Università degli Studi di Padova

GlitchHub TEAM

Verbale esterno 27/01/2026 (M31)

Ordine del giorno

1. Presentazione dello stato avanzamento lavori
2. Comprensione del **simulatore di gateway**$_G$
3. Discussione e dubbi sulla struttura del **PoC**$_G$
4. Separazione dei dati tra **tenant**$_G$
5. Gestione perdita di dati con **TimescaleDB**$_G$
6. Contrattazione su **use case**$_G$ suggeriti in precedenza
7. Tecnologie suggerite

•

Versione **1.0.0**

Stato	Verificato
Partecipanti	Alessandro Dinato Riccardo Graziani Elia Ernesto Stellin
Distribuzione	GlitchHub Team M31 SRL Prof. Cardin Riccardo Prof. Vardanega Tullio

Registro Modifiche

Ver.	Data	Autore	Verificatore	Descrizione
1.0.0	03/02/2026	Elia Ernesto Stellin	Alessandro Dinato	Creazione versione stabile del verbale esterno del 27/01/2026
0.1.1	03/02/2026	Elia Ernesto Stellin	Alessandro Dinato	Applicate correzioni relative a verifica della versione 0.1.0.
0.1.0	01/02/2026	Elia Ernesto Stellin	Alessandro Dinato	Stesura verbale esterno del 27/01/2026
0.0.1	01/02/2026	Elia Ernesto Stellin	Alessandro Dinato	Bozza iniziale verbale esterno del 27/01/2026

Indice

1. Introduzione

Il presente verbale attesta che in data 27 gennaio 2026 dalle 14:30 alle 15:45, si è svolto l'incontro con la proponente M31 SRL, in modalità remota.

L'incontro ha avuto l'obiettivo di presentare lo stato di avanzamento dei lavori e di discutere sulla struttura del PoC rilevata dal gruppo e su come soddisfare specifici requisiti del prodotto con le tecnologie rilevate.

2. Resoconto

Di seguito si riportano i punti salienti dell'incontro in oggetto, notando che ogni singolo argomento di discussione si rivelerà utile nella produzione del **PoC**$_G$ e dell'**MVP**$_G$ e, in generale, per una maggiore comprensione del capitolato.

2.1. Presentazione stato avanzamento lavori

Il gruppo ha raggiunto una versione stabile dei requisiti del prodotto, i quali sono conformi alle attese della proponente. Inoltre, il gruppo ha iniziato le attività di studio delle tecnologie da usare nel **PoC**$_G$, identificandone anche gli elementi architetturali principali.

2.2. Comprensione del simulatore di gateway$_G$

Il gruppo ha posto alla proponente una serie di domande concernenti il funzionamento specifico del simulatore di gateway. Le domande poste e le relative risposte ricevute dalla proponente sono riportate di seguito.

2.2.1. Implementazione risposta a comandi del cloud da parte del Gateway

Il **capitolato d'appalto**$_G$ pone come requisito funzionale obbligatorio (RQ 2.5) la possibilità che il **simulatore di gateway**$_G$ risponda a **comandi**$_G$ ricevuti dal **Cloud**$_G$.

Il gruppo ha rilevato due possibili soluzioni per soddisfare questo requisito, chiedendo alla proponente quale fosse preferibile. Le opzioni sono le seguenti:

1. Usare una soluzione *publisher/subscriber* (*Pub/Sub*) in cui il simulatore di gateway è al contempo *Publisher* dei dati IoT e *Subscriber* per i comandi ricevuti dal cloud. L'unico problema di ciò è il fatto che si fornisce al **simulatore di gateway**$_G$ la duplice responsabilità di inviare i dati e ricevere i comandi.
2. Usare una soluzione che sfrutta il **long polling**$_G$, ovvero in cui il **simulatore di gateway**$_G$ apre una connessione persistente con l'infrastruttura **Cloud**$_G$, chiedendo regolarmente a quest'ultima se siano presenti dei comandi eseguibili da parte del gateway.

La proponente ha dichiarato che la prima soluzione è altamente preferibile perché:

- Fornire al **simulatore di gateway**$_G$ la duplice responsabilità di inviare dati e ricevere comandi non è un problema, non essendoci altro modo di ovviare ciò.
- Usare il **long polling**$_G$ potrebbe essere problematico se l'infrastruttura **Cloud**$_G$ dovesse scalare orizzontalmente, in quanto la connessione persistente creata non sarebbe in grado di raggiungere tutti i nodi dell'infrastruttura **Cloud**$_G$ interessati nella comunicazione con il **gateway**$_G$.

2.2.2. Scelta del client usato dal Gateway

Il gruppo ha chiesto se fosse preferibile che, nella comunicazione con l'infrastruttura **Cloud**$_G$ tramite **NATS**$_G$, il gateway usasse il client nativo a tale tecnologia oppure un client **MQTT**$_G$ specifico.

La proponente ha dichiarato che è sufficiente utilizzare il client nativo di **NATS**$_G$ in quanto, nel contesto dell'**MVP**$_G$, il **simulatore di gateway**$_G$ non deve soddisfare particolari requisiti di performance, che si potrebbero soddisfare usando il client **MQTT**$_G$.

2.2.3. Utilizzo di *ingestion service*

Il gruppo ha chiesto se fosse necessario l'utilizzo di un *ingestion service* da porre come strato di protezione tra il **simulatore di gateway**$_G$ e il **Cloud**$_G$, con lo scopo di isolare totalmente **NATS**$_G$ dal resto dell'infrastruttura per:

- Eseguire autonomamente autenticazione, autorizzazione e validazione dei messaggi in ingresso;
- Rendere il **simulatore di gateway**$_G$ indipendente da **NATS**$_G$.

La proponente ha dichiarato che tale servizio non è necessario, poiché non si pongono requisiti di scalabilità stringenti sul software, in quanto prodotto di un **progetto didattico**$_G$.

2.3. Discussione e dubbi sulla struttura del PoC$_G$

Il gruppo ha delineato come elementi principali del **PoC**$_G$ i seguenti elementi:

1. Un servizio in **Go**$_G$ che esegue il *publishing* di dati su **NATS**$_G$, simulando la funzionalità di publishing del **gateway**$_G$
2. L'istanza **NATS**$_G$ che agisce da *message broker*
3. Delle istanze **Go**$_G$ che *consumano* i dati provenienti da **NATS**$_G$ e li salvano in un'istanza di **TimescaleDB**$_G$
4. Un servizio di **API**$_G$ REST che accede ai dati salvati in **TimescaleDB**$_G$.
5. Un web server che fornisce un'applicazione web scritta in **Angular.js**$_G$ che accede ai dati dei sensori

Il gruppo non era sicuro se utilizzare **Go**$_G$ o un linguaggio più strutturato come C# per i servizi di **API**$_G$ Rest (Punto 3) e di *data consuming* (Punto 4), poiché **Go**$_G$ non è considerato un linguaggio pienamente *object oriented* e manca di alcune funzionalità che facilitano lo sviluppo di applicazioni complesse.

La proponente, però, ha caldamente consigliato al gruppo di utilizzare **Go**$_G$ al posto di C# onde evitare di dover imparare tre linguaggi (C#, **Go**$_G$ e JavaScript) invece che solo due (**Go**$_G$ e JavaScript). Inoltre, sebbene **Go**$_G$ non sia un linguaggio tradizionalmente con molte librerie, utilizzarlo per la creazione di **API**$_G$ REST è ampiamente facilitato dall'esistenza di framework quali **Gin**$_G$ e **Fiber**$_G$.

2.4. Separazione di dati tra tenant

Il **capitolato d'appalto**$_G$ pone come requisito funzionale obbligatorio (RQ 3) la piena separazione dei dati tra **tenant**$_G$ diversi.

Il gruppo ha rilevato due soluzioni possibili per soddisfare questo requisito, non essendo sicuro su quale soluzione scegliere. Le opzioni sono le seguenti:

1. Usare una singola istanza di **TimescaleDB**$_G$ per salvare i dati di tutti i **tenant**$_G$, ma separarne l'accesso tramite l'utilizzo di **Schema**$_G$. Questa soluzione rende più immediato lo sviluppo dell'infrastruttura del sistema di persistenza, ma al contempo rendendolo anche meno sicuro da attacchi esterni.
2. Creare un'istanza separata per ogni singolo **tenant**$_G$, rendendo l'accesso ai dati più sicuro, ma rendendo il sistema più complicato da creare e mantenere, in quanto sarebbe richiesto anche l'utilizzo di un *resolver* e un *secrets manager* per associare a ogni **tenant**$_G$ l'indirizzo dell'istanza del database che ne contiene i dati e le credenziali per accedervi.

La proponente ha dichiarato che per quanto concerne l'**MVP**$_G$ la prima soluzione, per quanto parziale, è sufficiente, ma informandoci che in un progetto reale la separazione tra i dati dei **tenant**$_G$ è un requisito più stringente.

2.5. Gestione perdita di dati con TimescaleDB$_G$

Il gruppo ha deciso di utilizzare **TimescaleDB**$_G$ per la persistenza dei dati ottenuti dal servizio di *data consuming*, data la sua performance nell'inserimento di massa (*bulk insert*) di dati associati a una scala temporale.

Il gruppo ha, però, notato che per sfruttare la funzionalità di *bulk insert* di **TimescaleDB**$_G$ è necessario che l'istanza **Go**$_G$ di *data consuming* salvi i dati raccolti in memoria volatile prima di effettuare l'operazione sul database, comportando però un rischio di perdita di dati in caso l'istanza **Go**$_G$ riscontri un'avaria e si spenga. La proponente ha specificato che la perdita di dati è inammissibile e che il gruppo deve definire meglio questi dettagli dopo uno studio più approfondito delle tecnologie scelte.

Perciò, il gruppo ha deciso di cercare un'alternativa che possa comunque sfruttare la funzionalità di *bulk insert* garantendo al contempo che nessun dato venga mai perso durante questa procedura.

2.6. Contrattazione su use case$_G$ suggeriti in precedenza

Data la vicinanza della scadenza della revisione dell'**RTB**$_G$, il gruppo ha chiesto alla proponente se fosse necessario includere nel documento di **Analisi dei Requisiti**$_G$ i seguenti **use case**$_G$ opzionali identificati in incontri precedenti:

- Funzionalità di log di sistema che tiene traccia di tutti i **comandi**$_G$ inviati a uno specifico **gateway**$_G$
- Funzionalità di invio email in caso di alert;
- Visualizzazione di costo monetario speso nell'utilizzo di una specifica **API Key**$_G$
- Visualizzazione di mappa nella dashboard che mostra le posizioni geografiche dei **gateway**$_G$ associati al **tenant**$_G$.

Il gruppo, infatti, ritiene che questi **casi d'uso**$_G$ siano estremamente secondari e ininfluenti sul progetto finale e aggiungerli all'**Analisi dei Requisiti**$_G$ potrebbe causare ritardi non previsti.

La proponente ha dichiarato che i casi d'uso sopra specificati non sono importanti in vista dell'**MVP**$_G$.

2.7. Tecnologie suggerite

In vista delle discussioni precedentemente riportate, **M31 Srl** ha consigliato l'approfondimento di **Gin**$_G$ per la creazione del servizio di **API**$_G$ REST.

3. Attività conseguenti

Task	Assegnatari	Issue
Sviluppo parte **NATS**$_G$	Alessandro Dinato	**PoC/#1**
Sviluppo collegamento servizio *data consumer* - **TimescaleDB**$_G$	Alessandro Dinato	**PoC/#4**
Studio framework **Gin**$_G$	Michele Dioli, Jaume Bernardi, Elia Ernesto Stellin	**PoC/#6**
Sviluppo API REST con **Gin**$_G$	Michele Dioli, Jaume Bernardi, Elia Ernesto Stellin	**PoC/#7**
Sviluppo dashboard **Angular.js**$_G$	Siria Salvalaio, Riccardo Graziani	**PoC/#11**

Alessandro Dinato

Firma del revisore interno

Cristian Pirlog

Firma del revisore esterno