NoTIP

NO TESTS IN PRODUCTION

Manuale infrastruttura

Versione	1.0.0
Data Modifica	2026-04-13
Utilizzo	Esterno

Abstract dei contenuti

Documento relativo al Manuale Infrastruttura realizzato dal Gruppo NoTIP per il progetto Sistema di Acquisizione Dati da Sensori BLE

Changelog

Versione	Data	Autori	Verificatore	Descrizione
1.0.0	2026-04-13	Leonardo Preo (responsabile)		Approvazione per ingresso in baseline PB
0.5.0	2026-03-15	Alessandro Mazzariol	Leonardo Preo	Aggiunta capitolo relativo a Metriche e Monitoraggio con la descrizione delle relative dashboard Grafana
0.4.0	2026-03-10	Francesco Marcon	Alessandro Mazzariol	Stesura delle sezioni Autenticazione e Keycloak e introduzione dell'utilizzo del Simulatore di Gateway (CLI)
0.3.0	2026-03-01	Leonardo Preo	Mario De Pasquale	Aggiunta sezione sulle Operazioni Quotidiane, comandi di gestione dell'ambiente e reset
0.2.0	2026-02-20	Alessandro Mazzariol	Francesco Marcon	Integrazione Prerequisiti e Avvio del Sistema (Prima Installazione) con generazione dei segreti
0.1.0	2026-02-10	Alessandro Mazzariol	Valerio Solito	Stesura della Panoramica dell'Architettura (struttura generale, Nginx, sicurezza)
0.0.1	2026-02-01	Alessandro Mazzariol	Valerio Solito	Creazione del documento e prima bozza della struttura (Scopo del documento e Glossario)

Indice

Indice delle figure

Indice delle tabelle

1. Introduzione

1.1. Scopo del documento

Il presente documento ha lo scopo di fornire una guida operativa completa per gli amministratori dell'infrastruttura del Sistema NoTIP. Descrive l'architettura dei servizi, le procedure di avvio e gestione, la configurazione della sicurezza e del monitoraggio, e le scelte compiute in ottica di scalabilità. Il manuale è rivolto principalmente agli amministratori di sistema, ma può essere utile anche durante lo sviluppo e il testing.

1.2. Glossario

I termini tecnici rilevanti per la comprensione del manuale sono definiti nella sezione **Glossario** a fondo documento; nel testo tali termini sono contrassegnati con pedice *G*. Per il glossario completo del progetto, si rimanda al Glossario v3.0.0.

1.3. Riferimenti

1.3.1. Riferimenti Informativi

- Glossario v3.0.0
- Repository Notip-infra *Ultimo accesso: 2026-04-09*
- *Docker*$_G$ Documentation *Ultimo accesso: 2026-03-27*
- *NATS*$_G$ *JetStream*$_G$ Documentation *Ultimo accesso: 2026-03-06*

2. Panoramica dell'Architettura

2.1. Struttura generale

L'infrastruttura NoTIP è basata su container *Docker*$_G$ orchestrati tramite *Docker Compose*$_G$. Tutti i servizi comunicano su una rete interna (`internal`) e non sono esposti direttamente all'esterno: il punto di ingresso unico per il traffico HTTP è **Nginx**, che funge da reverse proxy e *API gateway*$_G$.

I servizi che compongono lo stack sono i seguenti:

Servizio	Ruolo
`nginx`	Reverse proxy e *API gateway*$_G$. Instrada il traffico verso i servizi interni.
`frontend`	Applicazione web Angular servita tramite Nginx interno al container.
`management-api`	API REST per la gestione di *tenant*$_G$, *gateway*$_G$ e configurazioni (*NestJS*$_G$).
`data-api`	API REST e streaming SSE per l'accesso ai dati telemetrici (*NestJS*$_G$).
`data-consumer`	Consumatore *NATS*$_G$ → *TimescaleDB*$_G$. Persiste la telemetria e monitora i *gateway*$_G$ (Go).
`provisioning`$_G$`-service`	Servizio di onboarding *gateway*$_G$: emette certificati *TLS*$_G$ e chiavi AES (*NestJS*$_G$).
`nats`$_G$	*Message broker*$_G$ con *JetStream*$_G$ abilitato. Usa *mTLS*$_G$ per autenticare i client.

Servizio	Ruolo
`mgmt-db`	Database *PostgreSQL*$_G$ per i dati gestionali (*tenant*$_G$, *gateway*$_G$, configurazioni).
`measures-db`	Database *TimescaleDB*$_G$ (*PostgreSQL*$_G$ + estensione time-series) per i dati telemetrici.
`keycloak`$_G$`-db`	Database *PostgreSQL*$_G$ dedicato a *Keycloak*$_G$.
`keycloak`$_G$	Identity Provider *OIDC*$_G$. Gestisce autenticazione e autorizzazione degli utenti.
`simulator`	Simulatore di *gateway*$_G$ IoT BLE (Go). Attivabile tramite profilo *Docker Compose*$_G$.
`sim-cli`	Interfaccia a riga di comando per controllare il simulatore.

Oltre ai servizi applicativi, lo stack prevede tre servizi di **inizializzazione one-shot** che vengono eseguiti una sola volta all'avvio e poi terminano:

- **`provisioning`$_G$`-init`**: genera la CA interna e i certificati *mTLS*$_G$ per tutti i servizi.
- **`nats`$_G$`-streams-init`**: crea gli stream *JetStream*$_G$ su *NATS*$_G$ (TELEMETRY, ALERTS, COMMANDS, ecc.).
- **`keycloak`$_G$`-init`**: importa il realm `notip` con client, ruoli e configurazioni in *Keycloak*$_G$.

2.2. Routing Nginx

Nginx espone la porta `80` dell'host e instrada le richieste come segue:

Percorso	Destinazione
`/auth/`	*Keycloak*$_G$ (`:8080`) — autenticazione *OIDC*$_G$.
`/api/mgmt/`	Management API (`:3000`) — gestione *tenant*$_G$ e *gateway*$_G$.
`/api/data/`	Data API (`:3000`) — query e streaming dati telemetrici.
`/api/data/*/stream`	Data API — endpoint SSE con buffering disabilitato per streaming in tempo reale.
`/api/provision/`	*Provisioning*$_G$ Service (`:3000`) — onboarding dei *gateway*$_G$.
`/`	Frontend Angular (`:8080`).
`/internal/`	Bloccato (404) — rotte interne mai esposte.

2.3. Sicurezza

L'infrastruttura implementa più livelli di sicurezza:

- ***mTLS*$_G$ su *NATS*$_G$**: ogni servizio backend si autentica con un certificato firmato dalla CA interna. *NATS*$_G$ verifica il certificato del client (`verify_and_map`) e associa i permessi tramite il Distinguished Name del certificato stesso.
- ***Keycloak*$_G$**: gestisce l'autenticazione degli utenti e dei client *OAuth2*$_G$ tramite token *JWT*$_G$. I servizi backend validano i token prima di elaborare le richieste.
- ***Docker*$_G$ Secrets**: le password più sensibili (ad esempio `measures_db_password`) vengono montate come file segreti nei container, evitando l'esposizione tramite variabili d'ambiente.
- **Cifratura a riposo**: le chiavi AES dei *gateway*$_G$ sono cifrate nel database con una chiave `DB_ENCRYPTION_KEY` (AES-256). La perdita di questa chiave rende irrecuperabili tutte le chiavi dei *gateway*$_G$.

- **Security headers HTTP**: Nginx aggiunge intestazioni di sicurezza (`X-Content-Type-Options`, `Referrer-Policy`) a tutte le risposte.

3. Prerequisiti

Prima di avviare il sistema, assicurarsi di avere installato:

- ***Docker***$_G$ (con *Docker Compose*$_G$ v2): Installazione *Docker*$_G$ (*Ultimo accesso: 2026-03-26*)
- **Make**: Installazione Make (*Ultimo accesso: 2026-03-20*)
- Terminale **Bash** o **PowerShell** >= **7.0**
- Accesso al repository: Notip-infra Repository

4. Avvio del Sistema (Prima Installazione)

Tutti i comandi devono essere eseguiti dalla directory `infra/`.

4.1. Passo 1 — Generazione dei segreti e del file `.env`

```
make bootstrap
```

Questo comando copia `.env.example` in `.env` e genera automaticamente valori casuali per tutti i segreti (password dei database, segreti *OAuth2*$_G$ *Keycloak*$_G$, chiave di cifratura AES). Al termine, aprire il file `.env` e verificare i valori non-segreti (hostname, nomi dei database, ecc.).

Variabile	Descrizione
`DB_ENCRYPTION_KEY`	Chiave AES-256 per cifrare le chiavi *gateway*$_G$ a riposo.
`MGMT_DB_PASSWORD`	Password del database *PostgreSQL*$_G$ gestionale.
`MEASURES_DB_PASSWORD`	Password di *TimescaleDB*$_G$. Scritta anche in `secrets/`.
`KEYCLOAK_ADMIN_PASSWORD`	Password dell'admin bootstrap di *Keycloak*$_G$.
`KEYCLOAK_MGMT_CLIENT_SECRET`	Segreto *OAuth2*$_G$ del client `notip-mgmt-backend`.
`KEYCLOAK_SIMULATOR_CLIENT_SECRET`	Segreto *OAuth2*$_G$ del client del simulatore.
`KEYCLOAK_DB_PASSWORD`	Password del database *PostgreSQL*$_G$ di *Keycloak*$_G$.

Tabella 1: Variabili generate da `make bootstrap`

Attenzione: eseguire di nuovo `make bootstrap` con i volumi *Docker*$_G$ già presenti sovrascrive i segreti nel `.env` ma **non** aggiorna i dati già scritti nei database, causando inconsistenze. Eseguirlo solo su un ambiente pulito oppure dopo `make reset-all`.

4.2. Passo 2 — Avvio dello stack

```
make up
```

Docker Compose$_G$ scarica le immagini più recenti da `ghcr.io/notipswe` e avvia tutti i container nell'ordine corretto, rispettando le dipendenze (healthcheck inclusi). L'avvio completo richiede alcuni minuti, in particolare per *Keycloak*$_G$.

Al termine, il frontend è raggiungibile all'indirizzo `http://localhost/`.

4.3. Passo 3 — Esecuzione delle migrazioni del database

Al primo avvio è necessario applicare le migrazioni TypeORM ai database:

```
make migration-run-all
```

In alternativa, i passi 2 e 3 possono essere combinati in un unico comando:

```
make up-with-migrations
```

4.4. Passo 4 — Verifica dello stato dei servizi

```
make health
```

Lo script controlla lo stato di tutti i container: i servizi applicativi devono essere in stato `running/healthy`, mentre i servizi one-shot (`provisioning`$_G$`-init`, `nats`$_G$`-streams-init`, `keycloak`$_G$`-init`) devono risultare `exited` con codice di uscita `0`.

5. Operazioni Quotidiane

5.1. Comandi principali

Tutti i comandi vanno eseguiti dalla directory `infra/`.

Comando	Effetto
`make up`	Avvia (o riavvia) tutti i container. Non tocca i volumi.
`make down`	Ferma e rimuove i container. I volumi (e i dati) vengono preservati.
`make ps`	Mostra lo stato di tutti i container.
`make logs`	Segue i log di tutti i servizi in tempo reale.
`make logs-svc SVC=<nome>`	Segue i log di un singolo servizio, es. `SVC=data-api`.
`make health`	Controlla lo stato di salute di tutti i servizi.
`make keycloak`$_G$`-import`	Reimporta la configurazione del realm *Keycloak*$_G$ da `infra/keycloak`$_G$`/`.
`make lint`	Esegue i controlli di qualità del codice tramite `pre-commit`.

5.2. Reset dell'ambiente

Quando è necessario ripartire da zero, esistono due livelli di reset:

- **`make reset`**: ferma i container, elimina tutti i volumi *Docker*$_G$ tranne `ca_certs` (i certificati *gateway*$_G$ vengono preservati), e riavvia lo stack.
- **`make reset-all`**: elimina **tutto**, incluso il volume della CA interna. Questo invalida tutti i certificati *gateway*$_G$ precedentemente emessi. Richiede conferma esplicita.

5.3. Sviluppo con immagini locali

Per sostituire una o più immagini con build locali (utile durante lo sviluppo), usare:

```
# Singolo servizio
make up-local LOCAL=management-api

# Più servizi (separati da virgola)
make up-local LOCAL=management-api,data-api
```

Valori accettati: `management-api`, `data-api`, `data-consumer`, `provisioning`$_G$`-service`, `frontend`, `simulator`, `sim-cli`.

Il Makefile compila l'immagine locale con `docker`$_G$ `build --target prod` a partire dalla directory del repository sibling corrispondente, quindi avvia lo stack con quell'immagine al posto di quella remota.

5.4. Gestione delle migrazioni del database

Comando	Effetto
`make migration-run-all`	Applica le migrazioni pending su tutti i database.
`make migration-revert-all`	Annulla l'ultima migrazione su tutti i database.
`make migration-run-management`	Applica le migrazioni solo al database gestionale.
`make migration-run-data`	Applica le migrazioni solo al database delle misure.
`make migration-revert-management`	Annulla l'ultima migrazione del database gestionale.
`make migration-revert-data`	Annulla l'ultima migrazione del database delle misure.

6. Autenticazione e *Keycloak*$_G$

6.1. Accesso al pannello di amministrazione

Per accedere all'area riservata di *Keycloak*$_G$, aprire l'URL `http://localhost/auth`. Inserire le credenziali presenti nel file `.env` generato da `make bootstrap`:

- **Username**: valore di `KEYCLOAK_ADMIN_USER` (default: `admin`).
- **Password**: valore di `KEYCLOAK_ADMIN_PASSWORD`.

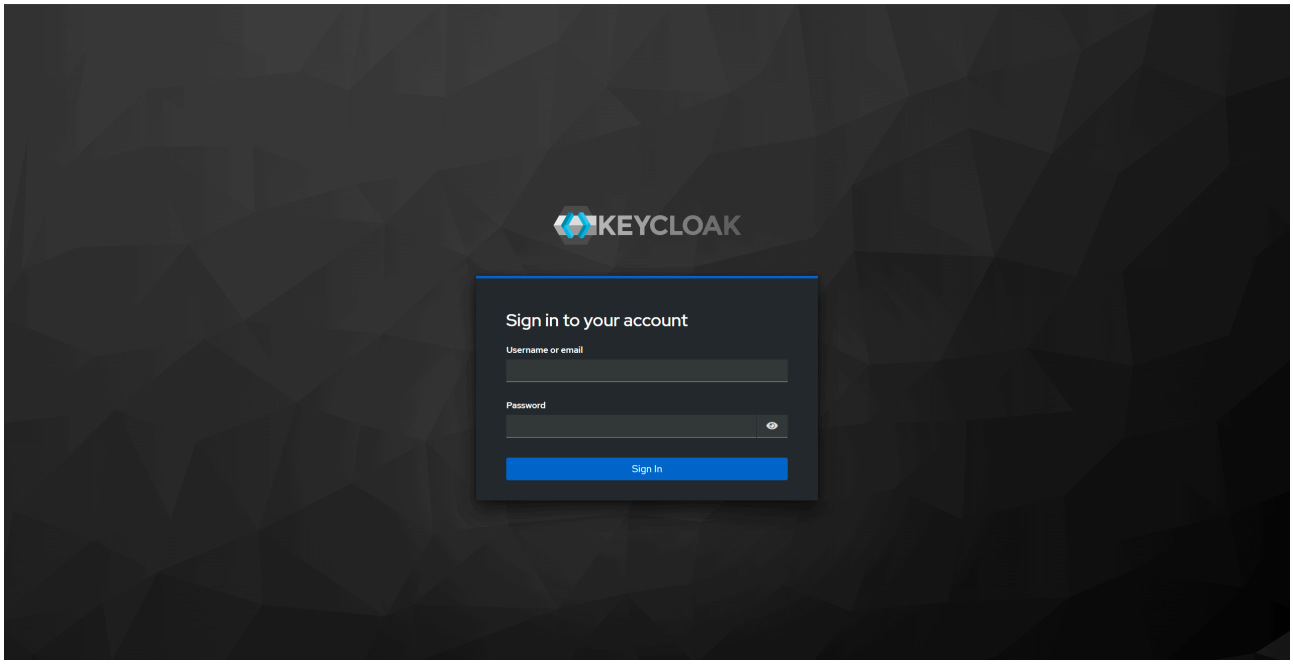

Figura 1: *Keycloak*$_G$ Login

6.2. Configurazione dell'utente amministratore definitivo

Al primo accesso è necessario creare un utente amministratore definitivo:

1. Cliccare su **Users** → **Add user** e creare un nuovo utente con:
   - **Username**: `administrator`.
   - **Email**: una email valida (es. con dominio `notip.it`).
   - **Password**: una nuova password sicura (oppure la stessa di `KEYCLOAK_ADMIN_PASSWORD`).
2. Fare logout e accedere con l'account appena creato.
3. Eliminare l'utente temporaneo di default.

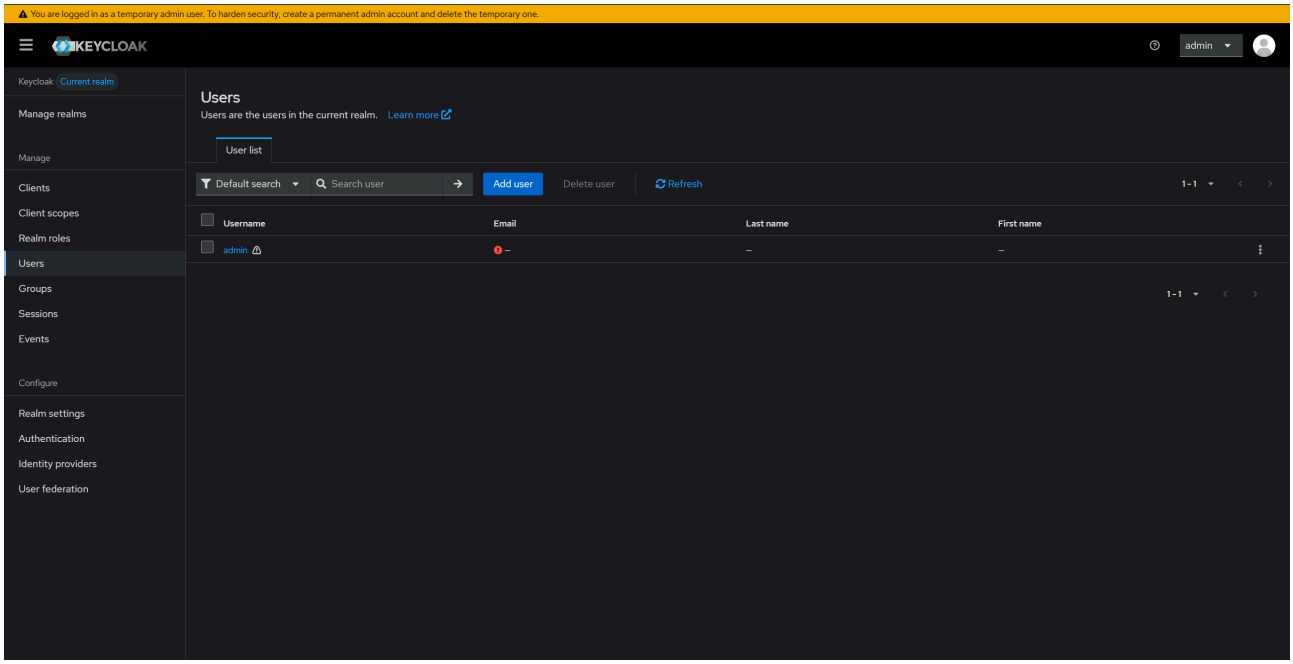

Figura 2: *Keycloak*$_G$ utente temporaneo

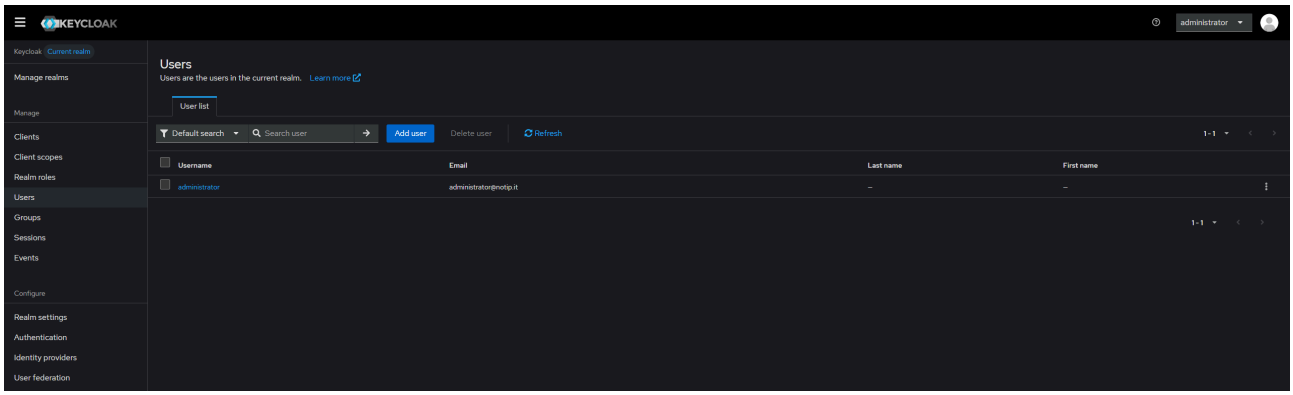

Figura 3: *Keycloak*$_G$ utente definitivo

6.3. Realm `notip`

All'avvio del sistema, lo script `keycloak`$_G$`-init` importa automaticamente il realm `notip` con client, ruoli e mappature preconfigurati. Per visualizzare i dettagli, accedere a **Manage Realms** → `notip`.

Per modificare la configurazione di *Keycloak*$_G$ è possibile:

- Usare l'interfaccia grafica di *Keycloak*$_G$ (le modifiche vengono applicate immediatamente).
- Modificare il file JSON di esportazione in `infra/keycloak`$_G$`/realm-export.json` e reimportarlo con `make keycloak`$_G$`-import`.

6.4. Accesso al Frontend

Per accedere all'applicazione web, navigare su `http://localhost/` e autenticarsi con:

- **Username**: `admin`.
- **Password**: valore di `KEYCLOAK_ADMIN_PASSWORD` dal file `.env`.

7. Simulatore di *Gateway*$_G$

Il simulatore è un componente opzionale che permette di simulare *gateway*$_G$ IoT BLE senza hardware fisico. Viene avviato tramite il profilo *Docker Compose*$_G$ `simulator`.

7.1. Avvio del simulatore

Il simulatore è già incluso nello stack standard avviato con `make up` (il Makefile include il profilo `--profile simulator` di default). Il servizio `simulator` è accessibile internamente sulla porta `8090`, ma non è esposto tramite Nginx: vi si accede tramite il `sim-cli`.

7.2. Utilizzo del Simulator CLI

Il `sim-cli` permette di gestire il parco *gateway*$_G$ simulati. Lo stack principale deve essere già in esecuzione.

Aprire una sessione interattiva:

```
cd infra
docker_G compose --project-directory . -f compose/docker_G-compose.yml run --rm sim-cli shell
```

All'interno della shell sono disponibili i seguenti comandi:

Comando	Descrizione
`gateways list`	Elenca tutti i *gateway*$_G$ simulati registrati.
`gateways create --factory-id ID --factory-key KEY --model M --firmware F --freq N`	Crea un singolo *gateway*$_G$ simulato.
`gateways bulk ...`	Crea N *gateway*$_G$ simulati, in base al numero di factory ids inseriti, con i parametri specificati.
`gateways delete <uuid>`	Elimina un *gateway*$_G$ simulato.
`sensors add <gateway_G-id> --type TYPE --min N --max N --algorithm ALG`	Aggiunge un sensore a un *gateway*$_G$ esistente.
`anomalies disconnect <uuid> --duration N`	Simula una disconnessione del *gateway*$_G$ per N secondi.

In alternativa, è possibile eseguire un singolo comando senza aprire la shell:

```
docker_G compose --project-directory . -f compose/docker_G-compose.yml run --rm -it sim-cli
gateways list
```

NB: per la creazione dei *gateway*$_G$ nella `sim-cli` è prima necessario che questi vengano creati ed assegnati ad un *Tenant*$_G$ dal System Admin lato Frontend. In caso contrario, all'invio del comando nella `sim-cli` per la creazione dei *gateway*$_G$, si otterrà un errore di validazione da parte del *Provisioning*$_G$ Service, in quanto il `factory_id` e `factory_key` non saranno riconosciuti.

```
ERROR   Failed to create gateway_G
Error: backend returned 401: onboard: invalid factory credentials:
{"error":"INVALID_CREDENTIALS"}
```

8. Metriche e Monitoraggio

8.1. Avvio dello stack di monitoraggio

Il monitoraggio (Prometheus + Grafana) è separato dallo stack principale e va avviato esplicitamente:

```
make up-monitoring
```

Per fermarlo:

```
make down-monitoring
```

Lo stack di monitoraggio si collega alla rete interna *Docker*$_G$ del progetto principale per poter raccogliere le metriche dai servizi.

8.2. Accesso a Grafana

Navigare su `http://localhost:13000/` e autenticarsi con:

- **Username**: `admin`.
- **Password**: `admin`.

Al primo accesso verrà richiesto di cambiare la password.

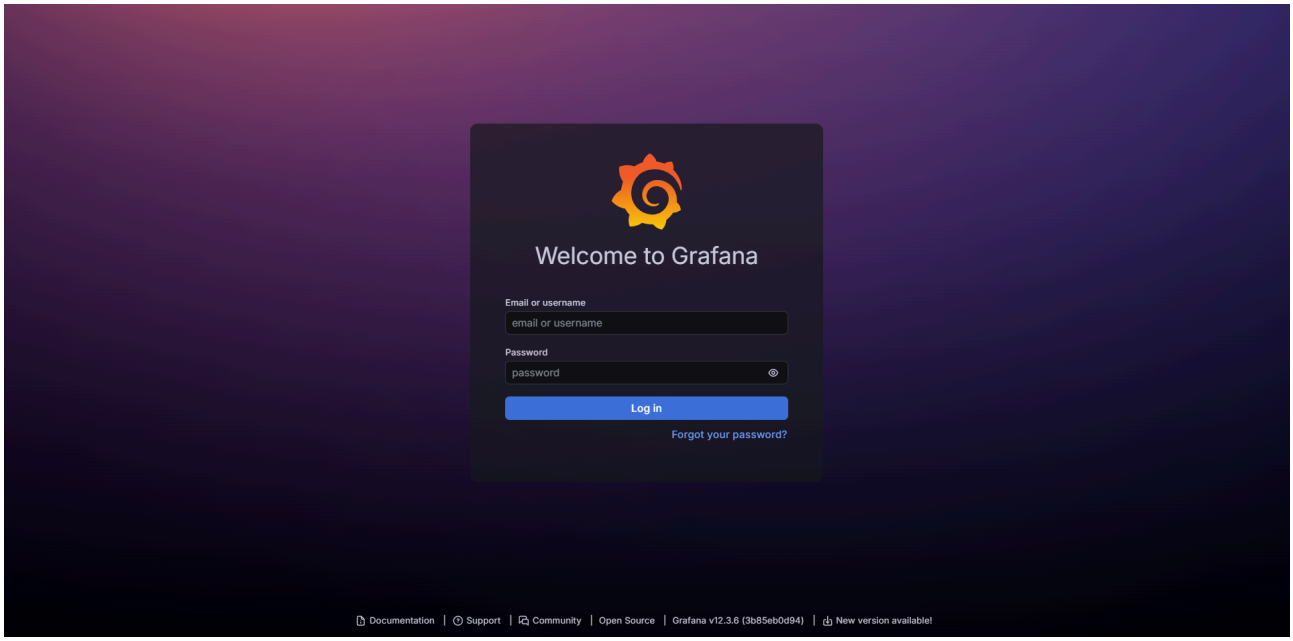

Figura 4: Grafana Login

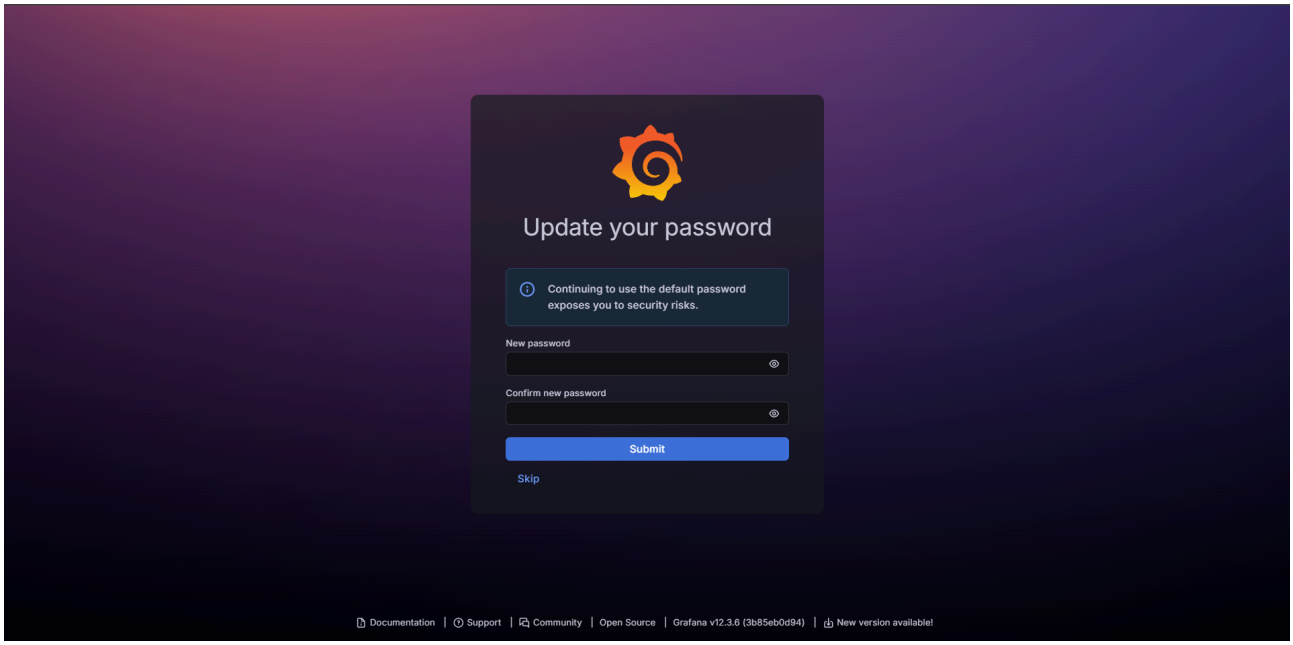

Figura 5: Grafana cambio password

Una volta effettuato l'accesso, navigare nella sezione **Dashboards**:

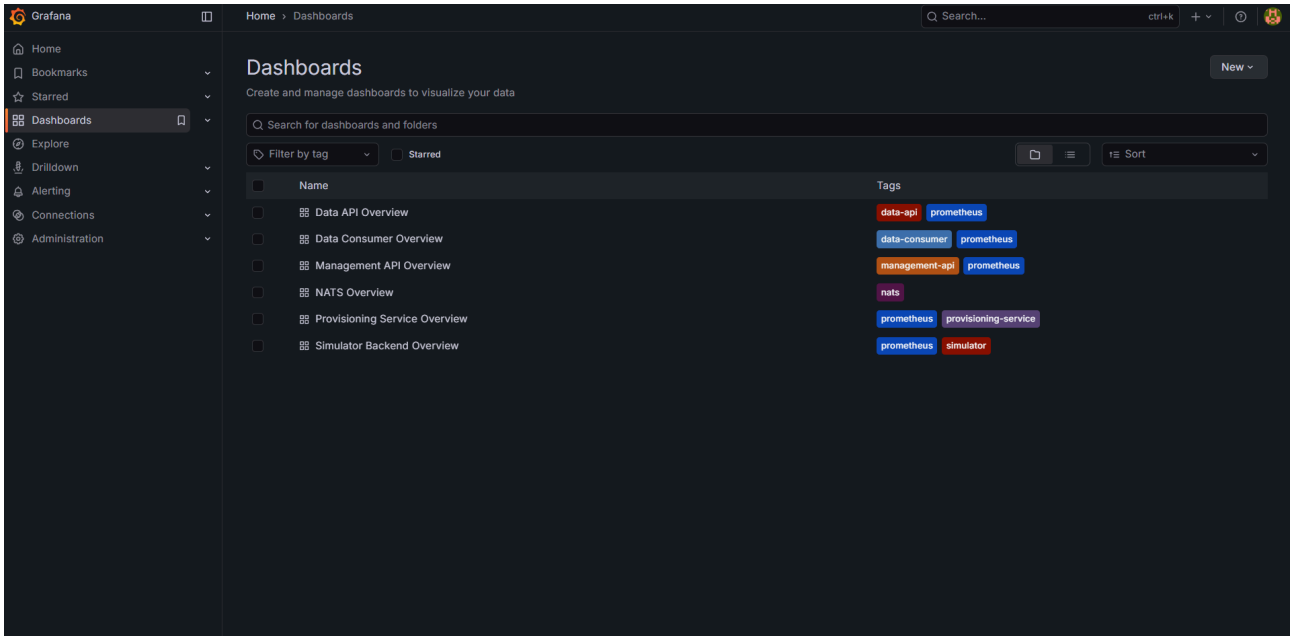

Figura 6: Grafana Dashboards

8.3. Dashboard disponibili

8.3.1. Dashboard Data API e Management API

Figura 7: Dashboard Data API e Management API

Le misure di monitoraggio presenti in questa dashboard sono:

- **HTTP Request Rate**: numero di richieste HTTP ricevute al secondo.
- **HTTP Error Rates**: numero di errori HTTP (4xx e 5xx) ogni 15 secondi.
- **HTTP Latency**: tempo medio di risposta delle richieste HTTP ogni 15 secondi.
- **In-flight Requests**: numero di richieste HTTP attualmente in elaborazione (tempo reale).
- **Process Memory**: memoria utilizzata dal processo in bytes (tempo reale).
- **CPU Usage**: percentuale di utilizzo della CPU (tempo reale).

8.3.2. Dashboard Data Consumer

Figura 8: Dashboard Data Consumer

Le misure di monitoraggio presenti in questa dashboard sono:

- **Telemetry Throughput**: numero di messaggi telemetrici processati ogni 5 secondi.
- **Error Rates**: numero di errori ogni 5 secondi.
- ***TimescaleDB*$_G$ Write Latency**: tempo medio di scrittura su *TimescaleDB*$_G$ ogni 5 secondi.
- **Tracked Gateways**: numero di *gateway*$_G$ attualmente monitorati (tempo reale).
- **Backpressure and Connectivity**: presenza di backpressure o problemi di connettività verso *TimescaleDB*$_G$.
- **Alerts and Config Cache Errors**: errori nella cache di configurazione e alert generati ogni 5 secondi.

8.3.3. Dashboard *Provisioning*$_G$ Service

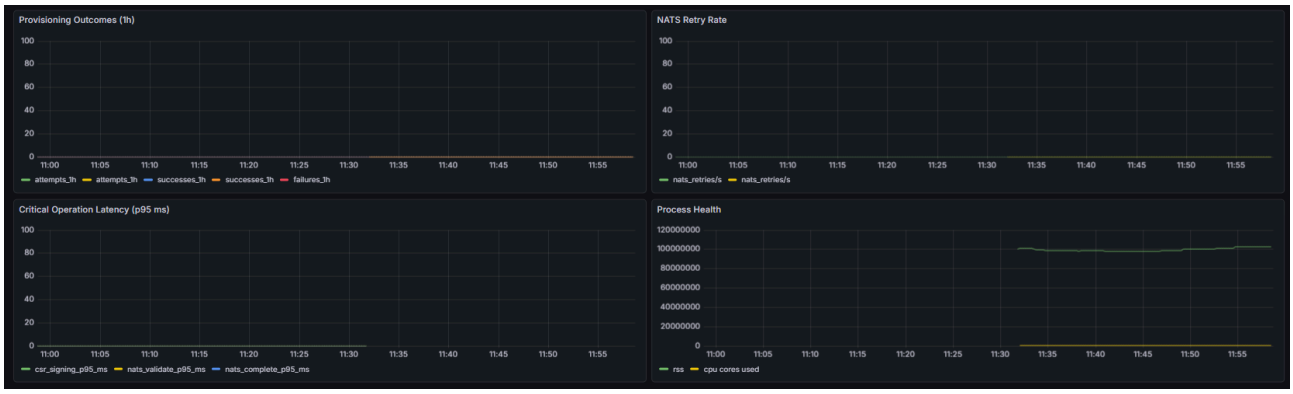

Figura 9: Dashboard *Provisioning*$_G$ Service

Le misure di monitoraggio presenti in questa dashboard sono:

- ***Provisioning*$_G$ Outcomes**: operazioni di *provisioning*$_G$ tentate, riuscite e fallite ogni 5 secondi.
- ***NATS*$_G$ Retry Rate**: tentativi di retry per la connessione a *NATS*$_G$ ogni 5 secondi.
- **Critical Operation Latency**: tempo medio di completamento delle operazioni critiche (firma CSR, validazione *NATS*$_G$, completamento *NATS*$_G$) ogni 5 secondi.
- **Process Health**: salute complessiva del processo basata su metriche di performance ed errori.

8.3.4. Dashboard Simulatore

Figura 10: Dashboard Simulatore

Le misure di monitoraggio presenti in questa dashboard sono:

- **Gateways Running**: numero di istanze del simulatore in esecuzione (tempo reale).
- **Publish Throughput and Errors**: messaggi pubblicati ed errori di pubblicazione ogni 5 secondi.
- **Buffer Health**: salute del buffer del simulatore basata su utilizzo memoria e latenza.
- ***Provisioning*$_G$**: operazioni di *provisioning*$_G$ riuscite e fallite per i dispositivi simulati ogni 5 secondi.
- ***NATS*$_G$ Reconnects**: riconnessioni a *NATS*$_G$ ogni 5 secondi.
- **Anomalies Injected**: anomalie simulate (dati fuori soglia, perdita di pacchetti) ogni 5 secondi.

8.3.5. *NATS*$_G$ Monitoring

Figura 11: Dashboard *NATS*$_G$

Le misure di monitoraggio presenti in questa dashboard sono:

- **Connections**: numero di connessioni attive al server *NATS*$_G$ (tempo reale).

9. Scalabilità dell'Infrastruttura

L'infrastruttura NoTIP è progettata per supportare una crescita graduale del numero di *gateway*$_G$, *tenant*$_G$ e volumi di dati. Di seguito vengono descritte le scelte architetturali che abilitano la scalabilità.

9.1. Disaccoppiamento tramite *NATS*$_G$ *JetStream*$_G$

Il cuore della pipeline di acquisizione dati è ***NATS*$_G$ *JetStream*$_G$**, un *message broker*$_G$ persistente che disaccoppia i produttori di dati (*gateway*$_G$ e simulatore) dai consumatori (Data Consumer, Data API).

Gli stream *JetStream*$_G$ sono configurati con politiche di retention basate su limiti temporali:

Stream	Soggetti	Retention / Note
`TELEMETRY`	`telemetry.data.>`	30 giorni. Storage su file. Dati telemetrici dai *gateway*$_G$.
`ALERTS`	`alert.>`	Illimitata. Alert generati dal Data Consumer.
`COMMANDS`	`command.gw.>`	75 secondi. Comandi inviati ai *gateway*$_G$ (breve TTL).
`COMMAND_ACKS`	`command.ack.>`	Acknowledgment dei comandi ricevuti dai *gateway*$_G$.
`AUDIT_LOG`	`log.audit.>`	90 giorni. Audit log delle operazioni.
`DECOMMISSION`	`gateway`$_G$`.decommissioned.>`	24 ore. Notifiche di decommissioning *gateway*$_G$.

Grazie alla persistenza su file di *JetStream*$_G$, i messaggi non vengono persi in caso di riavvio del Data Consumer. Il broker assorbe i picchi di carico e garantisce la consegna anche in caso di temporanea indisponibilità dei consumer.

9.2. *TimescaleDB*$_G$ per i dati telemetrici

I dati telemetrici sono archiviati su ***TimescaleDB*$_G$**, un'estensione di *PostgreSQL*$_G$ ottimizzata per serie temporali. Le principali caratteristiche che abilitano la scalabilità sono:

- **Hypertables**: partizionamento automatico dei dati per intervalli temporali, che mantiene le performance di query costanti all'aumentare del volume di dati.
- **Compressione nativa**: possibilità di comprimere automaticamente i chunk più vecchi, riducendo lo spazio su disco.
- **Separazione del database delle misure**: *TimescaleDB*$_G$ è un servizio indipendente da *PostgreSQL*$_G$ gestionale, che permette di scalare i due database separatamente in base al carico.

9.3. Separazione dei database

L'infrastruttura utilizza **tre database distinti**:

- **`keycloak`$_G$`-db`**: esclusivamente per *Keycloak*$_G$. Isola i dati di autenticazione dal resto del sistema.
- **`mgmt-db`**: dati gestionali (*tenant*$_G$, *gateway*$_G$, configurazioni). Volume di dati moderato e a bassa frequenza di scrittura.
- **`measures-db`**: dati telemetrici. Alto volume, alta frequenza di scrittura. Ottimizzato con *TimescaleDB*$_G$.

Questa separazione consente di allocare risorse (CPU, RAM, storage) in modo indipendente e di applicare policy di backup differenziate in base alla criticità e alla frequenza di aggiornamento.

9.4. Buffer e backpressure nel Data Consumer

Il Data Consumer gestisce il flusso di dati in ingresso da *NATS*$_G$ con un meccanismo di buffering configurabile tramite variabili d'ambiente:

- `GATEWAY_BUFFER_SIZE` (default: `1000`): dimensione del buffer per *gateway*$_G$. Aumentare questo valore consente di assorbire picchi di telemetria senza perdita di dati.
- `HEARTBEAT_GRACE_PERIOD_MS` (default: `120000` ms): periodo di grazia prima di considerare un *gateway*$_G$ offline. Permette di tollerare temporanee interruzioni di connettività.
- `ALERT_CONFIG_MAX_RETRIES` e `ALERT_CONFIG_MAX_BACKOFF_MS`: configurano il comportamento di retry per il recupero delle configurazioni di alerting, con backoff esponenziale.

La dashboard Grafana **Data Consumer** espone la metrica **Backpressure and Connectivity** per rilevare situazioni di sovraccarico e intervenire tempestivamente.

9.5. Containerizzazione e portabilità

Tutti i servizi sono distribuiti come immagini *Docker*$_G$ (`linux/amd64`) e pubblicati su GitHub Container Registry (`ghcr.io/notipswe`). Questa scelta consente di:

- Spostare lo stack su qualsiasi host che esegua *Docker*$_G$, indipendentemente dall'architettura hardware.
- Aggiornare singoli servizi sostituendo l'immagine senza interrompere gli altri.
- Effettuare il pin di versioni specifiche tramite le variabili `*_IMAGE` nel file `.env`.

9.6. Limiti dello stack attuale e percorso di crescita

Lo stack *Docker Compose*$_G$ attuale è ottimizzato per un singolo host. Per una crescita ulteriore verso ambienti multi-nodo, i componenti che beneficerebbero maggiormente dello scaling orizzontale sono:

Componente	Percorso di scala
Management API / Data API	Istanze multiple dietro un load balancer (Nginx upstream group o un orchestratore come Kubernetes).
NATS$_G$ *JetStream*$_G$	Cluster *NATS*$_G$ con `num_replicas > 1` per alta disponibilità degli stream.
TimescaleDB$_G$	Replica in lettura per distribuire il carico delle query. Upgrade a Timescale Cloud per scaling gestito.
Data Consumer	Più istanze con partizionamento dei soggetti *NATS*$_G$ per distribuire la pipeline di persistenza.

10. Troubleshooting

10.1. Container bloccato in attesa o in crash loop

Sintomi: `make ps` mostra un servizio in stato `restarting` oppure `make health` riporta un errore.

Soluzione:

1. Eseguire `make logs-svc SVC=<nome-servizio>` per leggere i log del container problematico.

2. Se il problema è una dipendenza non ancora pronta (es. database non healthy), attendere qualche secondo e riprovare con `make health`.
3. Se il problema persiste, eseguire `make down && make up` per riavviare lo stack.
4. Se i volumi sono corrotti o i segreti sono cambiati, eseguire `make reset` per ripulire i volumi (tranne i certificati CA) e ripartire.

10.2. HTTPS Required al momento dell'accesso a *Keycloak*$_G$ (macOS)

Sintomi: Il browser viene reindirizzato a una pagina HTTPS che non risponde.

Soluzione: Chiudere e riavviare *Docker*$_G$ Desktop, quindi eseguire `make reset-all && make up` e restartare il browser.

Via terminale:

```
osascript -e 'quit app "Docker"' && sleep 3 && open -a Docker
```

Attendere che *Docker*$_G$ si avvii, poi:

```
make reset-all && make up
```

10.3. Segreti non allineati tra `.env` e database

Sintomi: I servizi si avviano ma falliscono la connessione al database o a *Keycloak*$_G$ con errori di autenticazione.

Causa: `make bootstrap` è stato rieseguito con volumi già presenti, sovrascrivendo le password nel `.env` senza aggiornare quelle già scritte nei database.

Soluzione: Eseguire `make reset-all` (che elimina tutti i volumi) e poi ricominciare dalla procedura di installazione al Passo 1.

10.4. Importazione del realm *Keycloak*$_G$ fallita

Sintomi: `make keycloak-import` termina con errori, oppure il realm `notip` non compare nell'interfaccia di *Keycloak*$_G$.

Soluzione:
1. Verificare che *Keycloak*$_G$ sia in stato `healthy` con `make ps`.
2. Controllare i log di *Keycloak*$_G$ con `make logs-svc SVC=keycloak`.
3. Riprovare l'importazione con `make keycloak-import` una volta che *Keycloak*$_G$ è completamente avviato.

10.5. Il simulatore non invia dati

Sintomi: La dashboard Grafana **Simulatore** mostra `Gateways Running = 0` oppure `Publish Throughput = 0`.

Soluzione:
1. Verificare che il servizio `simulator` sia in esecuzione con `make ps`.
2. Controllare i log con `make logs-svc SVC=simulator`.
3. Usare `sim-cli` per verificare che esistano *gateway*$_G$ registrati: `gateways list`.
4. Se non ci sono *gateway*$_G$, crearne uno con `gateways create`.

11. Glossario

API Gateway Componente architetturale che funge da punto di accesso centralizzato per le API, gestendo autenticazione, autorizzazione e rate limiting delle richieste.

Docker Tecnologia di containerizzazione che consente di confezionare applicazioni e le loro dipendenze in container portabili, facilitando deployment coerente su diverse piattaforme.

Docker Compose Strumento che facilita la definizione e l'esecuzione di applicazioni multi-contenitore *Docker* attraverso file di configurazione YAML.

Gateway Dispositivo fisico o software che funge da punto di accesso e intermediario per la comunicazione tra reti, sensori o sistemi diversi.

JWT Acronimo di JSON Web Token, standard aperto per la creazione di token di accesso che consentono l'autenticazione e lo scambio sicuro di informazioni tra parti.

JetStream Estensione nativa di *NATS* che aggiunge persistenza dei dati e consegna garantita, permettendo l'archiviazione dei messaggi e il loro recupero successivo.

KeyCloak Piattaforma Open Source che permette di centralizzare l'autenticazione e l'autorizzazione per applicazioni e servizi moderni.

Message Broker Componente architetturale che gestisce l'ingestione, buffering e distribuzione di messaggi tra produttori e consumatori.

Multi-tenancy Architettura in cui una singola istanza dell'applicazione serve molteplici *tenant* con segregazione completa dei dati e delle risorse.

NATS Sistema di messaggistica Open Source sviluppato da Cloud Native Computing Foundation.

NestJS Framework Node.js basato su TypeScript per la costruzione di applicazioni server-side scalabili con architettura modulare e dependency injection.

OAuth2 Standard di autorizzazione che consente l'accesso delegato ai servizi, permettendo l'autenticazione tramite provider terzi senza esporre le credenziali.

OIDC Acronimo di OpenID Connect, livello di identità costruito sopra *OAuth2* che consente di autenticare un utente e ottenere informazioni standard sulla sua identità.

PostgreSQL Database relazionale open source caratterizzato da robustezza, conformità SQL avanzata e supporto di estensioni per casi d'uso specializzati.

Provisioning Processo di allocazione e configurazione di risorse necessarie per il funzionamento di un sistema.

TLS Acronimo di Transport Layer Security, protocollo crittografico che garantisce comunicazione sicura su rete proteggendo dati in transito.

Tenant Entità cliente in un'architettura *multi-tenancy* che condivide l'infrastruttura ma con segregazione completa dei dati e delle risorse.

TimescaleDB Estensione *PostgreSQL* specializzata per dati time-series, ottimizzata per archiviazione efficiente e query ad alte prestazioni su dati temporali.

mTLS Acronimo di mutual *TLS*, estensione del protocollo *TLS* in cui sia il client sia il server si autenticano reciprocamente tramite certificati digitali.