flusso cloud ha completato pubblicazione, preparazione e rollout con migrazione prima del backend. Un nuovo processo operatore ha ritrovato lo stesso publisher senza crearne un altro. La nuova sonda ha completato **360 richieste HTTP 200**; sono ricevute nuove tracce, metriche applicative e metriche SQL proxy. Risorse e grant temporanei sono rimossi. Risultati e ricevute del 9 novembre.

Il recupero ha letto gli stati reali e rifiutato retry non ammessi. La successiva fixture SQL-free ha provato il retry reale dello stesso job/candidato dopo un errore di permesso: nuovo JobRun riuscito, errore originale conservato, richiesta duplicata rifiutata e 10 risposte HTTP 200 prima, durante e dopo il guasto. Pulizia completata e piani finali senza differenze. Non prova recupero di migrazioni SQL, routing Platform→Farm o concorrenza fra operatori. La riconciliazione resta assistita; Lorenzo verifica le proprie azioni al normale primo uso, senza un collaudo duplicato come prerequisito.

I test dei guasti restano isolati; non inseriamo deliberatamente una migrazione errata nel servizio condiviso. Il recupero applicativo non annulla lo schema: serve compatibilità. Le code hanno retention finita anche a push sospeso; il test deve prevedere replay e cleanup, senza perdere backlog reale.

## Evidenze e chiusura

Per ciascuna prova salvare commit, hash dei contratti, identità, build/release/rollout o Execution ID, digest, orari, risultato e log sanitizzati. Registrare costi/risorse temporanee e relativa chiusura.

La prova comune verifica il profilo; ogni nuovo componente verifica i propri grant, configurazione e prima release. RPC, isolamento tra aziende e correttezza dei dati sono test applicativi. Si riusano le evidenze compatibili: la batteria comune si ripete quando cambia il meccanismo interessato, non per ogni servizio.

**La consegna iniziale si chiude con i punti 1-4 della scheda 05.** Le estensioni sono accettate separatamente, quando richieste, e restano esplicitamente aperte. Lo sviluppo può procedere prima; i rilasci ordinari usano profili già accettati. Web staff/clienti avranno test propri prima dell'apertura.

Le soglie iniziali, RPO/RTO proposti e la prova isolata sono nella scheda 12. Non sono nuovi allarmi applicati o un restore già eseguito.

# 07 - Fonti e limiti delle evidenze

Guida iniziale

La guida distingue **contratto architetturale, codice, risorse applicate e prove eseguite**. Una modifica alla doc o un test locale non è un nuovo deploy. Le ricevute operative complete restano nell'archivio aziendale degli operatori; questa versione condivisibile riporta sintesi sanitizzate, commit e risultati. Non contiene Terraform state, piani grezzi, token o dati degli allevamenti.

## Fonti normative e codice

| Tema | Fonte |
|---|---|
| Sistemi, domini e componenti | System Overview |
| Progetti, cartelle e ambienti | Resource Hierarchy |
| IAM e identità | Identity & Access |
| Ingress e IAP, aggiornamento Lorenzo del 10 novembre | Network |
| Cataloghi e procedura comune | Infra delivery · Catalogo |
| Kit e comandi | Kit · Runbook |
| Schema e profili ammessi | Validatore · SQL Farm |

Le fonti fissate a commit conservano quella versione; i collegamenti `main` seguono il codice corrente e richiedono accesso aziendale alle repository. I cambiamenti recenti di Lorenzo su IAP stage, standard, Status e client legacy sono disegno/sviluppo: non sono servizi aggiunti automaticamente al nostro stage.

## Sintesi delle ricevute

| Prova | Risultato e limite | Resoconto condivisibile |
|---|---|---|
| Access, 7-9 novembre | Release, migrazione prima del backend, richieste, tracce/metriche, ripresa senza duplicare publisher. Non prova tutti gli endpoint o la capacità per cento aziende. | Scheda 09 |
| Recovery isolato, 9 novembre | Retry dello stesso job/candidato dopo un diniego di deploy; errore originale conservato, duplicato rifiutato, cleanup. Non è rollback SQL o ordinamento distribuito. | Scheda 10 |
| Farm, 10 novembre | Secondo sistema reale: build, SQL prima/dopo, rollout, S2S in cinque casi, cleanup e piani finali senza differenze. Metriche e una traccia precisa ricevute. | Scheda 11 |
| Nuovi controlli della procedura | Approvazione esplicita con snapshot/inventario; regressioni sul database più avanti del codice. Test del codice, senza nuovi rilasci applicativi. | Procedura · Schema |
| Allarmi e restore | Soglie e obiettivi proposti, non allarmi già attivati o restore già riusciti. | Scheda 12 |

I riferimenti completi di release/build/Execution nei resoconti permettono agli operatori autorizzati di ritrovare il run. Le sintesi non sostituiscono le ricevute originali e non sono certificazioni di prestazioni o conformità.

## Regole di lettura

Aggiornamento degli strumenti del 10 novembre: il commit Infra `eb2cd10` ha superato 98 test operatore, di cui 15 sul comando di approvazione. La PR Infra #35 è unita su `main` (`de07d918`), senza modifiche Terraform applicate. Il kit Farm aggiornato è verificato offline, con gli stessi contratti applicati; hash del manifest: `e5dc2d5ee5c115c27e58722c861cb9e584dc309126e60f85dc867dc7ab8ff318`. Il 10 novembre alle 09:40 UTC `inspect` ha verificato il rollout Farm reale già `SUCCEEDED`, restituendo il blocco «non in attesa di approvazione», senza modifiche cloud. Il primo POST di approvazione con il nuovo comando resta da registrare al prossimo rilascio.

Le regressioni PostgreSQL 17 sul database più avanti del codice sono passate: tre casi in Access (`17ad6646`) e quattro in Farm (`c544d043`). Abbiamo riusato i test aggiunti da Lorenzo nella PR Farm #15, chiudendo la nostra #16 duplicata. Farm passa fmt, Clippy e 57 test del workspace; un doctest preesistente è ignorato. Passano anche i 14 test degli hook. Compose locale è riuscito: sei immagini compilate, nove servizi avviati, backend health/ready positivo, migratore e setup Pub/Sub conclusi, otto versioni nel ledger SQL. Container e risorse di prova sono rimossi; il motore isolato è fermo. Sono prove del codice: non aggiornano il backend o lo schema online.

- Un gruppo configurato non prova tutte le azioni di ogni persona; Lorenzo verifica le proprie azioni nel normale primo uso, senza un rilascio duplicato.
- Un grant IAM non prova rete, SQL o autorizzazione applicativa. S2S Farm prova health/readiness e IAM, non il Catalog o i dati di organizzazioni reali.
- Un Service Ready non prova telemetria; i segnali sono verificati separatamente. Per Farm manca una delle due tracce precise cercate: la richiesta `/ready` risulta comunque da HTTP 200, log e metriche.
- Farm online usa `fc5d06e` con due migrazioni. Il codice collaudato `c544d043` ne contiene otto: queste nuove migrazioni non sono state distribuite dalle prove qui registrate.
- Gli allarmi e il restore restano obiettivi della scheda12 finché non esistono le rispettive prove. Non sono SLA di produzione.

La doc ufficiale conserva anche il registro dei collaudi. Per l'implementazione delle regole recenti fare riferimento ai commit delle PR che le introducono; non attribuire quei comandi ai kit storici.

# 08 - Schema SQL e rilascio dei componenti

Indice · Piano · Collaudi

**Il runtime modifica i dati; il migratore modifica la struttura.** La base iniziale mantiene questa separazione. Il supporto per ulteriori componenti scriventi dello stesso sistema richiede un'estensione della nostra delivery, non un nuovo database né più privilegi al backend.

Qui parliamo di componenti che dipendono dalle stesse tabelle o dallo stesso schema applicativo. Condividere soltanto l'istanza Cloud SQL, come Access e Farm, non crea questa dipendenza.

## Chi è responsabile di cosa

| Parte | Responsabilità |
|---|---|
| Infrastruttura | Istanza/database, rete, identità, baseline dei ruoli e permessi. Terraform e la procedura SQL preparano l'accesso. |
| Sistema applicativo, per esempio Access | Modello dati, schema e migrazioni versionate, compatibilità delle proprie versioni. Possono restare nella stessa repository del codice. |
| Job migratore | Esegue le migrazioni approvate con identità distinta, registra quelle applicate e usa il lock previsto per evitare esecuzioni concorrenti incompatibili. |
| Backend, consumer e altri runtime | Leggono e modificano soltanto i dati autorizzati. Non creano/modificano tabelle, non assumono il ruolo del migratore e non eseguono migrazioni all'avvio. |

Per ogni schema del sistema serve **una responsabilità e una sequenza coerente di migrazioni**. Non un migratore globale per Access, Farm e tutti gli altri database, né migratori concorrenti con evoluzioni indipendenti dello stesso schema. La separazione dei privilegi non richiede una repository diversa o interventi manuali dalla console. In PostgreSQL l'ownership consente modifiche strutturali: non va assegnata ai runtime. Privilegi PostgreSQL.

## In quale senso il database guida il rilascio

**Lo schema disponibile vincola quali versioni possiamo attivare.** Modello dati e software vengono progettati insieme; le esigenze del prodotto possono richiedere nuove strutture. Il backend non decide autonomamente di trasformare il database quando parte.

Esempio: aggiungere una colonna opzionale, senza cambiare il comportamento atteso dal vecchio codice.

1. Una migrazione versionata prepara la colonna e i permessi necessari, restando compatibile con i componenti in esecuzione.
2. Dopo il successo si rilasciano i componenti che la usano, anche in momenti diversi.
3. Eventuali rimozioni di vecchie strutture arrivano soltanto quando nessun componente attivo o previsto per il recupero ne dipende più; si verificano anche i dati da conservare.

Quindi “prima tutto il database, poi tutto il codice” non è una regola universale. Le aggiunte compatibili possono precedere il codice; le rimozioni incompatibili lo seguono. Una patch che usa lo schema esistente non richiede una modifica SQL. Le transazioni e l'idempotenza delle normali scritture rimangono responsabilità applicative: il lock delle migrazioni non le risolve.

## Cosa serve ora per chiudere la base comune

Il percorso collaudato Access/Farm è **migratore separato → successo → nuovo backend**. Il Job applica solo le migrazioni mancanti; se non ce ne sono termina senza applicarne. Un errore ferma il nuovo backend, non implica lo spegnimento di quello già in esecuzione. Cloud Deploy fornisce il passaggio predeploy, mentre la nostra procedura avvia e verifica il Job.

Per la consegna iniziale completiamo procedura operatore, test dei privilegi e recupero di questo percorso, insieme agli altri punti del piano. Un nuovo sistema con un backend e un migratore può riusarlo dopo la propria registrazione e verifica. Non occorre prima abilitare tutti i consumer o worker futuri.

Già nella versione iniziale si possono preparare due release dello stesso backend: l'ordine di approvazione e recupero va quindi provato ora, come specificato in 02 e 06. Access verifica immagine/identità del Job, usa l'`etag` all'avvio e un lock SQL durante la migrazione; queste protezioni non certificano l'ordine dell'intero rollout. Nella v1 si torna al comportamento precedente con una fix/revert su main, conservando le migrazioni applicate e verificando lo schema corrente; non si riusa

una vecchia release Cloud Deploy. Non si presume una cancellazione delle modifiche SQL già riuscite.

## Database più avanti del codice

SeaORM possiede già il controllo delle versioni applicate: se il ledger contiene una migrazione che il codice del migratore non conosce, l'esecuzione si ferma. I test espliciti verificano che il Job fallisca e il nuovo backend non parta, senza riscrivere la storia del database. Non è una nuova funzione da affidare al backend o un confronto con il solo numero massimo di versione.

Una fix/revert valida conserva tutte le migrazioni già applicate e modifica solo il comportamento compatibile. Non rimuovere file o registrazioni per far passare il controllo. I test nuovi riguardano il codice della repository, non un deploy: Farm online resta al commit `fc5d06e` con le due migrazioni del collaudo; i successivi commit di sviluppo contengono altre migrazioni e non sono stati distribuiti da queste prove. Versioni a cui si riferiscono i risultati.

## Cosa aggiungere prima di un secondo runtime scrivente

PostgreSQL supporta più scrittori. Il validatore attuale ammette però SQL runtime soltanto nella coppia backend/migratore; gli altri profili sono senza SQL o readonly. Il seguente lavoro è **proposto e ancora da implementare**:

1. Un profilo di delivery per il componente SQL aggiuntivo, con identità propria e scrittura limitata, senza DDL né migratore indipendente.
2. Un prerequisito verificabile di schema: migrazioni richieste, compatibilità dichiarata e riscontro sul database di destinazione. Il solo numero di versione più alto non dimostra compatibilità.
3. Ordine ed esiti tracciati: riusare il migratore del sistema, attendere le modifiche necessarie prima dei componenti dipendenti e registrare immagini/esito per ciascun rilascio. Non serve subito un orchestratore generale.
4. Collaudo di permessi, prerequisiti, fallimenti e recupero, descritto in 06.

I rilasci non sono atomici tra componenti e non richiedono sempre lo stesso commit. Una patch compatibile può aggiornare solo il consumer. Se la migrazione fallisce, i rilasci dipendenti restano bloccati; se fallisce un consumer dopo una migrazione riuscita, si recupera quel componente senza presumere di annullare lo schema. L'estensione deve essere accettata prima dell'uso ordinario del componente aggiuntivo, non prima di ogni futuro servizio.

# 09 - Collaudo del kit e del rilascio Access - 9 novembre 2026

Guida iniziale · Lavoro residuo · Fonti

**Questa scheda conserva i due rilasci ordinari di Access e le relative prove.** Alla loro conclusione il retry positivo non era ancora collaudato; il test successivo riuscito su una fixture separata è nella scheda 10. L'intera base comune non è ancora chiusa.

## Cosa è stato verificato

| Parte | Risultato e riscontro |
|---|---|
| Infrastruttura | **2 aggiunte bootstrap + 4 stage, tutte IAM**, senza aggiornamenti o cancellazioni. Piani successivi bootstrap e stage senza differenze. |
| Kit | Pacchetto dagli output applicati, copia operatore verificata e ancora immutata a fine run. Strumenti e contratti consegnati senza state o credenziali. Riepilogo · Istruzioni. |
| Accessi dell'operatore | Jacopo ha letto Service, Job ed Execution senza impersonazione. Prova. Non certifica il profilo di Lorenzo o la sufficienza dei soli gruppi rispetto agli altri privilegi personali. |
| Pipeline | Publisher e preparazione riusciti dopo i gate; nuovo rollout `SUCCEEDED`. Un nuovo processo CLI ha ritrovato la stessa build publisher senza crearne un'altra. Registro del run. |
| Migrazione e backend | `access-migrate-k57km` concluso alle **17:47:36.428290 UTC**, prima del deploy backend alle **17:47:39.931236143 UTC**. Nessuna migrazione mancante, grant SQL stage rieseguiti. Revisione `access-backend-mw9jv7nr` Ready, immagine/identità attese e 100% traffico. Riepilogo. |
| Richieste | **360/360 HTTP 200**: 180 `/ready`, 180 `/health`, primo tentativo riuscito. Prova. Non sono test di login, operazioni gestionali o capacità per cento allevamenti. |
| Telemetria | Nuovi span e metriche applicative ricevuti in Platform; metriche del proxy SQL ricevute in DB. Tracce · Metriche. |
| Pulizia | Job sonda, service account e grant temporaneo rimossi dopo il successo dell'Execution. Riepilogo. La verifica non ha attivato produzione o Console. |

Prima del cloud sono passati 82 test Python infra, 63 test bootstrap, 93 test stage, 14 verifiche offline Access e 98 test Rust. Compose ha costruito sei immagini: otto componenti in esecuzione, tre Job conclusi con codice 0, `/ready` e `/health` del backend riusciti e 20 migrazioni nel ledger locale. L'ambiente locale di prova è stato ripulito. Sono prove locali distinte dal rilascio cloud.

## Tracce: che cosa dimostra la lettura

Sono stati riletti **40 record di span** della stessa istanza `c8bbcd0b-c4a0-4be3-a238-391da4c250bd`; le metriche applicative sono state filtrate per quella stessa istanza. Fra le tracce c'è l'ID esatto inviato dalla sonda per `/ready`: `6bcb02bfeb682b0c1457f3e30a870e11`.

Non sosteniamo di aver esportato 360 tracce. Il primo `/health` della sonda era marcato **non campionato** nei log Cloud Run, mentre il primo `/ready` era campionato: riscontro degli header. Nessun warning dell'exporter è stato rilevato nella finestra diagnostica. Le metriche possono includere anche i controlli automatici del servizio; non sono un conteggio esclusivo delle richieste della sonda.

## Recupero: implementato e provato, con quali limiti

`stage_recovery.py inspect` ha letto il rollout in attesa, in corso e riuscito. Il comando ha rifiutato retry prima dell'approvazione e dopo il successo, senza creare intento o richiesta di retry cloud.

**Questi due run non hanno provato il retry positivo.** La successiva fixture isolata lo ha verificato per il caso SQL-free. Resta obbligatorio riconciliare l'Operation/Execution precedente: una lista vuota o un timeout non provano che il lavoro non sia partito. Il primo deploy fallito con workload ancora assente richiede il percorso amministrativo di prima creazione. Il retry non annulla lo schema e non equivale al ritorno a una versione precedente.

## Secondo rilascio dello stesso commit: nessun guasto introdotto

È stato preparato un secondo run per provare un'interruzione del solo predeploy ancora in coda, prima del migratore. Il controllo si è fermato su una lettura HTTP 404: **nessuna cancellazione e nessun retry sono stati inviati**. Riscontro del tentativo. Non abbiamo forzato un altro guasto dopo che il lavoro era partito.

La release `access-c3563d26c1112ea1a51d61fa57570f81d386bc0658462e90` ha quindi completato un **secondo rilascio ordinario**, sullo stesso commit Access. Revisione `access-backend-mw91e435` Ready; migratore `access-migrate-7459c` concluso alle **18:29:11.605840 UTC**, prima del deploy backend alle **18:29:22.935710743 UTC**. Ancora nessuna migrazione mancante. Riepilogo separata.

La sonda della seconda revisione ha inizialmente ricevuto **HTTP 403 alla prima richiesta**: la causa non è stata accertata. Una sola ripetizione controllata della sonda, senza modificare Access o ampliare i ruoli, ha completato **360/360 HTTP 200** alle **18:43:23 UTC**. Sono stati riletti 17 nuovi span e metriche applicative per l'istanza `fc261ce8-3ff5-47f2-a5c1-b2b251313c26`, oltre alle metriche SQL proxy. Job, service account e grant temporanei sono rimossi.

Riepilogo finale del secondo run · Richieste · Diagnosi del primo 403 · Cleanup.

È stata ripetuta **la sonda, non il rollout Cloud Deploy**. Il successo successivo non dimostra la causa del 403 iniziale. Le 360 richieste e le tracce della prima sezione restano una prova distinta, riuscita al primo tentativo sulla prima revisione. Il secondo run non dimostra un retry Cloud Deploy positivo né l'ordinamento fra due candidati concorrenti.

## Riferimenti del rilascio

- Commit Infra · Commit Access.
- Hash manifest kit: `3017050e6ea3f06ccdcf9412a3287e340fbc4dd1e2b1d00ce2ec7a51a0be4de7`.
- Publisher: `31b73c94-d3d6-49d7-913d-8c15ba992019`; preparazione: `dfc9b103-8920-4a1e-a107-cd78bcfbf3e6`.
- Release e rollout in Google Cloud.

I collegamenti ai riscontri conducono ai riepiloghi pubblicati. Le ricevute complete restano presso gli operatori; non sono allegati raw di questa guida.

## Collegamento al lavoro successivo

Questo resta il resoconto datato del 9 novembre. Il retry SQL-free è stato poi verificato nella scheda 10, e onboarding Farm con SQL e percorso Platform→Farm nella scheda 11. Il guard di approvazione e il piano operativo sono descritti nelle schede 02, 05 e 12; non cambiano retroattivamente queste prove.

Readonly, eventi, ulteriori scrittori SQL, Console/staff e produzione mantengono le proprie integrazioni. Il collaudo Access non li dichiara automaticamente pronti.

# 10 - Recovery isolato: retry reale verificato

Guida iniziale · Rilasci Access precedenti · Lavoro residuo

**Il 9 novembre abbiamo verificato il recupero di un deploy fallito per un permesso mancante, su un servizio di prova senza SQL.** Il helper comune ha ritentato lo stesso job dello stesso candidato approvato, senza pubblicare un'altra release. Riepilogo finale: PASSED.

## Che cosa abbiamo fatto

La fixture `access-recovery` usa un piccolo pacchetto separato nella repo Access, con sole librerie standard. Ha pipeline `recovery`, runtime `sa-access-recovery`, identità di rilascio, ruoli e archivi dedicati; non usa database o migratore e non modifica il backend Access. Il meccanismo comune non contiene un hook speciale per provocare o aggirare il guasto.

| Passaggio | Riscontro concreto |
|---|---|
| Baseline | Prima revisione Ready: `access-recovery-mw9r8l0e`. La sonda privata ha verificato 10 HTTP 200 con il corpo esatto della fixture. |
| Guasto limitato | Rimosso solo `run.services.update` dal ruolo dedicato `stageRecoveryBackendUpdate`, con etag. Il deploy successivo è fallito con il diniego atteso; la vecchia revisione è rimasta Ready e ha risposto ad altre 10 richieste. |
| Ripristino | Ruolo ripristinato tramite piano Terraform verificato. Riconciliati JobRun/Build falliti e diniego preciso, senza dedurre la sicurezza del retry dal solo tempo trascorso. |
| Retry effettivo | `stage_recovery.py` ha ritentato l'esatto job `stable/deploy`. Nuovo JobRun riuscito alle **21:35:28 UTC**, stesso commit, candidato, bundle, immagini, release e rollout. L'errore originale resta nella storia. |
| Nuova revisione | `access-recovery-mw9rv6ga` Ready con identità e immagine attese, 100% traffico e altre 10 risposte corrette. Una seconda richiesta di retry è stata rifiutata senza nuova modifica cloud. |
| Cleanup | Rimossi Service fixture, tre Job sonda, grant temporanei e finestra IAM iniziale. Backend Access `access-backend-mw9le435` invariato. Piani finali bootstrap/stage: **zero differenze**. |

Verifica del retry · Rifiuto della ripetizione · Riconciliazione.

Le tre fasi hanno verificato **30 risposte HTTP 200**, 10 per fase. La baseline aveva inizialmente ricevuto un 403 prima di una sola ripetizione controllata: la causa non è stata accertata e quel tentativo resta registrato. Non confondiamo questa ripetizione della sonda con il successivo retry Cloud Deploy.

La sonda era nel progetto Platform con `ALL_TRAFFIC`. La rete dell'Execution effettiva è stata verificata anche tramite API v1 quando v2 ometteva `vpcAccess`, confrontando UID e generazione del Job. Non basta leggere la configurazione corrente del Job. Procedura e limiti della sonda.

## Che cosa rimane e che cosa non dimostra

Le protezioni `PREVENT`/`prevent_destroy` restano attive: pipeline/target inattivi, bucket di sorgenti e artefatti, immagine nel registry e identità di catalogo rimangono per audit. I trigger automatici sono disabilitati; non c'è un Service fixture in esecuzione. Non è una promessa di costo zero né la cancellazione di tutte le risorse. Nessun nuovo Cloud SQL è stato creato.

La prova **non certifica** recupero di migrazioni SQL, rollback dello schema, ritorno a una versione applicativa precedente, onboarding da un'altra repository, produzione, concorrenza fra operatori o percorso Platform→Farm. Quest'ultimo è stato poi provato separatamente con Farm, come riporta la scheda 11. Lorenzo verifica le proprie azioni al normale primo uso; non serve una sessione dedicata per procedere e non la dichiariamo già eseguita.

## Codice e ricevute

- Infra PR #28 e #29, commit `8d6187a10d9d9e9d326e1436e106ab70bf5a6a3d`.
- Access PR #31, commit `e17266a3e3ccc7b506de6e816cfca107e67a6d0c`. È il codice della fixture; non significa che il backend Access sia stato aggiornato a questo commit.
- Resoconto ufficiale pubblicato, sezione «Isolated SQL-free recovery», doc PR #93.
- Piano finale bootstrap · Piano finale stage · Service rimosso.

La PR infra #29 corregge un comportamento generale: Google omette la strategia standard vuota nella risposta API; Terraform ora dichiara la strategia esplicita soltanto se ci sono hook predeploy. I servizi senza hook usano il default, evitando un aggiornamento ripetuto a ogni piano. **24 test del modulo**, compreso quello che preserva l'hook delle migrazioni, sono passati.

I collegamenti conducono ai riepiloghi condivisibili delle prove. Gli operatori conservano ricevute complete e hash nell’archivio aziendale; state, piani grezzi e credenziali non vengono pubblicati con la guida.

# 11 - Secondo sistema: onboarding Farm

Guida · Aggiornamento del 10 novembre 2026

**Farm riusa la pipeline di Access con un backend e un migratore reali.** Codice e registrazione sono su `main`, le immagini sono pubblicate, il DNS privato è applicato e la baseline SQL è verificata. **Il primo rollout è riuscito:** migrazione prima del backend e separazione SQL verificata con entrambe le identità. Sono riusciti anche i cinque casi S2S e il cleanup. Finestra IAM chiusa, metriche e una traccia ricevute; il migratore rieseguito non ha applicato nuove versioni. Il prossimo servizio può riusare questo profilo e la stessa procedura.

## Cosa abbiamo fatto

| Parte | Risultato verificato |
|---|---|
| Codice Farm | Skeleton di Lorenzo e adattamenti stage integrati; commit del publisher `fc5d06e` |
| Registrazione | Progetto `moonitoring-farm-stage`, database `farm` su `sql-stage`, identità `sa-farm-api` / `sa-farm-migrate`; nessun nuovo progetto o database |
| Build | Repository Farm autorizzata in Developer Connect; publisher comune riuscito, due immagini fissate per digest |
| Delivery | Profilo comune `service-with-migration`, pipeline `farm` e target `farm-stage`; kit verificato e primo rollout approvato riuscito; migrazione prima del backend |
| Rete | DNS privato `run.app` sulla Shared VPC applicato e riletto; Access invariato. La lettura DNS non è la prova S2S |
| Baseline SQL | Ruoli `farm_owner` / `farm_runtime`, ACL e default verificati; vecchia fixture conservata, catalogo Access invariato |
| Enrollment SQL | Entrambe le identità verificate prima/dopo migrazione; sei altri database negati a ciascuna, runtime senza DDL né accesso al ledger |
| Pulizia SQL | Amministratore, secret, service account, Job e binding temporanei rimossi e assenza riletta; anche i Job di verifica SQL sono rimossi |
| Runtime | `farm-backend-mw9ygxlx` Ready con 100% traffico; migratore `farm-migrate-lbjrc` riuscito. Access invariato |
| Chiusura IAM | Finestra di prima creazione rimossa, grant applicati sui soli `farm-backend` e `farm-migrate` esistenti |

Il backend usa i dati autorizzati senza DDL. Soltanto il migratore assume `farm_owner` per applicare le migrazioni mancanti prima del backend. La baseline infrastrutturale prepara i privilegi; le migrazioni applicative creano lo schema del prodotto. Sono due passaggi distinti.

La vecchia fixture `stage_check` non è stata adottata o cancellata. Ne sono stati confrontati struttura, proprietà, ACL e dipendenze; **le righe non sono state lette né certificate**. Il controllo Access ha ritrovato le stesse 141 relazioni e lo stesso fingerprint. Dettaglio SQL.

## Codice online e sviluppo successivo

Il servizio online di questo collaudo usa `fc5d06e` e due migrazioni. Il main Farm successivo di Lorenzo, `7f386d8`, contiene otto migrazioni: non è diventato online perché abbiamo aggiornato documentazione e test. I test sul database più avanti del codice verificano il guard SeaORM già presente; una nuova versione applicativa continuerà a richiedere la normale release approvata.

## Come riusiamo la base comune

- Cataloghi e moduli Terraform registrano il sistema; non abbiamo creato una seconda implementazione della pipeline per Farm.
- Il Job migratore gira nel progetto Farm, verifica bundle, immagine e identità, serializza le migrazioni e termina il proxy SQL anche in caso di errore.
- Il backend ha minimo zero, massimo due istanze e due connessioni SQL per istanza.

- Il profilo stage comprende backend e migratore. Relay e altri scrittori restano esclusi; Common è fissato alla versione già collaudata con Access.
- Gli eventi automatici dei trigger rimangono spenti. Publisher, preparazione release e rollout usano la procedura manuale approvata.

Dopo il primo onboarding, gli aggiornamenti ordinari riusano il kit senza apply Terraform, salvo nuove risorse o autorizzazioni. Non occorre completare Farm, Console o tutte le schede dei futuri sistemi per lavorare in parallelo.

## Cosa abbiamo verificato

1. **SQL prima e dopo il rilascio: riuscito.** Login privato e membership di entrambe le identità verificati; sei altri database negati a ciascuna. Dopo la migrazione il runtime non può fare DDL né leggere il ledger. Job di verifica rimossi.
2. **Kit e rollout: riusciti.** Il kit ha adottato la build pubblicata. Il migratore ha applicato le due versioni mancanti e terminato il proxy; il backend è diventato Ready successivamente. La finestra IAM è chiusa con i grant nominali.
3. **Platform → Farm: cinque casi riusciti.** Rete `private-ranges-only`, DNS privato e token Google: prima del grant 403, autorizzato 200, senza token 403, audience errata 401, dopo revoca 403. Ogni caso ha provato `/health` e `/ready`; il positivo è correlato con i log Farm. Cinque Job e grant rimossi.
4. **Telemetria e no-op: verificati entro un perimetro preciso.** Ricevute metriche HTTP dei due percorsi e metriche SQL proxy; una traccia applicativa `/health` correlata; la traccia della richiesta `/ready` non risultava presente nell'ultima lettura. La richiesta è provata da HTTP 200, log e metriche. La riesecuzione del migratore ha riportato nessuna migrazione mancante, lasciando Job e backend invariati. L'ha avviata l'identità operatore stage, senza nuovi grant: non prova un secondo rollout Cloud Deploy né l'uso effettivo dei permessi del deployer dopo la chiusura IAM.

Il `verified=false` della ricevuta telemetria indica la seconda traccia mancante, non l'assenza di tutti i segnali: traccia e metriche osservate restano prove dell'arrivo effettivo.

Il kit per i prossimi rilasci usa i contratti successivi alla chiusura IAM. Il kit iniziale e il registro del primo rollout restano immutati. Non occorre un apply per ogni modifica al codice. I piani finali bootstrap e stage, sul commit pulito `1de3800`, non mostrano differenze.

Il Catalog gRPC resta chiuso finché non definiamo audience e chiamante esplicito. Questa prova riguarda rete privata e IAM sui percorsi health/readiness; non certifica le autorizzazioni business. DNS pubblico, bilanciatore, IAP e produzione sono consegne separate.

## Evidenze essenziali

Le PR Farm #9, Infra #30, #31, #32 e Docs #94 sono unite. Gli esiti finali sono pubblicati anche tramite Infra #34 e Docs #95, entrambe unite a main. La #32 è applicata: 35 modifiche bootstrap e 10 stage, senza cancellazioni o modifiche ad Access. La #33 chiude la finestra IAM: dieci risorse IAM aggiunte e rimozione del ruolo e del binding temporanei di prima creazione. L'uso effettivo dei permessi SQL è provato separatamente. Le altre PR Farm di Lorenzo restano distinte.

| Evidenza | Cosa dimostra |
|---|---|
| Test applicazione | Formato/Clippy, 7 test Rust, 14 test hook/render, 21 contratti; Compose con migrazioni, no-op e blocco intenzionale del backend dopo errore SQL |
| Pubblicazione | Build `9f1de53b` riuscita sul commit `fc5d06e`, immagini backend/migratore pubblicate; nessun deployment creato dalla build |
| DNS e S2S/cleanup | Rete privata verificata, cinque casi riusciti, sonde e grant assenti; Access invariato |
| Baseline e cleanup SQL | Apply confermato, metadati fixture e catalogo Access preservati, risorse amministrative temporanee assenti |
| Kit Farm e rollout | Kit verificato; release `farm-b8c3e911…` e rollout `farm-stage-0001` riusciti |
| Migrazione, SQL, no-op | Due migrazioni applicate; privilegi e dinieghi verificati; una riesecuzione senza versioni mancanti |
| Telemetria | Metriche HTTP/proxy ricevute e una delle due tracce precise osservata; la ricevuta mantiene esplicitamente il limite |

Le ricevute tecniche complete restano nell'archivio degli operatori. Questa scheda riporta il risultato e i limiti, senza sostituire ricevute, contratti e procedura comune.

# 12 - Soglie e prova di ripristino dello stage

Guida iniziale · Stato

**Questa scheda definisce una proposta operativa concreta.** I valori della prima tabella sono nella configurazione già applicata; i nuovi allarmi e la prova di restore qui descritti non sono stati attivati o eseguiti. Gli obiettivi riguardano lo stage, non sono SLA di produzione né un dimensionamento per cento allevamenti.

## Base attuale e responsabilità

| Voce | Configurazione di riferimento |
|---|---|
| Budget | 200 USD/mese, avvisi al 50%, 90% e 100% della spesa corrente, crediti esclusi. Filtro sulle cartelle shared e stage; non rappresenta tutta la spesa aziendale |
| Destinatari budget | Email esplicita di Jacopo; destinatari IAM predefiniti mantenuti. Ricezione effettiva del messaggio da collaudare |
| Cloud SQL | `sql-stage`, PostgreSQL 17, 1 vCPU / 3,75 GiB, zonale; disco iniziale 40 GB con crescita fino a 100 GB |
| Backup SQL | Automatico giornaliero nella finestra delle 02:00 UTC, otto backup conservati per conteggio; PITR abilitato con sette giorni di log |
| Backend Access/Farm distribuiti | Minimo zero, massimo configurato due istanze per revisione, pool SQL massimo due connessioni per istanza; nessun nuovo valore applicato con questa scheda |
| Responsabilità | Jacopo e Lorenzo gestiscono configurazione, diagnosi e restore. Chi sviluppa un sistema definisce anche le sue metriche applicative e i bisogni di capacità |

Fonti: budget Terraform, configurazione bootstrap, modulo SQL, configurazione stage.

Le notifiche di budget non sono un tetto automatico di spesa. Anche il limite Cloud Run non va usato come garanzia di costo: durante un rilascio possono coesistere revisioni e il conteggio reale deve essere osservato. Budget Google · Limiti Cloud Run.

## Soglie iniziali proposte

Destinatari proposti: `jtedeschi@moonitoring.com` e `lcipolletti@moonitoring.com`, senza creare un altro gruppo. Un referente prende in carico l'incidente e lo segna nel registro condiviso; stage non implica reperibilità notturna. Prima dell'abilitazione verificare canali e ricevere un messaggio di prova.

| Segnale | Soglia proposta | Azione richiesta |
|---|---|---|
| Spesa | Conservare 100 / 180 / 200 USD effettivi; aggiungere previsione mensile al 100% del budget | Revisione consumi congiunta al 90% o previsione oltre 200 USD; niente spegnimento automatico del DB |
| CPU SQL | Oltre 80% per 15 minuti | Cercare query lente, attese e carichi di build/dataset; aumentare il tier solo dopo misura |
| Disco SQL | Oltre 80% del disco allocato per 10 minuti; urgenza se almeno 90 GB usati rispetto al tetto di 100 GB | Verificare crescita e spazio residuo; niente cancellazione automatica di dati o backup |
| Connessioni SQL | Oltre 70% di `max_connections` per 10 minuti; urgenza oltre 85% per 5 minuti | Verificare pool, revisioni sovrapposte e Job; leggere il limite effettivo, non dedurlo dal numero di aziende |
| Errori backend | 5xx almeno 5% su 5 minuti, con almeno 20 richieste nello stesso intervallo | Correlare revisione, errori SQL e log; distinguere risposte IAM negate attese dai 5xx applicativi |
| Readiness privata | Tre fallimenti consecutivi di una sonda IAM ogni 5 minuti | Diagnosi rete, IAM e DB. La sonda avvia anche istanze a zero: abilitarla consapevolmente, non dichiararla già disponibile |

| Segnale | Soglia proposta | Azione richiesta |
|---|---|---|
| Backup e PITR | Backup fallito, nessun backup riuscito da 30 ore, oppure ultimo istante recuperabile più vecchio di 15 minuti per due controlli consecutivi | Incidente di recuperabilità: leggere lo stato reale delle API, non solo il flag Terraform |

Per percentuali usare finestre e aggregazioni coerenti; un singolo picco non deve diventare un allarme prolungato. Assenza di campioni HTTP su uno stage inattivo non significa errore; un controllo backup mancante è invece un esito incerto da segnalare. I nomi delle metriche, i filtri per progetto/istanza e le query vanno verificati nell'implementazione prima dell'apply.

Il budget delle connessioni comprende tutte le revisioni contemporanee: `somma(istanze per revisione × pool) + pool dei Job concorrenti + riserva`. Manteniamo almeno il 20% del limite per operazioni e margine; un nuovo componente deve rientrare nel totale prima del suo onboarding. Il numero di connessioni non prova capacità di elaborazione, correttezza delle query o isolamento tenant.

## Obiettivi e prova di restore proposta

- **RPO obiettivo: 15 minuti con PITR.** È la perdita massima di dati che vogliamo accettare; va confrontata con l'ultimo istante effettivamente recuperabile.
- **RTO obiettivo: quattro ore dall'avvio della procedura alla verifica della copia.** È l'obiettivo del collaudo tecnico in orario presidiato; non include una promessa di reperibilità o il ripristino completo del prodotto.
- Il ripiego su backup giornaliero ha obiettivo massimo di 24 ore di dati persi, se il backup è disponibile: è una modalità degradata e non soddisfa il RPO PITR.
- Prima prova una volta prima della chiusura operativa, poi dopo cambiamenti sostanziali di backup/SQL e almeno trimestralmente. Preventivo della prova entro 10 USD, da verificare prima di creare la copia; non è un limite di spesa imposto automaticamente da Google.

La destinazione proposta è `sql-stage-restore-YYYYMMDD` in `moonitoring-db-stage`, con IP privato e isolamento da verificare prima della creazione. Non si cambiano DNS o connessioni delle applicazioni correnti. Il PITR crea una nuova istanza e ripristina tutti i suoi database: non sovrascrivere `sql-stage` per recuperare soltanto Farm. PITR Google.

Procedura da rendere eseguibile e collaudare:

1. Leggere backup riusciti e finestra PITR reale, identificare origine, destinazione, orario UTC recuperabile e costo previsto. Progettare e verificare l'isolamento della copia **prima di crearla**: nello stesso progetto i permessi IAM SQL esistenti possono valere anche per la nuova istanza mentre nasce. Se non si può documentare una separazione efficace nella destinazione proposta, predisporre prima una destinazione isolata. Registrare un campione sintetico prima/dopo l'orario scelto, in uno schema di prova separato e senza dati aziendali.
2. Creare la copia con un identificativo univoco e registrare l'operazione API. Un timeout del client richiede riconciliazione, non un secondo restore.
3. Prima di collegare le sonde, verificare rete, login e grant ripristinati. Verificare e rimuovere sulla sola copia gli eventuali login runtime, migratore e altri account applicativi copiati; autorizzare l'identità di collaudo. Confermare l'isolamento deciso al punto 1 prima di proseguire: non cambiare DNS, da solo, non impedisce l'accesso alla copia.
4. Verificare tramite connessione privata versione PostgreSQL, database, schema, migrazioni, ruoli e campione: dato precedente presente, dato successivo assente. Non eseguire automaticamente il migratore più recente sulla copia prima di controllare cosa è stato recuperato. Nessuna email, webhook o consumer reali.
5. Misurare punto recuperato, tempi e risultato delle verifiche. Il test termina con successo solo se contenuto e accessi sono corretti entro gli obiettivi; l'istanza `RUNNABLE` da sola non basta. Un test locale non sostituisce questa prova.
6. Rimuovere solo copia, sonde e permessi temporanei identificati dalla ricevuta; verificare l'assenza delle risorse e che lo stage originale sia invariato. Un'etichetta con scadenza non cancella da sola l'istanza: la pulizia è un passo operativo verificato, anche quando la prova fallisce.

Prima dell'esecuzione fissare nomi, permessi e preventivo. Registrare poi RPO/RTO osservati, costo, esito e cleanup. Raw e dataset su object storage richiedono prove di recupero separate.

**Chiusura:** gli allarmi devono produrre una notifica ricevuta e un rientro visibile; il restore deve recuperare il campione con accessi verificati e cleanup. Fino ad allora soglie e RPO/RTO restano obiettivi documentati, non garanzie.