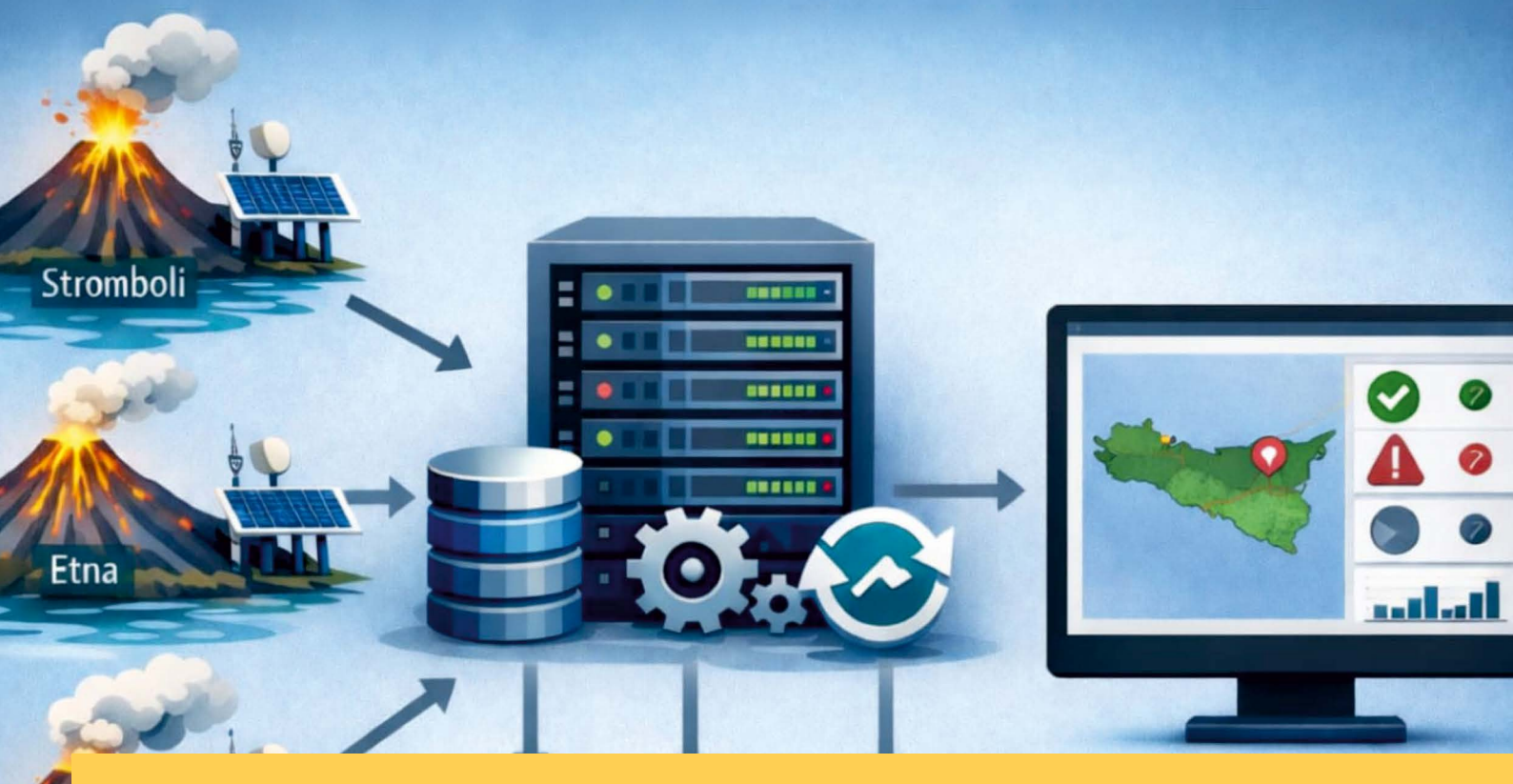




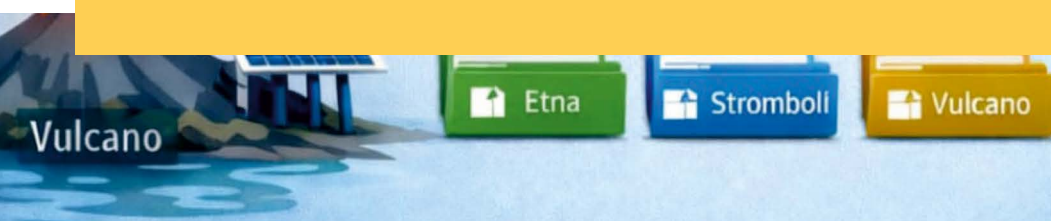
# RAPPORTI TECNICI INGV

Implementazione di un sistema  
centralizzato per il trasferimento dei dati  
e il monitoraggio delle stazioni  
FLAME



ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA

520



**Direttore Responsabile**

Daniela VERSACE

**Editor in Chief**

Milena MORETTI

**Editorial Board**

Laura ALFONSI

Sofia BARANELLO

Carla BARNABA

Christian BIGNAMI

Simona CARANNANTE

Maria Grazia CIACCIO

Luca COCCHI

Luigi CUCCI

Lorenzo CUGLIARI

Elisabetta D'ANASTASIO

Alessia DI CAPRIO

Roberto DI MARTINO

Domenico DI MAURO

Domenico DORONZO

Filippo GRECO

Alessandro IAROCCI

Mario MATTIA

Daniele MELINI

Anna NARDI

Lucia NARDONE

Marco OLIVIERI

Arianna PESCI

Pierangelo ROMANO

Paola SBARRA

Davide SCAFIDI

Maurizio SOLDANI

Sara STOPPONI

Umberto TAMMARO

David ZULIANI

**REGISTRAZIONE AL TRIBUNALE DI ROMA N.173 | 2014, 23 LUGLIO**

© INGV Istituto Nazionale di Geofisica e Vulcanologia

Rappresentante legale: Presidente INGV

Sede: Via di Vigna Murata, 605 | Roma

Tutti i diritti riservati

[www.ingv.it](http://www.ingv.it)



**ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA**

# RAPPORTI TECNICI INGV

Implementazione di un sistema centralizzato  
per il trasferimento dei dati e il monitoraggio  
delle stazioni FLAME

*Implementation of a centralized system for data  
transfer and monitoring of FLAME stations*

Carmelo La Piana\*, Gaetano Mazzaglia, Paolo Principato

INGV | Istituto Nazionale di Geofisica e Vulcanologia, Osservatorio Etneo

\*Corresponding author

Accettato 22 marzo 2026 | Accepted 22 March 2026

Come citare | How to cite La Piana, C., Mazzaglia, G., Principato, P. (2026). Implementazione di un sistema centralizzato per il trasferimento dei dati e il monitoraggio delle stazioni FLAME. *Rapporti Tecnici INGV*, 520, 1-36. <https://doi.org/10.13127/rpt/520>

In copertina Immagine generata con Google Gemini su prompt dell'autore (19/01/2026) | Cover Image generated with Google Gemini at the author's prompt (19/01/2026)

520



# INDICE

|                                                                                             |           |
|---------------------------------------------------------------------------------------------|-----------|
| <b>Riassunto</b>                                                                            | <b>7</b>  |
| <b><i>Abstract</i></b>                                                                      | <b>7</b>  |
| <b>Introduzione</b>                                                                         | <b>8</b>  |
| <b>1. Architettura del sistema</b>                                                          | <b>8</b>  |
| 1.1 Componente stazione ( <i>server</i> di invio)                                           | 9         |
| 1.2 Componente centrale ( <i>server</i> di acquisizione)                                    | 9         |
| <b>2. Implementazione dell'interfaccia utente</b>                                           | <b>10</b> |
| 2.1 Tecnologie utilizzate                                                                   | 10        |
| 2.2 Dashboard principale                                                                    | 10        |
| <b>3. Requisiti di sistema e installazione</b>                                              | <b>11</b> |
| <b>4. Configurazione dell'ambiente di produzione</b>                                        | <b>12</b> |
| 4.1 Configurazione centralizzata ( <i>config.ini</i> )                                      | 12        |
| 4.2 Architettura di avvio e <i>scripting</i> di gestione (utilità di pianificazione)        | 12        |
| 4.3 Configurazione del <i>firewall</i>                                                      | 13        |
| <b>5. Dettagli implementativi</b>                                                           | <b>13</b> |
| 5.1 Architettura di comunicazione: <i>API RESTful</i>                                       | 13        |
| 5.2 Garanzia di integrità dei dati (processo di spostamento sicuro)                         | 14        |
| 5.3 Logica di sato basata su contatore di errori                                            | 14        |
| 5.4 Archiviazione dinamica e prova di duplicati                                             | 15        |
| 5.5 Gestione della cancellazione (modalità <i>test</i> /produzione)                         | 15        |
| 5.6 Gestione della concorrenza del <i>database</i> (prevenzione <i>database is locked</i> ) | 15        |
| 5.7 Aggiornamento di stato non bloccante ( <i>liveness update</i> )                         | 16        |
| 5.8 Gestione dei processi in <i>background</i> ( <i>PID files</i> )                         | 16        |
| <b>6. Validazione del <i>software</i></b>                                                   | <b>16</b> |
| 6.1 Metriche tecniche e <i>stress test</i>                                                  | 17        |
| <b>7. Limiti e sviluppi futuri</b>                                                          | <b>17</b> |
| <b>8. Conclusioni</b>                                                                       | <b>18</b> |
| <b>9. Disponibilità del codice sorgente</b>                                                 | <b>18</b> |
| <b>Ringraziamenti</b>                                                                       | <b>18</b> |
| <b>Bibliografia</b>                                                                         | <b>19</b> |

|                    |           |
|--------------------|-----------|
| <b>Sitografia</b>  | <b>19</b> |
| <b>Appendice A</b> | <b>21</b> |
| <b>Appendice B</b> | <b>25</b> |

## Riassunto

Il presente rapporto tecnico descrive l'architettura e l'implementazione di una piattaforma *software*, progettata per gestire il trasferimento sicuro e il monitoraggio in tempo reale di *file* provenienti dalle stazioni remote FLAME (*FLux Automatic MEasurements*) verso un *server* di acquisizione centrale. La necessità di un sistema unificato nasce dall'esigenza di automatizzare il processo di acquisizione dei dati, garantire l'integrità dei *file* durante il trasferimento (tramite *hash*), e fornire al personale un'interfaccia unica (*dashboard*) per il controllo e la configurazione della rete.

La soluzione implementata è un'applicazione basata su *Python*, con un'interfaccia *web* (*dashboard*) sviluppata in *Flask* e un *server* leggero *FastAPI* per le stazioni remote. Il sistema è supportato da un *database SQLite* per la registrazione e lo storico. Un servizio in *background* gestisce l'acquisizione parallela e continua dei dati, garantendo l'aggiornamento costante dello stato delle stazioni.

Le funzionalità chiave includono:

1. Una ***dashboard "live"*** con indicatori *KPI* (*Key Performance Indicator*) per lo stato della rete (*Online*, *Errore*, *Offline*) e contatori di trasferimento giornalieri.
2. Un sistema di trasferimento dati sicuro basato su verifica *hash* (*SHA-256*) per garantire l'integrità dei *file*.
3. Una ***logica di stato robusta*** che rileva automaticamente le stazioni irraggiungibili (tramite un contatore di errori consecutivi) e le segnala come "Errore" o "Offline".
4. Un'***architettura di archiviazione automatica*** che organizza i *file* ricevuti in base alla località (*Etna*, *Stromboli*, *Vulcano*) e alla data di origine del *file* (formato *AAMMGG*).
5. Un sistema di ***configurazione dinamica*** basato su un *file config.ini* centralizzato e su un *file stations.json* gestibile via interfaccia.

## Abstract

This technical report details the architecture and implementation of a centralized client-server platform, "File Transfer," designed for the secure transfer and real-time monitoring of data files from remote stations. The need for a unified system arises from the requirement to automate the data acquisition pipeline, guarantee file integrity during transfer (via hash verification), and provide personnel with a single interface (*dashboard*) for network health monitoring and configuration.

The implemented solution is a *Python*-based application, featuring a central *Flask* web *dashboard* and a lightweight *FastAPI* server on each remote station. The system utilizes a *SQLite* database with monthly rotation for persistent logging and status tracking. The system utilizes a central background service that employs a multi-threaded *ThreadPoolExecutor* for parallel and continuous data acquisition from all configured stations.

Key features include:

1. A "live" *dashboard* with *KPIs* for network health (*Online*, *Error*, *Offline*) and daily transfer statistics.
2. A secure transfer protocol using *SHA-256* hash verification to ensure zero data corruption.
3. An automatic anomaly detection system based on a consecutive error counter to identify and flag unresponsive stations.
4. A dynamic storage architecture that sorts received files based on pre-configured location tags (*Etna*, *Stromboli*, *Vulcano*) and the file's original date (format *yymmdd*).
5. A flexible configuration system, including a per-station toggle (*checkbox*) to switch between "Copy" (test) and "Move" (production) modes.

# Introduzione

La gestione dei *file* provenienti da stazioni remote dislocate sul territorio comporta sfide significative, legate alla necessità di garantire trasferimenti rapidi, integri e resilienti ai guasti di rete. In assenza di un sistema automatizzato, il processo richiedeva controlli manuali, era soggetto a errori di corruzione e non offriva alcuna visione d'insieme sullo stato delle connessioni. L'obiettivo principale di questo progetto è stato sviluppare una piattaforma *software* unica, capace di fungere sia da sistema di trasporto dati affidabile e sia da controllo per le connessioni verso le stazioni remote. L'applicazione è stata progettata per:

- Contattare più stazioni in parallelo, ottimizzando i tempi di acquisizione.
- Trasferire i *file* solo dopo averne verificato l'integrità, garantendo la correttezza dei dati, con la possibilità di cancellarli solo al completamento del trasferimento, evitando così dati ridondanti e il riempimento dell'*HDD* nella stazione remote.
- Gestire in modo intelligente le interruzioni di rete, riprendendo il lavoro senza perdita di dati o duplicazioni.
- Organizzare i *file* ricevuti in una struttura di cartelle predefinita, basata su località e data.
- Fornire un'interfaccia *web* intuitiva, per monitorare lo stato della rete delle stazioni e configurare il sistema.

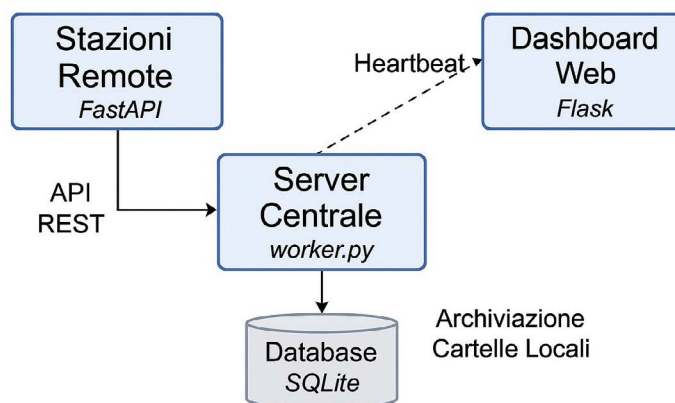
## 1. Architettura del sistema

Il sistema è concepito con un'architettura *client-server* distribuita, focalizzata su affidabilità, scalabilità e facilità di manutenzione. Si articola in tre componenti chiave (come illustrato in Figura 1):

1. **Componente stazione:** un *server* leggero installato su ciascuna stazione remota FLAME, responsabile dell'invio sicuro dei dati.
2. **Componente centrale:** il fulcro del sistema, che gestisce l'acquisizione e l'organizzazione dei *file*, integrando logiche di resilienza e parallelismo.
3. **Dashboard web:** l'interfaccia di monitoraggio e controllo, accessibile direttamente tramite *browser*.

Questa struttura modulare garantisce un'efficiente gestione di grandi volumi di dati e permette l'integrazione di nuove stazioni senza richiedere modifiche architetturali significative.

Figura 1 Schema dell'architettura del sistema.  
Figure 1 System architecture diagram.



## 1.1 Componente stazione (server di invio)

Basato su **FastAPI** [4] [5], questo *script* leggero, in esecuzione su ogni stazione remota, assicura le seguenti funzionalità essenziali:

- **Esposizione di endpoint API:** mette a disposizione del *worker* centrale gli *endpoint* necessari per la gestione dei *file*: */files*, */file-hash*, */download*, */delete* e */bulk-delete-folders*.
- **Sicurezza e autenticazione:** tutte le richieste sono protette e autenticate tramite una chiave API segreta (X-API-Key), configurabile nel *file config.ini* locale.
- **Filtro intelligente dei dati:** garantisce l'invio esclusivo dei dati utili, escludendo automaticamente *file* di sistema, *log* e *file* temporanei (es. *.temp*, *.tmp*, *.log*).
- **Efficienza e leggerezza:** è progettato per operare con un impatto minimo sulle risorse della stazione remota.

## 1.2 Componente centrale (server di acquisizione)

Il cuore del sistema risiede sul PC centrale e comprende diverse logiche avanzate:

- **Servizio in background (worker.py):** *script* Python [1] progettato per funzionare 24/7, gestendo il ciclo di acquisizione.
- **Parallelismo (ThreadPoolExecutor):** gestione simultanea di più stazioni, con un *thread* dedicato per ciascuna.
- **Tolleranza ai guasti (requests.Retry):** ritenta automaticamente le connessioni fino a tre volte in caso di errori temporanei.
- **Heartbeat attivo:** aggiorna il *file* di stato (*status.json*) ogni 30 secondi per evitare falsi allarmi di blocco.
- **Logica di stato persistente:** aggiorna nel *database* lo stato delle stazioni (Online, Errore) e i contatori di errori.

### Database (SQLite)

- **Scelta tecnologica:** SQLite [6] per semplicità e portabilità.
- **Rotazione giornaliera:** creazione automatica di un nuovo *file* di *database* ogni giorno (es. *history\_2025\_10\_12.db*) per evitare crescita incontrollata.
- **Gestione concorrenza:** *timeout* di dieci secondi per prevenire errori di tipo “*database is locked*”.
- **Tabelle chiave:**
  - *logs*: storico delle operazioni;
  - *station\_status*: stato attuale e contatori per ogni stazione;
  - *folders\_to\_delete*: gestione della cancellazione ritardata delle cartelle sorgente.

### Interfaccia utente (dashboard – app.py)

- **Tecnologia:** applicazione web basata su Flask [2] e Waitress [3], con *frontend responsive* in Bootstrap 5 [7].
- **Funzionalità:**
  - Visione d'insieme dello stato del servizio e delle stazioni.
  - Configurazione (aggiunta stazioni, impostazione località, attivazione cancellazione).
  - Controllo del servizio “*worker*” (Avvia, Stop, Riavvia).
  - Visualizzazione *log* e storico in schede dedicate.

## 2. Implementazione dell'interfaccia utente

L'interfaccia utente è una componente chiave per il controllo e il monitoraggio del sistema, progettata per essere intuitiva, *responsive* e sicura. È stata realizzata come *Single Page Application* (SPA) utilizzando il *micro-framework* *Flask* (con *Waitress* come *server* di produzione) e *Bootstrap 5* per garantire un *design* moderno e adattabile a *PC*, *tablet* e *smartphone*.

La *dashboard* fornisce:

- **Aggiornamento in tempo reale** dello stato delle stazioni tramite *polling* asincrono (*AJAX - Asynchronous JavaScript and XML*).
- **Gestione interattiva** delle modalità operative (*test*/produzione) con *checkbox* e pulsanti dinamici.
- **Visualizzazione dei log e degli errori**.

### 2.1 Tecnologie utilizzate

- **Backend:** il *server web* è implementato in *app.py* utilizzando il *micro-framework* *Flask*, scelto per la sua leggerezza, semplicità e flessibilità nello sviluppo di *API* e *interfacce web*.
- **Server di produzione:** l'applicazione *Flask* è eseguita tramite *Waitress*, un *server WSGI* (*Web Server Gateway Interface*) *production-ready* scritto in puro *Python*, che garantisce stabilità, sicurezza e buone prestazioni in ambienti reali.
- **Frontend:** l'interfaccia utente è sviluppata con *Bootstrap 5*, adottando un approccio *mobile-first* per garantire un *design* moderno, *responsive* e compatibile con *PC*, *tablet* e *smartphone*.
- **Comunicazione dati:** la *dashboard* utilizza *JavaScript* (*API fetch*) per effettuare richieste asincrone all'endpoint/*dashboard-data* del *server* ogni cinque secondi (*polling*). Il *server* risponde con un oggetto *JSON* (*JavaScript Object Notation*) contenente lo stato completo del sistema, che viene elaborato dal *JavaScript* per aggiornare dinamicamente la pagina senza ricaricarla, assicurando un'esperienza fluida e in tempo reale.

### 2.2 Dashboard principale

La *dashboard* è divisa in quattro aree logiche:

1. **Panoramica generale:** mostra i 4 indicatori *KPI* principali (*File Trasferiti Oggi*, *Dati Trasferiti Oggi*, *Errori Oggi*, *Stato del Servizio*) (Figura 2).



Figura 2 Dashboard principale dell'applicazione (KPI).

Figure 2 Main application dashboard (KPI).

2. **Stato individuale stazioni:** una tabella raggruppata per schede (*Tutte*, *Etna*, *Stromboli*, *Vulcano*) che mostra i dettagli di ogni stazione, inclusa la *Località*, lo *Stato* (*Online*, *Errore*, *Offline*) e il numero di *file* trasferiti (vedi Figura 3).

| Stato Individuale Stazioni                                               |           |                     |        |                 |
|--------------------------------------------------------------------------|-----------|---------------------|--------|-----------------|
| <div>Tutte</div> <div>Etna</div> <div>Stromboli</div> <div>Vulcano</div> |           |                     |        |                 |
| Stazione                                                                 | Località  | Ultimo Contatto     | Stato  | File Trasferiti |
| Flame_Test                                                               | Etna      | 2025-11-14 07:18:53 | Online | 0               |
| ELIN                                                                     | Etna      | 2025-11-14 07:18:56 | Online | 80              |
| ELI                                                                      | Etna      | 2025-11-14 07:19:08 | Online | 77              |
| SPL1                                                                     | Stromboli | 2025-11-14 07:19:06 | Online | 79              |
| EMIL                                                                     | Etna      | 2025-11-14 07:20:24 | Errore | 20              |
| ECUR                                                                     | Etna      | 2025-11-14 07:18:56 | Online | 88              |
| EBRO                                                                     | Etna      | 2025-11-14 07:18:54 | Online | 88              |

**Figura 3** Dettaglio stato stazioni in tempo reale.

**Figure 3** Real-time station status detail.

3. **Configurazione e controlli:** il “pannello di controllo” che permette agli operatori di (Figura 4):

- Controllare il servizio *worker* (AVVIA, STOP, RIAVVIA).
- Aggiungere, modificare o rimuovere stazioni.
- Impostare la Località di archiviazione per ogni stazione FLAME (Etna, Stromboli, Vulcano).
- Impostare la modalità di funzionamento (test o produzione) tramite il *checkbox* "Cancella?".

### Configurazione e Controlli

Controlli Servizio

▶ AVVIA

■ STOP

🔄 RIANVIA

### Configurazione Stazioni

| Nome Stazione | URL                               | Località                 | Cancella?                           | Azione |
|---------------|-----------------------------------|--------------------------|-------------------------------------|--------|
| Flame_Test    | http://testflame.dyn.invg.it:5000 | Ètna <span>▼</span>      | <input checked="" type="checkbox"/> |        |
| ELIN          | http://elinflame.dyn.invg.it:5000 | Ètna <span>▼</span>      | <input type="checkbox"/>            |        |
| ELI           | http://eliflame.dyn.invg.it:5000  | Ètna <span>▼</span>      | <input type="checkbox"/>            |        |
| SPL1          | http://spl1flame.dyn.invg.it:5000 | Stromboli <span>▼</span> | <input type="checkbox"/>            |        |
| EMIL          | http://emilflame.dyn.invg.it:5000 | Ètna <span>▼</span>      | <input type="checkbox"/>            |        |
| ECUR          | http://ecurflame.dyn.invg.it:5000 | Ètna <span>▼</span>      | <input type="checkbox"/>            |        |
| EBRO          | http://ebroflame.dyn.invg.it:5000 | Ètna <span>▼</span>      | <input type="checkbox"/>            |        |

➡ Aggiungi Stazione

💾 Salva Configurazione

**Figura 4** Sezione controlli e configurazione nuova stazione.

**Figure 4** Controls and configuration section of the new station.

4. **Storico recente:** un'area a schede che mostra il “Log Completo”, gli “Ultimi Trasferimenti” e gli “Ultimi Errori” per una diagnosi rapida, ed in tempo reale (Figura 5).

```
Log Completo      Ultimi Trasferimenti      Ultimi Errori
```

---

|                     |      |                                                                                           |
|---------------------|------|-------------------------------------------------------------------------------------------|
| 2025-11-18 14:11:04 | TWFO | (E1LM) [251872_071013_eain_v_s_V3_R0ck2162.net] File già presente a destinazione, salto.  |
| 2025-11-18 14:11:04 | TWFO | (E1LM) [251872_071013_eain_v_s_V3_R0ck2162.net] File già presente a destinazione, salto.  |
| 2025-11-18 14:11:19 | TWFO | (E1LM) [251872_070918_eain_v_s_V3_R0ck2162.net] File già presente a destinazione, salto.  |
| 2025-11-18 14:11:43 | TWFO | (E1CM) [251809_1340131_ccur_v_s_V3_R0ck1316.net] File già presente a destinazione, salto. |
| 2025-11-18 14:11:43 | TWFO | (E1CM) [251809_1340131_ccur_v_s_V3_R0ck1316.net] File già presente a destinazione, salto. |
| 2025-11-18 14:11:43 | TWFO | (E1CM) [251809_1340131_ccur_v_s_V3_R0ck1316.net] File già presente a destinazione, salto. |
| 2025-11-18 14:11:43 | TWFO | (E1CM) [251809_1340131_ccur_v_s_V3_R0ck1316.net] File già presente a destinazione, salto. |
| 2025-11-18 14:11:43 | TWFO | (E1CM) [251809_134134_eain_v_s_V3_R0ck1316.net] File già presente a destinazione, salto.  |
| 2025-11-18 14:11:33 | TWFO | (E1CM) [251809_134134_eain_v_s_V3_R0ck1316.net] File già presente a destinazione, salto.  |
| 2025-11-18 14:11:33 | TWFO | (E1CM) [251809_134134_eain_v_s_V3_R0ck1316.net] File già presente a destinazione, salto.  |
| 2025-11-18 14:11:33 | TWFO | (E1CM) [251809_134134_eain_v_s_V3_R0ck1316.net] File già presente a destinazione, salto.  |
| 2025-11-18 14:11:43 | TWFO | (E1BM) [251807_133209_eain_v_s_V3_R0ck4341.txt] File già presente a destinazione, salto.  |
| 2025-11-18 14:11:43 | TWFO | (E1BM) [251807_133209_eain_v_s_V3_R0ck4341.txt] File già presente a destinazione, salto.  |
| 2025-11-18 14:11:43 | TWFO | (E1BM) [251807_133209_eain_v_s_V3_R0ck4341.txt] File già presente a destinazione, salto.  |
| 2025-11-18 14:11:43 | TWFO | (E1BM) [251807_133209_eain_v_s_V3_R0ck4341.txt] File già presente a destinazione, salto.  |
| 2025-11-18 14:11:43 | TWFO | (E1BM) [251807_133209_eain_v_s_V3_R0ck4341.txt] File già presente a destinazione, salto.  |
| 2025-11-18 14:11:43 | TWFO | (E1BM) [251807_133209_eain_v_s_V3_R0ck4341.txt] File già presente a destinazione, salto.  |
| 2025-11-18 14:11:43 | TWFO | (E1BM) [251807_133209_eain_v_s_V3_R0ck4341.txt] File già presente a destinazione, salto.  |
| 2025-11-18 14:11:43 | TWFO | (E1BM) [251807_133209_eain_v_s_V3_R0ck4341.txt] File già presente a destinazione, salto.  |
| 2025-11-18 14:11:39 | TWFO | (G1L1) [251807_133308_eain_v_s_V3_R0ck4391.net] File già presente a destinazione, salto.  |
| 2025-11-18 14:11:39 | TWFO | (G1L1) [251807_133308_eain_v_s_V3_R0ck4391.net] File già presente a destinazione, salto.  |

**Figura 5** Sezione *log* completo /Ultimi trasferimenti/Ultimi errori.

**Figure 5** Full log section /Latest transfers /Latest errors.

### 3. Requisiti di sistema e installazione

Per garantire la riproducibilità dell'ambiente di esecuzione, sono stati definiti i seguenti requisiti minimi e dipendenze *software*:

- Interprete: *Python 3.9* o superiore.
- Sistema Operativo: *Windows Server e/o Windows 10/11, SO Linux.*

Dipendenze (*Python Libraries*): le librerie necessarie sono elencate nel *file* `standard requirements.txt` e includono principalmente:

- *flask*: per il *server web* e l'interfaccia di gestione;
- *requests*: per le chiamate *API* verso le stazioni;
- *sqlite3*: (incluso nella *standard library*) per la persistenza dei dati;

**Procedura di installazione:** il *deployment* prevede la clonazione della *repository* e l'installazione delle dipendenze in un ambiente virtuale isolato (*venv*) per evitare conflitti con le librerie di sistema:

```
python -m venv venv
venv\Scripts\activate
pip install -r requirements.txt
```

## 4. Configurazione dell'ambiente di produzione

Questa sezione descrive le scelte tecniche adottate per garantire **robustezza, sicurezza e manutenibilità** del sistema in ambiente operativo. L'obiettivo è assicurare che i servizi funzionino in modo continuo, sicuro e scalabile, riducendo al minimo gli interventi manuali. Le configurazioni sono esternalizzate per semplificare la gestione, mentre l'avvio e il monitoraggio dei processi sono automatizzati per garantire disponibilità 24/7. Inoltre, vengono definite le regole di rete e di sicurezza necessarie per la comunicazione tra i componenti.

### 4.1 Configurazione centralizzata (`config.ini`)

Per garantire la flessibilità e ridurre al minimo le modifiche al codice sorgente, tutte le impostazioni operative sono state esternalizzate in un *file* di configurazione `config.ini` (vedi Appendice A). Questo approccio consente di aggiornare i parametri senza intervenire sul codice, migliorando la manutenibilità del sistema. Il *file* utilizza la libreria `configparser` di *Python* e definisce le seguenti sezioni principali:

- *[General]*: impostazioni globali, come `API_KEY` e `PROJECT_PATH`.
- *[Worker]*: parametri specifici per il *PC* centrale, ad esempio `BASE_DESTINATION_PATH` e `MAX_WORKERS`.
- *[WebApp]*: configurazione dell'interfaccia *web*, come la porta di ascolto (`PORT = 8080`).
- *[StationServer]*: impostazioni per le stazioni remote, tra cui `SOURCE_DATA_PATH` e la porta di comunicazione (`PORT = 5000`).

### 4.2 Architettura di avvio e *scripting* di gestione (utilità di pianificazione)

Per garantire l'operatività continua del sistema e una gestione affidabile dei processi, l'esecuzione dei moduli *Python* (`app.py` e `worker.py`) non avviene direttamente, ma è orchestrata tramite un set di *script batch* e *wrapper* dedicati:

- `start_worker.bat` / `start_all.bat`: *script* responsabili dell'inizializzazione dell'ambiente,

dell'attivazione del *virtual environment* e del lancio effettivo dei processi *Python*;

- *start\_silenzioso.vbs*: un *wrapper* *VBScript* utilizzato per invocare i *file batch* in modalità invisibile, evitando la comparsa di finestre di terminale persistenti;
- *stop\_all.bat*: *script* deputato alla chiusura controllata di tutti i servizi attivi.

**Pianificazione ed esecuzione:** l'automazione è gestita tramite l'utilità di pianificazione di *Windows*, è stata configurata un'attività pianificata che invoca lo *script* di avvio (*start\_silenzioso.vbs*) automaticamente all'accensione del sistema.

Questa configurazione garantisce che il *software* operi in *background* come un servizio *always-on*, monitorando continuamente lo stato delle stazioni senza necessità di interventi manuali e assicurando che i processi vengano eseguiti con i privilegi necessari.

### 4.3 Configurazione del *firewall*

Per consentire la comunicazione sicura tra i componenti del sistema, è necessario configurare regole specifiche sul *firewall*:

- **PC centrale:** creare una regola di connessione in entrata per il *protocollo TCP* sulla porta 8080, utilizzata dall'interfaccia *web* e dai servizi centrali.
- **stazioni remote con router LTE:** configurare nel *router* regole di *NAT (port forwarding)* per il *protocollo TCP*, in modo da inoltrare i pacchetti in arrivo su specifiche porte all'indirizzo *IP* interno della stazione remota. Per reti *LTE* pubbliche, dove l'*IP* può cambiare, si utilizza il servizio *DynDNS@INGV* per rendere la stazione sempre raggiungibile tramite un *hostname* statico, evitando problemi di accesso remoto dovuti alla variazione dell'*IP*.

## 5. Dettagli implementativi

Questa sezione descrive le scelte tecniche e le logiche di controllo adottate per garantire affidabilità, sicurezza, automazione e resilienza del sistema. Vengono illustrate le architetture di comunicazione, i protocolli di trasferimento sicuro, le strategie di gestione degli errori, la persistenza dello stato, la prevenzione di duplicati, la gestione della concorrenza sul *database* e le modalità operative (*test* e *produzione*). Inoltre, sono incluse le soluzioni per il monitoraggio continuo, la gestione dei processi in *background* e l'aggiornamento non bloccante dello stato, elementi fondamentali per assicurare stabilità e scalabilità.

### 5.1 Architettura di comunicazione: *API RESTful*

La comunicazione tra il *PC* centrale e le stazioni FLAME si basa su *API RESTful*. Il *station\_server.py* (basato su *FastAPI*, vedi appendice B) espone *endpoint* sicuri (es. */files*, */download*) che il *worker.py* (il *client*) interroga usando i metodi *GET* e *DELETE* di *HTTP*.

L'autenticazione è garantita da una chiave *API* statica inviata nell'*header X-API-Key* di ogni richiesta, garantendo che solo i componenti autorizzati del sistema possano comunicare tra loro.

**Cos'è un'API REST?** Un'API *REST (Representational State Transfer)* è uno stile architetturale per la comunicazione tra sistemi *software* sulla rete. Invece di protocolli complessi, utilizza i comandi *standard* del *web* (gli stessi che usa un normale *browser*):

- **GET:** per richiedere dati (es. la lista dei *file*).
- **POST:** per inviare dati (es. un comando).
- **DELETE:** per eliminare una risorsa (es. un *file* sulla stazione).

## 5.2. Garanzia di integrità dei dati (processo di spostamento sicuro)

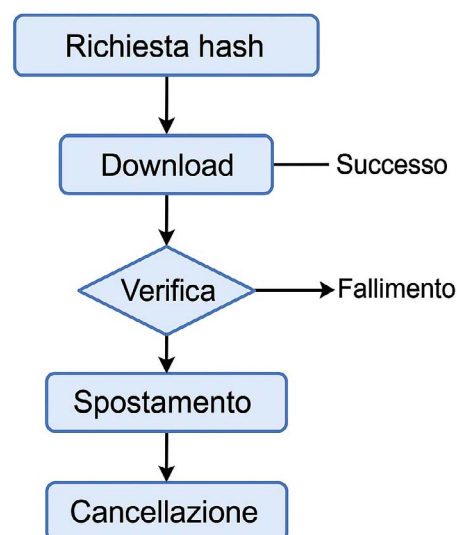
Il trasferimento dei *file* è progettato per garantire **integrità assoluta** e prevenire corruzioni o duplicazioni. Ogni operazione segue un protocollo rigoroso in cinque fasi (come illustrato in Figura 6):

1. **Calcolo dell'hash sorgente:** il *worker* richiede alla stazione l'hash SHA-256 del *file* originale.
2. **Download sicuro:** il *file* viene scaricato in una cartella temporanea (.tmp) per evitare modifiche premature.
3. **Verifica di integrità:** il *worker* calcola l'hash del *file* scaricato e lo confronta con quello fornito dalla stazione.
4. **Conferma e spostamento:** solo se gli hash corrispondono, il *file* temporaneo viene rinominato e spostato nella destinazione finale tramite *shutil.move*.
5. **Cancellazione controllata** (opzionale): se la modalità di cancellazione è attiva, il *worker* invia alla stazione la richiesta di eliminazione del *file* originale.

Questo processo garantisce che nessun *file* venga considerato valido senza verifica, riducendo il rischio di corruzione e assicurando la coerenza dei dati trasferiti.

Figura 6 Flusso operativo per il trasferimento sicuro dei *file*.

Figure 6 Operational flow for secure File Transfer.



## 5.3. Logica di stato basata su contatore di errori

Per garantire una rappresentazione stabile e affidabile dello stato della rete delle stazioni, il sistema utilizza una logica persistente che evita cambiamenti improvvisi. Gli stati sono gestiti nel *database* secondo queste regole:

- **Online:** impostato solo dopo una connessione riuscita, confermando la piena operatività della stazione.
- **Errore:** assegnato in caso di fallimento nella comunicazione o nel trasferimento. Questo stato rimane attivo finché non viene stabilita una nuova connessione valida.
- **Offline:** la *dashboard* applica una regola di visualizzazione basata sul contatore di errori consecutivi: se una stazione accumula **cinque o più errori di seguito**, viene mostrata come *Offline* (in rosso), segnalando un problema persistente.

Questa strategia assicura che lo stato visualizzato rifletta la reale condizione della stazione, evitando falsi positivi dovuti a errori temporanei.

## 5.4 Archiviazione dinamica e prova di duplicati

Il componente *worker.py* (vedi appendice B) è responsabile dell'organizzazione automatica dei dati ricevuti dalle stazioni, seguendo una logica strutturata che garantisce ordine e coerenza. Il processo si articola in quattro fasi principali:

1. **Estrazione della data:** analizza il percorso del *file* sorgente (es. *251020TEST/file.txt*) per ricavare la data (es. *251020*).
2. **Lettura della configurazione:** recupera la località associata alla stazione (es. Etna) dal *file stations.json*.
3. **Costruzione del percorso finale:** combina le informazioni per generare la struttura definitiva, ad esempio: *E:\UVDATA\Etna\NomeStazione\2025\10\251020*
4. **Verifica duplicati:** prima di spostare il *file*, il sistema controlla se esiste già nella destinazione (*os.path.exists*). In caso positivo, il *file* viene ignorato per evitare copie ridondanti e contatori errati.

Questa logica garantisce archiviazione ordinata, eliminazione dei duplicati e integrità dei dati, riducendo al minimo gli interventi manuali.

## 5.5 Gestione della cancellazione (modalità *test*/produzione)

Il sistema supporta due modalità operative, configurabili per ogni stazione tramite un semplice *checkbox* nell'interfaccia *web*:

- **Modalità *test*** (*checkbox* non selezionato): il *worker* copia i *file* nella destinazione, evitando duplicati, ma non invia mai richieste di cancellazione alla stazione. Questa modalità è pensata per verifiche e simulazioni senza rischi di perdita dati.
- **Modalità *produzione*** (*checkbox* selezionato): il *worker* sposta i *file* nella destinazione e pianifica la cancellazione della cartella sorgente per il giorno successivo, garantendo che le cartelle attive non vengano mai eliminate prematuramente.

Questa logica consente di passare facilmente da un ambiente di *test* sicuro a un flusso operativo completo, mantenendo il controllo sulla gestione dei dati.

## 5.6. Gestione della concorrenza del *database* (prevenzione *database is locked*)

Il sistema utilizza *SQLite*, un *database* leggero e portatile, che tuttavia può andare incontro a blocchi (*database is locked*) quando più processi tentano di accedere contemporaneamente in scrittura. Questo rischio è concreto in un'architettura *multi-thread* (il *worker.py* con i suoi *thread* paralleli) e multi-processo (il *worker.py* che scrive mentre *app.py* legge).

- **Il problema:** un accesso simultaneo non gestito può causare il *crash* del *worker.py* con errore *database is locked*.
- **La soluzione:** ogni connessione al *database* è stata configurata con il parametro *timeout=10*. In pratica, se il *database* risulta temporaneamente bloccato, *Python* non genera un errore immediato, ma attende fino a dieci secondi prima di ritentare. Questo approccio elimina conflitti di concorrenza e garantisce la stabilità del servizio anche in scenari ad alta attività.

Questa strategia, semplice ma efficace, consente di mantenere il sistema **robusto e resiliente**, evitando interruzioni dovute a operazioni concorrenti.

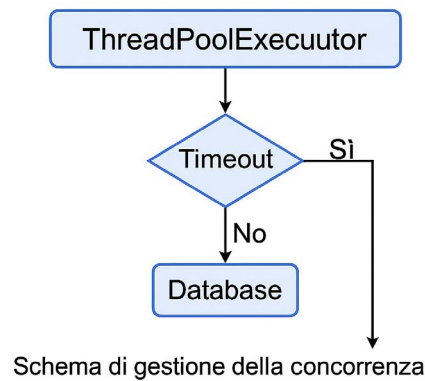


**Figura 7** Schema di gestione della concorrenza che illustra come il sistema evita l'errore “database is locked”.  
*Figure 7* Concurrency management diagram illustrating how the system avoids the “database is locked” error.

## 5.7 Aggiornamento di stato non bloccante (*liveness update*)

Un problema comune nei servizi in *background* è il rischio di apparire “congelati” durante operazioni lunghe, come il trasferimento di *file* di grandi dimensioni. Questo può generare falsi allarmi nella *dashboard*.

- **Il Problema:** se il *worker* è impegnato in un ciclo di copia che dura più di tre minuti, la *dashboard* non riceve aggiornamenti del *file* di stato (*status.json*) e segnala erroneamente: “Nessun battito recente. Bloccato?”.
- **La Soluzione:** è stata implementata una logica di aggiornamento attivo. Il ciclo di lavoro parallelo, gestito tramite *ThreadPoolExecuutor*, è stato modificato per utilizzare *wait (futures, timeout=30)*. In questo modo, il *worker* invia un aggiornamento di stato (“In elaborazione...”) ogni 30 secondi, anche durante operazioni prolungate. Questo garantisce che la *dashboard* rifletta sempre lo stato reale del servizio, evitando segnalazioni errate di blocco.

## 5.8. Gestione dei processi in *background* (*PID files*)

Come introdotto nella sezione 4.2, la gestione dei servizi è affidata agli *script batch*. Per garantire che l'avvio e l'arresto siano operazioni sicure e prive di conflitti, è stato implementato un meccanismo di controllo basato su *PID files* (*process ID*).

- **Prevenzione duplicati (Avvio):** quando *start\_worker.bat* viene eseguito, lo script *Python* verifica e scrive il proprio *ID* di processo in un *file di lock* (*worker.pid*). Se il *file* esiste già e il processo è attivo, il sistema impedisce l'avvio di una seconda istanza.
- **Terminazione chirurgica (Arresto):** gli *script* di arresto (*stop\_all.bat*) non utilizzano comandi generici di chiusura, ma leggono l'*ID* specifico dal *file PID* e inviano un segnale di terminazione mirato (*taskkill /PID*) solo al processo corretto.

Questo approccio eleva l'affidabilità del sistema rispetto a una gestione manuale, assicurando che i processi fantasma vengano rimossi correttamente prima di ogni nuovo avvio.

## 6. Validazione del software

Il software **FileTransfer** è stato sottoposto a un rigoroso processo di validazione collaborativa per garantirne qualità, stabilità e piena aderenza ai requisiti operativi. Questo processo è stato

strutturato attraverso due pilastri fondamentali che hanno visto il coinvolgimento attivo di personale chiave:

- **Sviluppo sinergico e *feedback* continuo:** la fase di sviluppo è stata caratterizzata da una stretta e costante collaborazione tra lo Sviluppatore e il Gruppo Rete FLAME. Questo approccio sinergico ha permesso di raccogliere tempestivamente suggerimenti e osservazioni su funzionalità, usabilità e prestazioni. La comunicazione diretta è stata cruciale per identificare e risolvere rapidamente eventuali problematiche, ottimizzando i tempi di correzione e migliorando significativamente la qualità finale del prodotto.
- **Adozione di cicli iterativi per il miglioramento proattivo:** *dopo la release* iniziale, è stata implementata una metodologia iterativa basata su cicli di revisione rapidi. Questo flusso ininterrotto di *feedback* tra il *team* di sviluppo e il gruppo della rete FLAME ha consentito di introdurre con rapidità modifiche sia correttive che migliorative. Tale prassi ha assicurato che il *software* rimanesse sempre allineato alle emergenti necessità operative, incrementando in modo sostanziale la sua stabilità e affidabilità nel tempo.

Grazie a questa strategia di validazione fondata sulla collaborazione e sull'iterazione continua, **FileTransfer** ha potuto evolvere progressivamente, perfezionando le sue funzionalità e raggiungendo un elevato grado di soddisfazione tra gli utilizzatori finali. Il risultato è un applicativo solido, intuitivo e perfettamente in linea con le esigenze del contesto operativo.

## 6.1 Metriche tecniche e *stress test*

Considerata la natura “*always-on*” del sistema, la fase di *test* si è concentrata sulla stabilità a lungo termine piuttosto che sui picchi di carico (scalabilità verticale), dato il numero noto e limitato di stazioni da monitorare.

**Durata del test:** il sistema è stato sottoposto a un ciclo di esecuzione continua di circa 31 giorni (circa 744 ore).

**Condizioni:** ambiente di rete reale con connessioni verso stazioni remote, simulando interruzioni di rete intermittenti per verificare la resilienza del *database* e il corretto trasferimento dei *file*.

**Metriche rilevate:**

- *Uptime* del servizio: 99.9% durante il periodo di *test*.
- Consumo risorse: utilizzo CPU medio < 5%, Occupazione RAM stabile (nessun *memory leak* rilevato).
- Integrità dati: controllo dei dati trasferiti senza errori e zero corruzioni del *database SQLite* a seguito di riavvii forzati o cadute di connessione.

## 7. Limiti e sviluppi futuri

L'attuale implementazione rappresenta un progetto per operare in un ambiente controllato (*Intranet*) con un numero definito di *client*.

Di seguito si analizzano le limitazioni attuali e le strategie di mitigazione previste per una messa in produzione su larga scala:

- **Sicurezza delle comunicazioni (*HTTP* vs *HTTPS*):** attualmente il servizio comunica via *HTTP*. Dato che l'utilizzo è limitato alla rete locale protetta da *firewall* dell'Istituto, il rischio di intercettazione è considerato accettabile per la fase di *test* e primo sviluppo. In un'evoluzione futura è prevista l'implementazione di un *Reverse Proxy* per gestire la terminazione *SSL/TLS (HTTPS)* e garantire la cifratura del traffico.
- **Gestione credenziali:** le credenziali sono attualmente gestite a livello applicativo per

semplificare il *deployment* iniziale. Si prevede l'integrazione con sistemi di *active directory* per l'autenticazione degli utenti per eliminare qualsiasi dato sensibile in chiaro.

- **Strategia di *disaster recovery* e salvaguardia del dato:** per quanto riguarda la sicurezza e la persistenza delle informazioni, l'attuale strategia di *disaster recovery* si basa sulla robustezza dell'infrastruttura ospitante.
  - Sicurezza del dato: la perdita dei dati è mitigata a livello *hardware*. Il server centrale effettua la scrittura dei file direttamente su una SAN (*Storage Area Network*) istituzionale. Questa infrastruttura è soggetta a *policy* di *backup* automatico e ridondanza gestite dal CED.
  - Continuità del servizio: attualmente, il ripristino del servizio in caso di guasto richiede un intervento manuale di riconfigurazione su una nuova macchina. Per automatizzare questo processo e ridurre i tempi di fermo, la *roadmap* potrebbe prevedere la migrazione verso un'architettura containerizzata (*Docker*). Questo permetterà di riavviare l'intera istanza applicativa su un qualsiasi nodo del *cluster* di calcolo in pochi secondi, svincolandosi dalla dipendenza fisica dal singolo server *Windows* attuale.

## 8. Conclusioni

L'applicazione *FileTransfer* ha raggiunto pienamente gli obiettivi prefissati, realizzando un sistema completamente automatizzato che ha integralmente centralizzato e reso autonomo un processo che prima richiedeva anche interventi manuali. Questo sistema è in grado di anticipare i problemi e offre funzionalità chiave quali:

- **Monitoraggio in tempo reale** dello stato delle stazioni e della salute delle connessioni.
- **Gestione della configurazione** semplice e sicura, eliminando la necessità di modifiche al codice.
- **Analisi dei log e dello storico delle operazioni**, migliorando il controllo e la tracciabilità.

Questa soluzione non solo assicura **integrità e continuità operativa**, ma getta le basi per sviluppi futuri come l'integrazione di sistemi di *alert*, reportistica avanzata e analisi predittiva.

## 9. Disponibilità del Codice Sorgente

In risposta alle esigenze di trasparenza, replicabilità e riutilizzo da parte di altri gruppi di ricerca dell'Istituto, l'intero codice sorgente del progetto, completo di *file* di configurazione, script di installazione automatica e lista delle dipendenze (*requirements.txt*) e guida all'installazione, è stato pubblicato sul *repository*:

- **Repository URL:** <https://gitlab.rm.ingv.it/carmelo.lapiana/filetransfer>
- **Licenza:** *Open Source*.

Le appendici di questo documento riportano, per comodità di consultazione, solo gli estratti più significativi del codice, rimandando al *repository* per la versione completa e mantenuta.

## Ringraziamenti

Si desidera esprimere un sincero ringraziamento al gruppo della rete FLAME per il supporto e, in particolare, al dott. Giuseppe Salerno, al dott. Roberto Maugeri e a Filippo Murè, il cui contributo ha reso possibile la realizzazione del progetto descritto nel presente rapporto.

## Bibliografia

Questa sezione elenca i principali protocolli e i *framework* tecnologici adottati come fondamento architetturale e funzionale del sistema.

API REST (Representational State Transfer). Standard architetturale per la comunicazione distribuita e asincrona tra i componenti del sistema.

SHA-256 (Secure Hash Algorithm 256-bit). Funzione crittografica di *hash* utilizzata per garantire l'integrità dei dati (Controllo *Hash*) durante il trasferimento dei *file*.

Python. Linguaggio di programmazione principale utilizzato per l'intera logica di *backend* (*Worker*) e l'applicazione *web* (*dashboard*).

Flask. Micro-*framework* Python utilizzato per costruire l'interfaccia utente centralizzata.

FastAPI. *Framework* Python ad alte prestazioni adottato per il *Server* della Stazione, garantendo la velocità e l'affidabilità degli endpoint *API*.

SQLite. *Database* relazionale leggero utilizzato per la persistenza dello stato, dei *log* e dello storico.

Bootstrap 5. *Framework* CSS "*mobile-first*" utilizzato per il *design responsive* e l'interfaccia utente.

## Sitografia

Questa sezione elenca i riferimenti web della documentazione ufficiale per le librerie e i *framework* utilizzati.

[1] Python (Documentazione Ufficiale): <https://docs.python.org/>

[2] Flask (Micro-*framework*): <https://flask.palletsprojects.com/>

[3] Waitress (WSGI Server): <https://docs.pylonsproject.org/projects/waitress/en/latest/>

[4] FastAPI (API Framework): <https://fastapi.tiangolo.com/>

[5] Requests (Libreria HTTP): <https://docs.python-requests.org/>

[6] SQLite (*database*): <https://www.sqlite.org/docs.html>

[7] Bootstrap 5 (Design Framework): <https://getbootstrap.com/docs/5.3/getting-started/introduction/>

[8] DynDNS@INGV: <https://doi.org/10.13127/rpt/420>



# APPENDICE A



Il file *config.ini* (PC di acquisizione) centralizza tutte le variabili operative del sistema, permettendo una facile manutenzione senza modificare il codice sorgente.

```
[General]
PROJECT_PATH                =                C:\FileTransfer
API_KEY                     =                "la chiave segreta"

[Worker]
BASE_DESTINATION_PATH      =                E:\UVDATA\

MAX_WORKERS                 =                20

[WebApp]
PORT                       =                8080
NOTIFY_URL                 =                http://localhost:8080

[StationServer]
# Questa sezione è ignorata dal PC Centrale, ma la lasciamo per coerenza
SOURCE_DATA_PATH           =                C:\Ignorato
PORT = 5000
```

1. **PROJECT\_PATH**: percorso radice dell'applicazione (dove si trovano le cartelle 'python', 'scripts', 'data')
2. **API\_KEY**: chiave API segreta. Deve essere identica su tutti i server delle stazioni.
3. **BASE\_DESTINATION\_PATH**: percorso di base dove il worker salverà i *file* scaricati (può essere un'unità di rete).
4. **MAX\_WORKERS**: numero massimo di stazioni da contattare in parallelo.
5. **PORT**: porta su cui sarà accessibile l'interfaccia web della dashboard.
6. **NOTIFY\_URL**: URL usato internamente dal worker per notificare l'interfaccia
7. **SOURCE\_DATA\_PATH**: percorso da cui il server della stazione legge i *file* da spedire.
8. **PORT**: porta su cui il server della stazione si mette in ascolto.

Il file *config.ini* (PC Stazione FLAME) centralizza tutte le variabili operative del sistema, permettendo una facile manutenzione senza modificare il codice sorgente.

```
[General]
# QUESTA CHIAVE DEVE ESSERE IDENTICA A QUELLA DEL PC CENTRALE
API_KEY = "la chiave segreta"

[StationServer]
# PERCORSO CORRETTO DA CUI LA STAZIONE LEGGE I FILE
SOURCE_DATA_PATH = C:\CondivisioneDati\UVDData

# Porta su cui questo server si mette in ascolto
PORT = 5000
```

1. **API\_KEY:** deve essere la stessa, identica stringa che hai impostato nel *config.ini* del PC centrale.
2. **SOURCE\_DATA\_PATH:** è il percorso sulla stazione da cui il programma leggerà i *file*.
3. **Sezioni Mancanti:** le sezioni [Worker] e [WebApp] non sono necessarie qui, perché lo script *station\_server.py* non le legge e non gli servono per funzionare.

# APPENDICE B

*station\_server.py* in esecuzione sulle stazioni remote

```
import os, hashlib, logging, configparser, sys
from fastapi import FastAPI, HTTPException, Security
from fastapi.security import APIKeyHeader
from fastapi.responses import FileResponse, JSONResponse
from pathlib import Path
from pydantic import BaseModel
from typing import List
config = configparser.ConfigParser()
config_path = os.path.join(os.path.dirname(__file__), '..', 'config', 'config.ini')
if not os.path.exists(config_path):
    print(f"ERRORE: File di configurazione non trovato in '{config_path}'")
    sys.exit(1)

config.read(config_path)
BASE_DATA_PATH = config.get('StationServer', 'SOURCE_DATA_PATH')
API_KEY = config.get('General', 'API_KEY')
STATION_SERVER_PORT = config.getint('StationServer', 'PORT')
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
app = FastAPI()
os.makedirs(BASE_DATA_PATH, exist_ok=True)

api_key_header = APIKeyHeader(name="X-API-Key")
async def api_key_security(x_api_key: str = Security(api_key_header)):
    if x_api_key != API_KEY:
        raise HTTPException(status_code=401, detail="Chiave API non valida o mancante")
def calculate_sha256(file_path):
    sha256_hash = hashlib.sha256()
    try:
        with open(file_path, "rb") as f:
            for byte_block in iter(lambda: f.read(4096), b''):
                sha256_hash.update(byte_block)
        return sha256_hash.hexdigest()
    except PermissionError as e:
        logging.warning(f"Permesso negato durante il calcolo hash per {file_path}. Il file è bloccato: {e}")
        raise HTTPException(status_code=423, detail=f"File bloccato: {e}") # 423 Locked
    except Exception as e:
        logging.error(f"Errore imprevisto durante il calcolo hash per {file_path}: {e}")
        raise HTTPException(status_code=500, detail=f"Errore interno: {e}")

def get_safe_path(base_path, relative_path):
    if not relative_path:
        raise HTTPException(status_code=400, detail="Percorso non valido.")
    full_path = Path(os.path.join(base_path, relative_path)).resolve()
    base_path = Path(base_path).resolve()
    if base_path not in full_path.parents and str(full_path) != str(base_path):
        raise HTTPException(status_code=400, detail="Accesso non consentito.")
    return str(full_path)
```

```

@app.get("/files", dependencies=[Security(api_key_security)])
def list_files():
    all_files = []
    ignore_files = ['Thumbs.db', '.DS_Store']
    # --- MODIFICA CHIAVE QUI: Aggiunto '.log' alla lista ---
    ignore_extensions = ['.tmp', '.temp', '.bak', '.~', '.log']
    # --- FINE MODIFICA ---

    if not os.path.isdir(BASE_DATA_PATH): return {"files": []}

    for root, dirs, files in os.walk(BASE_DATA_PATH):
        for name in files:
            if not name or name in ignore_files or any(name.lower().endswith(ext) for ext in
ignore_extensions):
                logging.info(f"File ignorato perché nella lista di esclusione: {name}")
                continue

            full_path = os.path.join(root, name)
            relative_path = os.path.relpath(full_path, BASE_DATA_PATH)
            if relative_path and relative_path != '.':
                all_files.append(relative_path.replace('\\', '/'))
    return {"files": all_files}

class FolderList(BaseModel):
    folders: List[str]

@app.post("/bulk-delete-folders", dependencies=[Security(api_key_security)])
def bulk_delete_folders(payload: FolderList):
    deleted, failed = [], {}
    for folder_name in payload.folders:
        try:
            folder_path = get_safe_path(BASE_DATA_PATH, folder_name)
            if os.path.isdir(folder_path):
                if not os.listdir(folder_path):
                    os.rmdir(folder_path); deleted.append(folder_name)
                else:
                    failed[folder_name] = "La cartella non è vuota."
            else:
                failed[folder_name] = "Non è una cartella."
        except Exception as e:
            failed[folder_name] = str(e)
    logging.info(f"Cancellazione massiva: {len(deleted)} OK, {len(failed)} fallite.")
    if failed: logging.warning(f"Dettagli fallimenti: {failed}")
    return {"status": "completed", "deleted": deleted, "failed": failed}

@app.get("/download/{filepath:path}", dependencies=[Security(api_key_security)])
def download_file(filepath: str):
    safe_full_path = get_safe_path(BASE_DATA_PATH, filepath)
    if os.path.exists(safe_full_path):
        try:

```

```

        return FileResponse(path=safe_full_path, media_type='application/octet-stream',
filename=os.path.basename(filepath))
    except Exception as e:
        logging.error(f"Errore durante il download di {safe_full_path}: {e}")
        raise HTTPException(status_code=423, detail=f"File bloccato: {e}")
        raise HTTPException(status_code=404, detail="File not found")

@app.get("/file-hash/{filepath:path}", dependencies=[Security(api_key_security)])
def get_file_hash(filepath: str):
    safe_full_path = get_safe_path(BASE_DATA_PATH, filepath)
    if os.path.exists(safe_full_path):
        return {"filename": filepath, "sha256": calculate_sha256(safe_full_path)}
    raise HTTPException(status_code=404, detail="File not found")

@app.delete("/delete/{filepath:path}", dependencies=[Security(api_key_security)])
def delete_file(filepath: str):
    safe_full_path = get_safe_path(BASE_DATA_PATH, filepath)
    if not os.path.exists(safe_full_path):
        raise HTTPException(status_code=404, detail="File not found")
    try:
        os.remove(safe_full_path)
        logging.info(f"File '{safe_full_path}' cancellato con successo.")
        return {"status": "success", "filename": filepath}
    except OSError as e:
        logging.error(f"Errore OS durante la cancellazione di '{safe_full_path}': {e}")
        raise HTTPException(status_code=403, detail=f"Impossibile cancellare il file. Potrebbe essere bloccato o mancano i permessi. Dettagli: {e}")
    except Exception as e:
        logging.error(f"Errore imprevisto durante la cancellazione di '{safe_full_path}': {e}")
        raise HTTPException(status_code=500, detail=f"Errore del server: {e}")

if __name__ == '__main__':
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=STATION_SERVER_PORT)

```

il *worker.py* in esecuzione nel PC di acquisizione

```

import os
import sys
import requests
import time
import shutil
import hashlib
import json
import sqlite3
import configparser
from datetime import datetime, date

```

```

from concurrent.futures import ThreadPoolExecutor, wait
from requests.adapters import HTTPAdapter
from urllib3.util.retry import Retry

config = configparser.ConfigParser()
config_path = os.path.join(os.path.dirname(__file__), '..', 'config', 'config.ini')
config.read(config_path)
PROJECT_PATH = config.get('General', 'PROJECT_PATH')
API_KEY = config.get('General', 'API_KEY')
BASE_DESTINATION_PATH = config.get('Worker', 'BASE_DESTINATION_PATH')
MAX_WORKERS = config.getint('Worker', 'MAX_WORKERS')
WEB_APP_URL = config.get('WebApp', 'NOTIFY_URL')

DB_FOLDER = os.path.join(PROJECT_PATH, 'data', 'db')
os.makedirs(DB_FOLDER, exist_ok=True)

STATUS_FILE = os.path.join(PROJECT_PATH, 'status.json')
STATIONS_FILE = os.path.join(PROJECT_PATH, 'config', 'stations.json')
PID_FILE = os.path.join(PROJECT_PATH, 'worker.pid')
LAST_RESET_FILE = os.path.join(PROJECT_PATH, 'last_daily_task.txt')

def get_current_db_path():
    return os.path.join(DB_FOLDER, datetime.now().strftime('history_%Y_%m.db'))

def create_pid_file():
    try:
        with open(PID_FILE, 'w', encoding='ascii') as f:
            f.write(str(os.getpid()))
    except Exception as e:
        with open(os.path.join(PROJECT_PATH, 'worker_error.log'), 'a') as log_f:
            log_f.write(f'{datetime.now()}: Impossibile creare il file PID: {e}\n')

def setup_database():
    db_path = get_current_db_path()
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
    cursor.execute('CREATE TABLE IF NOT EXISTS logs (id INTEGER PRIMARY KEY
    AUTOINCREMENT, timestamp DATETIME NOT NULL, station_name TEXT, level TEXT NOT
    NULL, message TEXT NOT NULL, filename TEXT, filesize_kb REAL)')
    cursor.execute('CREATE TABLE IF NOT EXISTS station_status (station_name TEXT PRIMARY
    KEY, current_status TEXT, last_contact DATETIME, files_today INTEGER DEFAULT 0,
    consecutive_errors INTEGER DEFAULT 0)')
    cursor.execute('CREATE INDEX IF NOT EXISTS idx_logs_timestamp ON logs(timestamp)')
    conn.commit()
    conn.close()

def log_event(station_name, level, message, filename=None, filesize_kb=None):
    db_path = get_current_db_path()
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()

```

```

        cursor.execute('INSERT INTO logs (timestamp, station_name, level, message, filename,
        filesize_kb) VALUES (?, ?, ?, ?, ?, ?)',
                        (datetime.now(), station_name, level, message, filename, filesize_kb))
        conn.commit()
        conn.close()

def update_station_status(station_name, status, error_increment=0):
    db_path = get_current_db_path()
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
    cursor.execute("INSERT OR IGNORE INTO station_status (station_name) VALUES (?)",
    (station_name,))
    if status == "Online":
        cursor.execute("UPDATE station_status SET current_status = 'Online', last_contact = ?,
        consecutive_errors = 0 WHERE station_name = ?", (datetime.now(), station_name))
    elif status == "Errore":
        cursor.execute("UPDATE station_status SET current_status = 'Errore', last_contact = ?,
        consecutive_errors = consecutive_errors + ? WHERE station_name = ?", (datetime.now(), error_
        increment, station_name))
    conn.commit()
    conn.close()

def increment_station_file_count(station_name):
    db_path = get_current_db_path()
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
    cursor.execute("UPDATE station_status SET files_today = files_today + 1 WHERE station_name
    = ?", (station_name,))
    conn.commit()
    conn.close()

def run_daily_tasks():
    today_str = str(date.today())
    last_run_str = ""
    if os.path.exists(LAST_RESET_FILE):
        with open(LAST_RESET_FILE, 'r') as f:
            last_run_str = f.read()
    if today_str != last_run_str:
        log_event(None, "INFO", "Azzero i contatori giornalieri.")
        db_path = get_current_db_path()
        conn = sqlite3.connect(db_path)
        cursor = conn.cursor()
        cursor.execute("UPDATE station_status SET files_today = 0")
        conn.commit()
        conn.close()
        with open(LAST_RESET_FILE, 'w') as f:
            f.write(today_str)

def create_retry_session():
    session = requests.Session()

```

```

    retry_strategy = Retry(total=3, backoff_factor=1, status_forcelist=[500, 502, 503, 504], allowed_
methods=["HEAD", "GET", "POST", "DELETE"])
    adapter = HTTPAdapter(max_retries=retry_strategy)
    session.mount("http://", adapter)
    session.mount("https://", adapter)
    return session

def update_status(status, message=""):
    try:
        with open(STATUS_FILE, 'w') as f:
            json.dump({'status': status, 'timestamp': datetime.now().isoformat(), 'message':
message}, f)
    except Exception as e:
        log_event(None, "ERROR", f"Impossibile aggiornare il file di stato: {e}")

def load_stations():
    try:
        with open(STATIONS_FILE, 'r') as f:
            return json.load(f)
    except:
        return []

def calculate_local_sha256(file_path):
    sha256_hash = hashlib.sha256()
    try:
        with open(file_path, "rb") as f:
            for byte_block in iter(lambda: f.read(4096), b''):
                sha256_hash.update(byte_block)
            return sha256_hash.hexdigest()
    except:
        return None

def process_station(station, session):
    station_name, station_url, location = station.get("name", "N/D"), station.get("url", ""), station.
get("location", "Sconosciuta")
    if not station_url:
        log_event(station_name, "ERROR", "Configurazione stazione incompleta.")
        return

    headers = {"X-API-Key": API_KEY}
    log_event(station_name, "INFO", f"--- Inizio controllo (Località: {location}) ---")

    try:
        response = session.get(f"{station_url}/files", timeout=20, headers=headers)
        response.raise_for_status()
        files_to_download = response.json().get("files", [])

        update_station_status(station_name, "Online")

        if not files_to_download:

```

```

log_event(station_name, "INFO", "Nessun file nuovo.")
return

for relative_filepath in files_to_download:
    if not relative_filepath or not isinstance(relative_filepath, str): continue
    filename = os.path.basename(relative_filepath)
    if not filename or filename == '.': continue

    try:
        source_folder_full = os.path.dirname(relative_filepath).replace('\\', '/').split('/')[0]
        original_date = datetime.strptime(source_folder_full[:6], '%y%m%d')
        year, month, day_folder = original_date.strftime('%Y'), original_date.strftime('%m'),
source_folder_full[:6]
    except (ValueError, IndexError):
        now = datetime.now()
        year, month, day_folder = now.strftime('%Y'), now.strftime('%m'), now.
strftime('%y%m%d')

    destination_path = os.path.join(BASE_DESTINATION_PATH, location, station_name,
year, month, day_folder)
    os.makedirs(destination_path, exist_ok=True)
    final_dest_path = os.path.join(destination_path, filename)
    tmp_dest_path = final_dest_path + ".tmp"

    try:
        response_hash = session.get(f"{station_url}/file-hash/{relative_filepath}", timeout=30,
headers=headers)
        response_hash.raise_for_status()
        original_hash = response_hash.json()["sha256"]
        with session.get(f"{station_url}/download/{relative_filepath}", stream=True,
timeout=120, headers=headers) as r:
            r.raise_for_status()
            with open(tmp_dest_path, 'wb') as f:
                shutil.copyfileobj(r.raw, f)

        filesize_kb = round(os.path.getsize(tmp_dest_path) / 1024, 2)
        local_hash = calculate_local_sha256(tmp_dest_path)

        if original_hash and original_hash == local_hash:
            shutil.move(tmp_dest_path, final_dest_path)
            log_event(station_name, "SUCCESS", f"File COPIATO in '{destination_path}'.
(Cancellazione disabilitata per test)", filename, filesize_kb)
            increment_station_file_count(station_name)
        else:
            except Exception as e:
                log_event(station_name, "ERROR", f"Errore processo file: {e}", filename)
            if os.path.exists(tmp_dest_path):
                os.remove(tmp_dest_path)
    except Exception as e:
        log_event(station_name, "CRITICAL", f"Errore critico stazione: {e}")

```

```

    update_station_status(station_name, "Errore", error_increment=1)
if __name__ == "__main__":
    create_pid_file()
    setup_database()
    log_event(None, "INFO", "Servizio avviato in modalità parallela.")
    update_status("running", "Servizio avviato.")
    session = create_retry_session()

    try:
        while True:
            run_daily_tasks()

            stations = load_stations()
            if stations:
                log_event(None, "INFO", f"===== Inizio ciclo per {len(stations)} stazioni =====")

                # --- MODIFICA CHIAVE: Logica con Heartbeat Attivo ---
                with ThreadPoolExecutor(max_workers=MAX_WORKERS) as executor:
                    # Sottomette i task e ottiene i "futures"
                    futures = {executor.submit(process_station, s, session) for s in stations}

                    # Finché ci sono task in esecuzione...
                    while futures:
                        # Aspetta per un massimo di 30 secondi che qualche task finisca
                        done, futures = wait(futures, timeout=30)

                        # Invia un "battito cardiaco" per dire che sei vivo!
                        message = f"In elaborazione... {len(futures)} stazioni rimanenti."
                        update_status("running", message)
                    # --- FINE MODIFICA ---

                log_event(None, "INFO", "===== Ciclo parallelo completato. =====")
            else:
                log_event(None, "INFO", "Nessuna stazione configurata.")

            update_status("idle", "In attesa del prossimo ciclo.")
            time.sleep(60)
    finally:
        log_event(None, "INFO", "Servizio fermato.")
        update_status("stopped", "Fermato.")
        if os.path.exists(PID_FILE):
            os.remove(PID_FILE)

```



## QUADERNI di GEOFISICA

ISSN 1590-2595

<https://collane-editoriali.ingv.it/index.php/qdg/it/index/>

QUADERNI di GEOFISICA è una rivista scientifica che pubblica dati, misurazioni e osservazioni geofisiche (anche preliminari) in italiano e inglese. L'obiettivo è una diffusione rapida e gratuita per la comunità scientifica, garantita da un processo di peer-review e da un Editorial Board multidisciplinare. La rivista è inclusa in "Emerging Sources Citation Index" di Clarivate Analytics e "Open Access Journals" di Scopus, e adotta il modello Diamond Open Access, che non prevede costi né per gli autori né per i lettori.

QUADERNI di GEOFISICA is a scientific journal that publishes geophysical data, measurements and observations, including preliminary results, in Italian and English. Its aim is to ensure rapid and free dissemination to the research community, supported by a peer-review process and by a multidisciplinary Editorial Board. The journal is included in the Emerging Sources Citation Index of Clarivate Analytics and it's included in the Open Access journals of Scopus. It follows the Diamond Open Access model, with no charges for either authors or readers.



## RAPPORTI TECNICI INGV

ISSN 2039-7941

<https://collane-editoriali.ingv.it/index.php/rt/it/index>

RAPPORTI TECNICI INGV è una rivista tecnico-scientifica che mette in mostra le numerose attività tecnologiche e di ricerca condotte dai ricercatori e tecnici INGV, e non solo. Pubblicata in italiano e inglese, RAPPORTI TECNICI INGV è ideata per valorizzare anche quei lavori che offrono un contributo significativo dal punto di vista tecnico, operativo o documentale fondamentale per lo sviluppo dell'Ente. La rivista adotta il modello Diamond Open Access, che non prevede costi né per gli autori né per i lettori.

RAPPORTI TECNICI INGV is a technical-scientific journal that showcases the wide range of technological and research activities carried out by INGV researchers and technical staff, as well as by external contributors. Published in Italian and English, RAPPORTI TECNICI INGV is designed to highlight works that provide a significant contribution from a technical, operational, or documentary perspective, and that are essential to INGV development. The journal follows the Diamond Open Access model, with no charges for either authors or readers.



## MISCELLANEA INGV

ISSN 2039-6651

<https://collane-editoriali.ingv.it/index.php/mi/it/index>

MISCELLANEA INGV rappresenta una tipologia di editoria non tradizionalmente catalogabile nelle ordinarie riviste scientifiche di settore. Pubblicata in lingua italiana e inglese, MISCELLANEA INGV offre uno spazio per la pubblicazione di materiali eterogenei che, pur non rientrando nei formati più strutturati delle altre Collane editoriali INGV, rivestono un interesse scientifico, tecnico o divulgativo. La rivista adotta il modello Diamond Open Access, che non prevede costi né per gli autori né per i lettori.

MISCELLANEA INGV represents a form of publishing that cannot be classified within the standard scientific journals. Published in Italian and English, MISCELLANEA INGV provides a venue for the publication of heterogeneous materials that are of scientific, technical or outreach interest, even though they do not fall within the more structured formats of the other INGV editorial series. The journal follows the Diamond Open Access model, with no charges for either authors or readers.

**Progetto grafico**  
Barbara ANGIONI  
Istituto Nazionale di Geofisica e Vulcanologia

**Ufficio Editoriale**  
Francesca DI STEFANO | Coordinatore

**Segreteria di redazione**  
segreteria.collane-editoriali@ingv.it

**Editing | Produzione | Grafica redazionale**  
Barbara ANGIONI  
Massimiliano CASCONI  
Francesca DI STEFANO  
Patrizia PANTANI

© 2026 the Author(s). Open Access.  
This article is licensed under a Creative Commons Attribution 4.0 International License.



Creative Commons Attribution 4.0 International



ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA

