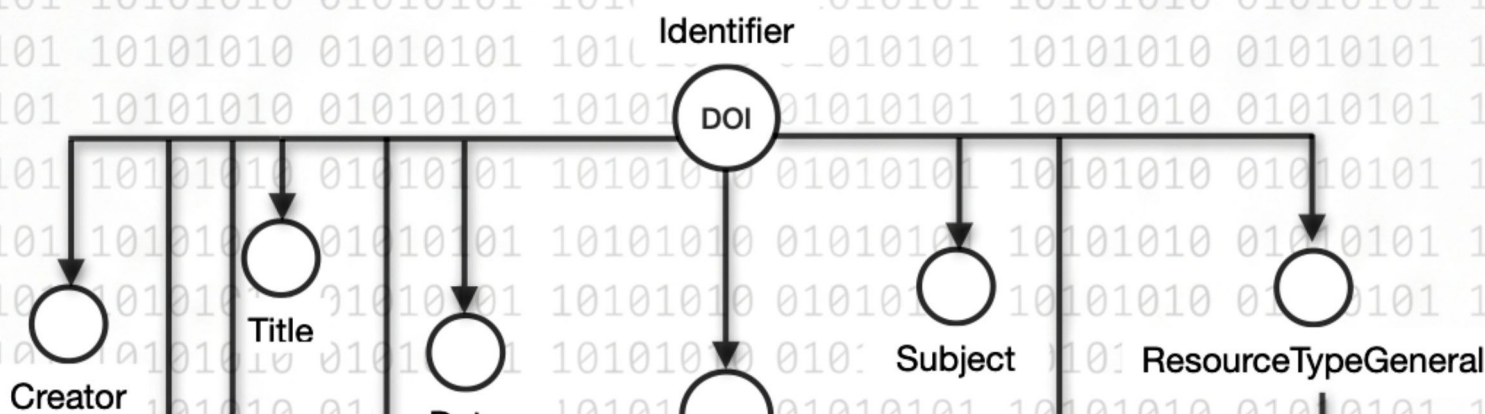


MetaCat

< MetaData Catalog / >



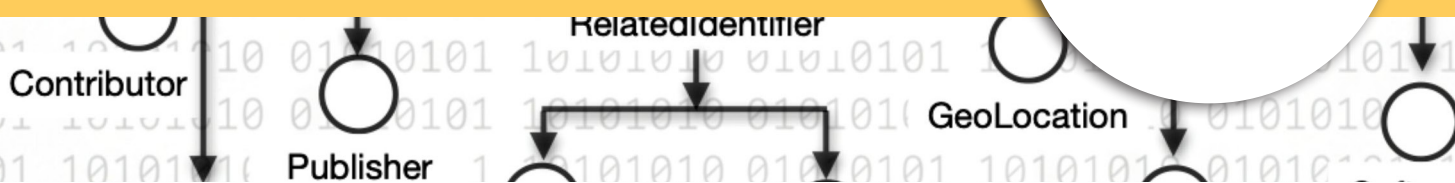
RAPPORTI TECNICI INGV

MetaCat: un catalogo per
Metadati DataCite



ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA

524



**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

Stefano PINTORE

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



ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA

RAPPORTI TECNICI INGV

MetaCat: un catalogo per Metadati DataCite

MetaCat: a DataCite Metadata catalog

Fabrizio Pistagna*

INGV | Istituto Nazionale di Geofisica e Vulcanologia, Sezione di Catania - Osservatorio Etneo

*Corresponding author

Accettato 19 giugno 2026 | Accepted 19 June 2026

Come citare | How to cite Pistagna, F. (2026). MetaCat: un catalogo per Metadati DataCite. *Rapporti Tecnici INGV*, 524, 1-34. <https://doi.org/10.13127/rpt/524>

In copertina Rappresentazione stilizzata a grafo di uno schema di MetaDati Datacite | Cover Stylized graph representation of a Datacite MetaData schema

524

INDICE

Riassunto	7
<i>Abstract</i>	7
Introduzione	8
1. Contesto: il Patrimonio Dati dell'INGV	9
2. MetaCat: un Catalogo di Metadati	10
2.1 Filosofia di Progettazione	11
3. MetaCat: Architettura <i>software</i>	12
3.1 Livello di Validazione e Persistenza: la “Doppia Difesa”	13
3.1.1 Validazione Esterna (AJV: Il <i>Gatekeeper</i> dell'API)	13
3.1.2 Validazione Interna e Modellazione (Mongoose)	14
3.2 Livello Logico: il <i>Pattern Controller-Service-DAO</i>	14
3.2.1 <i>Controller</i> : Gestione del Protocollo HTTP	15
3.2.2 <i>Service</i> : Logica di <i>Business</i> e Dominio	15
3.2.3 <i>DAO (Data Access Object)</i> : Astrazione dello <i>Storage</i>	16
3.3 Livello API: <i>Endpoint</i> pragmatici <i>REST-like</i>	16
3.3.1 Autenticazione ibrida (<i>authRoutes.js</i>)	16
3.3.2 Ciclo di vita del <i>Record (recordRoutes.js)</i>	18
3.3.2.1 Creazione Bozza: POST /records	19
3.3.2.2 Modifica e aggiornamento <i>record</i>	19
3.3.2.3 Modifica di un attributo	19
3.3.2.4 Altre operazioni	20
3.4 Sicurezza: <i>Pipeline</i> di <i>Middleware</i>	21
3.4.1 Autenticazione (<i>authenticationMiddleware.js</i>)	21
3.4.2 Autorizzazione (<i>authorizationMiddleware.js</i>)	21
4. Risultati: <i>deployment</i> e validazione	22
4.1 Modello di <i>deployment</i>	22
4.2 Caso d'uso: <i>Ingestion</i> del Catalogo Metadati INGV	22
4.2.1 <i>Data Preparation</i> e Strategia di recupero Metadati	23
4.2.2 Obiettivi e Metodologia del <i>Benchmark</i>	24
4.3 Esecuzione del <i>Benchmark</i>	24
4.3.1 <i>Baseline Performance: Load Test</i>	24
4.3.2 Saturazione e Resilienza: <i>Stress Test</i>	25
5. Conclusioni	26
Bibliografia	27
Sitografia	28

Riassunto

La proliferazione di dati e prodotti scientifici digitali richiede sistemi di gestione e catalogazione robusti per garantire la conformità ai principi FAIR (*Findable, Accessible, Interoperable, Reusable*). I metadati standardizzati associati ai *dataset* sono il fondamento per raggiungere questo obiettivo. Questo rapporto tecnico presenta **MetaCat**, un servizio *backend* specializzato, sviluppato per affrontare la sfida della metadatazione, fungendo esclusivamente da **Catalogo di Metadati**. Concepito per essere agile e leggero, MetaCat implementa rigorosamente lo standard DataCite Metadata Schema (v4.6) disaccoppiando la gestione dei metadati dallo *storage* fisico dei dati. Il sistema espone un'architettura *API-first* di tipo REST per le operazioni di Creazione, Lettura, Aggiornamento e Cancellazione (CRUD) dei *record* di metadati. Sviluppato su uno *stack* moderno (Node.js, Express, MongoDB), MetaCat integra un sistema di autenticazione ibrido (JWT locale e OAuth2/ORCID) e un modello di autorizzazione granulare basato su ruoli (*user, curator, admin*) per gestire un flusso di pubblicazione completo (da bozze private a *record* pubblici). L'applicazione è interamente *containerizzata* tramite Docker e orchestrata con Docker Compose, garantendo portabilità, scalabilità e aderenza alle moderne pratiche DevOps. MetaCat è rilasciato come prodotto di ricerca *open-source* (AGPL v3.0) per fornire ad altre istituzioni uno strumento estendibile per la catalogazione di *record* di metadati.

Abstract

The proliferation of digital scientific data and products necessitates robust management and cataloguing systems to ensure compliance with the FAIR principles (Findable, Accessible, Interoperable, Reusable). Standardised metadata associated with datasets form the bedrock for achieving this objective. This technical report introduces MetaCat, a specialised backend service developed to address the challenge of metadata management, serving exclusively as a Metadata Catalog. Conceived to be agile and lightweight, MetaCat rigorously implements the DataCite Metadata Schema (v4.6) standard, decoupling metadata management from the physical data storage. The system exposes a pragmatic, REST-like, API-first architecture for Create, Read, Update, and Delete (CRUD) operations on metadata records. Developed on a modern stack (Node.js, Express, MongoDB), MetaCat integrates a hybrid authentication system (local JWT and OAuth2/ORCID) and a granular role-based authorisation model (user, curator, admin) to manage a complete publication workflow (from private drafts to public records). The application is entirely containerised via Docker and orchestrated with Docker Compose, ensuring portability, scalability, and adherence to contemporary DevOps practices. MetaCat is released as an open-source research product (AGPL v3.0) to provide other institutions an extensible tool for cataloguing their scientific metadata records.

Keywords Schema Datacite; Principi FAIR; Catalogo di Metadati | Datacite Schema; FAIR principles; Metadata Catalog

Introduzione

Negli ultimi anni, il notevole incremento della produzione scientifica dettato dalla sempre crescente capacità di risorse sia computazionali che di memorizzazione (*storage*), ha generato un volume senza precedenti di risorse digitali. Questo fenomeno non riguarda esclusivamente le pubblicazioni tradizionali (*paper*, articoli, libri), ma include soprattutto i dati grezzi ed elaborati raccolti in *dataset*, i *software* di ricerca che li hanno generati e i modelli computazionali utilizzati. Di fronte a questa abbondanza di informazioni eterogenee, la comunità scientifica si è presto confrontata con una criticità fondamentale: in assenza di una descrizione standardizzata, questi prodotti rischiano di rimanere “invisibili”, difficili da rintracciare, validare e riutilizzare.

Questo scenario ha catalizzato lo sviluppo di iniziative globali come la filosofia “**Open Data**” [1] e l'adozione dei principi **FAIR** (*Findable, Accessible, Interoperable, Reusable*) [2].

La filosofia “*Open Data*” affonda le proprie radici nell'ultimo decennio del XX secolo, traendo ispirazione dalla cultura della condivisione digitale nata originariamente con il movimento del *Software Libero* (*Free Software*). Questo movimento, istituzionalizzato dalla *Free Software Foundation* (FSF) [3] fondata da Richard Stallman, era guidato principalmente da rigorose motivazioni etiche e dalla tutela delle libertà fondamentali dell'utente. Negli anni successivi, questa filosofia ha trovato una declinazione più pragmatica, metodologica e orientata all'efficienza dello sviluppo collaborativo attraverso il movimento *Open Source* [Stallman, n.d.] [Yeats, 2007]. È proprio dall'intersezione tra le istanze etiche originarie del *Software Libero* e i modelli operativi di successo dell'*Open Source* che la comunità scientifica ha tratto il paradigma ideale per estendere l'accesso aperto anche ai prodotti della ricerca, gettando le basi per l'attuale ecosistema *Open Data*. Si ritiene che il termine stesso sia apparso per la prima volta verso la metà degli anni '90 in un rapporto di un'agenzia federale americana (verosimilmente la NOAA - National Oceanic and Atmospheric Administration) [4], che promuoveva lo scambio completo e aperto di informazioni scientifiche come prerequisito per la ricerca ambientale globale.

Con l'espansione di Internet e del *World Wide Web*, divenne prioritario garantire un identificatore univoco e persistente per i contenuti digitali, indipendente dalla loro collocazione fisica sui *server*, per risolvere il problema dei collegamenti interrotti - *broken link*. A tale scopo, alla fine degli anni '90 fu istituita l'International DOI Foundation (IDF) [5], l'organizzazione non-profit responsabile dello sviluppo e della promozione del DOI (*Digital Object Identifier*) [6].

Nel primo decennio del XXI secolo, i tempi maturarono per la formalizzazione delle filosofie “*Open*” nella ricerca scientifica. La Budapest Open Access Initiative (BOAI) del 2002 [Budapest Open Access Initiative (BOAI), 2002] sancì la nascita del movimento globale per l'*Open Access* (OA) [7]. Successivamente, nel 2006, l'OCSE adottò una raccomandazione sull'accesso ai dati della ricerca finanziata con fondi pubblici, gettando le basi per le iniziative attuali. A chiusura del decennio, il 1° dicembre 2009, venne fondata l'organizzazione DataCite [Neumann et al., 2014] [8]: una risposta pratica all'esigenza di un'infrastruttura affidabile per rendere i *dataset* citabili, identificabili e riutilizzabili al pari delle pubblicazioni testuali.

Il 15 marzo 2016 segna un punto di svolta con la pubblicazione su *Scientific Data* dell'articolo “*The FAIR Guiding Principles for scientific data management and stewardship*” [Wilkinson et al., 2016]. Tali principi definiscono le linee guida per migliorare la gestibilità dei dati scientifici, garantendo che siano non solo aperti, ma anche utilizzabili efficacemente sia da esseri umani che da processi automatizzati - *Machine-to-Machine*¹ (M2M). Tali normative sono state ulteriormente rafforzate dalla Direttiva (UE) 2019/1024, nota come “Direttiva *Open Data*”.

Il pilastro per rendere un dato FAIR è la sua metadattazione: la creazione di una “carta d'identità” digitale, il cui contenuto - i metadati - viene definito a partire da schemi stabiliti e memorizzato in strutture dati leggibili dalle macchine.

¹ *Machine-to-Machine* (M2M) è un paradigma tecnologico il cui prevede che differenti dispositivi collegati in rete possano scambiare dati automaticamente ed eseguire azioni vicendevolmente senza necessario intervento umano, utilizzando sensori, software e reti di calcolatori per comunicare direttamente.

Per rispondere a questa esigenza, la comunità scientifica ha converso verso standard di metadati robusti e interoperabili. Tra questi, il DataCite Metadata Schema [DataCite Metadata Working Group, 2024] si è affermato come standard *de-facto*. Progettato e mantenuto da DataCite, questo schema permette l'identificazione univoca di un'ampia gamma di prodotti della ricerca (*dataset*, *software*, *workflow*, campioni fisici) tramite l'assegnazione di un **DOI univoco**. Oltre all'identificazione, lo schema fornisce un insieme di proprietà essenziali (Creatori, Titolo, Editore, Anno, Tipo di Risorsa) che garantiscono una descrizione ricca, vitale per la *discovery* (reperibilità) e la citazione corretta. Inoltre, una caratteristica cruciale dello *schema* è la sua capacità di stabilire relazioni strutturate tra diversi oggetti digitali — quali dati, pubblicazioni e *software* — abilitando la creazione di un ecosistema di ricerca interconnesso e tracciabile definito come *PID Graph* [Cousijn et al., 2021]. Tecnicamente, questa interconnessione relazionale tra oggetti digitali è implementata tramite la proprietà *relatedIdentifier* [9] dello *schema*, la quale permette di dichiarare la natura esatta del collegamento (tramite la proprietà *relationType*) conferendogli un preciso significato semantico. Ad esempio, la relazione *IsSupplementTo* viene tipicamente impiegata per vincolare indissolubilmente un *dataset* sperimentale (o un codice sorgente) all'articolo scientifico che ne discute i risultati; allo stesso modo la proprietà *IsCitedBy* documenta quando la risorsa viene formalmente menzionata in studi successivi; infine, *IsCompiledBy* risulta fondamentale per la tracciabilità della provenienza (*data provenance*), permettendo di indicare con esattezza quale versione di un *software* o algoritmo ha generato o aggregato un determinato set di informazioni.

L'Istituto Nazionale di Geofisica e Vulcanologia (INGV) genera quotidianamente una vasta mole di dati eterogenei, che spaziano dal monitoraggio sismico e vulcanico alla modellazione geofisica. Catalogare, preservare e garantire l'adesione ai principi FAIR per questi prodotti è una priorità strategica.

Sebbene nella comunità scientifica siano diffusi *software* di “Research Data Management (RDM)” [10] come InvenioRDM [11], CKAN [12], DSpace [13] e Dataverse [14], questi presentano spesso architetture monolitiche complesse. Essi nascono primariamente come *data repository* completi per gestire l'intero ciclo di vita del dato (incluso lo *storage* dei file), piuttosto che come agili cataloghi di metadati. Lo stesso INGV - Osservatorio Etneo, utilizza un repository di tipo generalista [Pistagna et al., 2023] basato su InvenioRDM per l'archiviazione di *dataset* non catalogati in banche dati specialistiche [15].

Tuttavia, dall'utilizzo di moderni sistemi *software* che durante il loro *workflow* di esecuzione consultano in maniera automatizzata - *Machine-to-Machine* - cataloghi di dati e metadati, è emersa la necessità di una soluzione più “leggera” e specializzata, che fungesse esclusivamente da *backend* [16] per la catalogazione ed il reperimento dei metadati associati ad un *dataset*, con le seguenti caratteristiche distintive:

- **Agilità e Leggerezza:** Ridurre l'impronta *software* e la complessità infrastrutturale, evitando di replicare funzionalità di storage già esistenti altrove.
- **Architettura Headless:** Disaccoppiare completamente il *backend* dall'interfaccia utente (UI). Questo approccio garantisce la massima flessibilità, permettendo lo sviluppo futuro di diverse interfacce (web, mobile, M2M) senza modificare il nucleo di gestione dei dati.

Per colmare questa lacuna, è stato sviluppato MetaCat [Pistagna, 2025] [17]: un servizio per la catalogazione dei metadati delle produzioni scientifiche, rigorosamente conforme allo schema DataCite (v4.6) [18]. Questo rapporto tecnico ne descrive l'architettura, le scelte implementative e i risultati ottenuti attraverso un caso d'uso basato sui *dataset* istituzionali.

1. Contesto: il Patrimonio Dati dell'INGV

L'Istituto Nazionale di Geofisica e Vulcanologia (INGV) persegue attivamente una politica di produzione e disseminazione di dati aperti (*Open Data*) [Puglisi et al., 2018] [19], derivanti dalle proprie attività di ricerca istituzionale nei settori della sismologia, vulcanologia, geofisica

marina, ambientale e del monitoraggio del territorio. Tale impegno si pone in piena conformità con le politiche internazionali di Scienza Aperta (*Open Science*) e con i principi FAIR. L'obiettivo strategico non è limitarsi alla semplice disponibilità del dato, ma garantirne la reperibilità, l'accessibilità e, soprattutto, l'interoperabilità tra diverse piattaforme, prerequisito essenziale per un effettivo riutilizzo da parte della comunità scientifica e del pubblico.

Per rendere operativa questa filosofia, l'INGV ha istituito il Portale *Open Data* [INGV Data Management Office, 2020; Locati et al., 2026] [20], che funge da catalogo centrale e ufficiale dell'Ente. Ben più di un semplice archivio statico, il portale agisce come aggregatore per l'intera collezione di *dataset* istituzionali, centralizzando l'accesso e facilitando le operazioni di ricerca. Al momento della stesura di questo documento, il portale cataloga più di 700 *dataset*, suddivisi in macro-categorie tematiche che riflettono la multidisciplinarietà dell'Istituto.

La varietà dei dati gestiti evidenzia la complessità della sfida di metadatazione. Le principali categorie includono:

- **Dati Sismologici e Infrasonici:** Comprendono le registrazioni di eventi sismici, la sismicità storica (es. il Catalogo Parametrico dei Terremoti Italiani - CPTI [21] [Rovida et al., 2022]) e i dati grezzi acquisiti dalle Reti Sismiche Nazionali e locali, sia terrestri che marine. Questa categoria include le forme d'onda (*waveforms*), i metadati strumentali delle stazioni e le soluzioni di localizzazione ipocentrale.
- **Dati Vulcanologici:** Includono i flussi di dati provenienti dalle reti di monitoraggio dei vulcani attivi (in particolare Etna, Vesuvio e Campi Flegrei). I *dataset* spaziano dalle misurazioni geologiche e sismiche a quelle geodetiche e geochemiche relative ai fenomeni eruttivi e pre-eruttivi.
- **Dati Geodetici:** Riguardano le serie temporali relative alle deformazioni del suolo, acquisite tramite tecnologie GNSS/GPS e clinometria. Questi dati sono cruciali per la modellazione delle sorgenti vulcaniche e sismiche.
- **Dati Geochemici:** Raccolgono le analisi chimiche effettuate su matrici diverse (rocce, acque, gas, suoli) e campioni fisici, fondamentali per la comprensione dei processi di degassamento e della circolazione dei fluidi nel sistema Terra.
- **Dati Geofisici:** Una categoria ampia che include misurazioni di geomagnetismo, geoelettrica ed elettromagnetismo, acquisite sia presso osservatori permanenti che durante campagne di ricerca temporanee (inclusi dati da fondali marini).
- **Dati Geologici:** Includono informazioni strutturali e stratigrafiche del territorio italiano e delle aree marine, Modelli Digitali di Elevazione (DEM) e *database* specialistici come il *Database of Individual Seismogenic Sources* (DISS) [DISS Working Group, 2025].
- **Dati di Telerilevamento:** Comprendono *dataset* e immagini acquisiti da sensori remoti (satellitari, aerei o da drone) per l'analisi e il monitoraggio delle superfici terrestri e marine.

2. MetaCat: un Catalogo di Metadati

L'ideazione e lo sviluppo di MetaCat si inquadrano esplicitamente come un caso di studio in ambito di ricerca e sviluppo (*Research & Development*) e come dimostratore tecnologico. Il progetto nasce con l'obiettivo di esplorare architetture *software* moderne con il fine di realizzare un "Catalogo di Metadati" agile e leggero, concepito nativamente per l'interazione *Machine-to-Machine* grazie alla sua rigorosa filosofia *API-first*². Sotto questa prospettiva, MetaCat non si

² In ingegneria del *software*, l'approccio di sviluppo *API-first* è un paradigma lungimirante che privilegia la progettazione e la creazione di API (*Application Programming Interface*, cioè un insieme di procedure per far comunicare diversi *software* o componenti) all'inizio del ciclo di vita dello sviluppo *software*. A differenza dell'approccio tradizionale, in cui le applicazioni vengono create prima e le API vengono elaborate in un secondo momento, lo sviluppo *API-first* considera le API come i componenti fondamentali che dettano lo sviluppo del codice delle altre applicazioni.

pone in sovrapposizione all’attuale Portale *Open Data* INGV - le cui procedure logiche e flussi di validazione sono ampiamente rodati e collaudati negli anni - ma funge da ambiente sperimentale complementare. L’esperimento mira a fornire evidenze architettureali che, in futuro, potrebbero ispirare e guidare un consolidamento tecnologico dell’infrastruttura principale. Di conseguenza, il perimetro funzionale di MetaCat è stato circoscritto in modo rigoroso: il sistema opera esclusivamente come motore di *ingestion* e di *backend* per l’archiviazione di *record* di metadati descrittivi in stretta conformità allo schema DataCite, garantendo così la massima chiarezza semantica e interoperabilità dei dati catalogati.

2.1 Filosofia di Progettazione

MetaCat è stato concepito primariamente come un servizio di *backend*: la sua funzione essenziale è la gestione persistente e strutturata dei metadati associati ad un *dataset*, offrendo interfacce programmatiche (API) per la loro manipolazione. La visione strategica posiziona MetaCat come un componente infrastrutturale chiave, destinato a essere integrato in ecosistemi *software* più ampi (es. portali *web*, app mobili, *pipeline* di dati). Questo ruolo è cruciale per la promozione dei principi FAIR e per il sostegno alla riproducibilità scientifica, permettendo l’interazione automatizzata tra macchine (M2M) in *workflow* automatizzati che usano *pipeline* di dati.

I pilastri architetturali del progetto sono:

- **API-First**: Il nucleo del sistema è costituito da un insieme di API REST robuste e ben documentate, che espongono tutte le funzionalità del catalogo.
- **Cloud-Native**³ e portabilità: Un requisito imprescindibile era la capacità di *deployment* (§4.1) su qualsiasi infrastruttura, sia *on-premises* (su *server* locali, installati presso l’istituzione di appartenenza) che in *cloud* pubblico. Questo obiettivo è stato raggiunto attraverso la *containerizzazione* standardizzata dell’applicazione.
- **DevOps-ready**: Il progetto aderisce a un moderno ciclo di vita *DevOps* [22], includendo nativamente tutti gli artefatti necessari per processi riproducibili: *suite* di test automatici, gestione tramite versioni delle migrazioni del *database* e orchestrazione dei servizi.

Di seguito un tabulato (Tabella 1) che descrive schematicamente lo *stack* tecnologico usato per l’implementazione del catalogo MetaCat.

Categoria	Tecnologia	Ruolo e Motivazione
Backend Core	Node.js con Express.js	Node.js [23] fornisce un ambiente di esecuzione asincrono ideale per API ad alto traffico I/O. Express.js [24] è stato scelto come <i>framework web</i> minimalista per la creazione rapida e modulare delle rotte REST.
Database	MongoDB	<i>Database</i> documentale NoSQL selezionato per la sua flessibilità. La natura semi-strutturata di MongoDB [25] si adatta perfettamente alla struttura gerarchica con cui sono descritti i metadati di tipo DataCite.

³ *Cloud-Native* è un approccio moderno allo sviluppo di *software*, non significa solo spostare *app* nel *cloud*, ma costruirle per il *cloud*, sfruttando microservizi, *container* (come Docker) e orchestrazione (come Kubernetes) per creare applicazioni scalabili, resilienti e agili, rilasciando nuove funzionalità più velocemente, in modo affidabile e sfruttando al meglio l’elasticità dei *provider* di servizi *cloud*.

Data Modeling (ODM)	Mongoose	Mongoose [26] è un <i>Object Data Modeling</i> ⁴ che viene utilizzato per imporre uno schema rigoroso a livello applicativo sui dati in MongoDB. Gestisce la validazione dei tipi, le relazioni tra entità e semplifica le operazioni di <i>query</i> complesse.
Autenticazione	JWT e OAuth2 (ORCID)	Strategia ibrida: JWT [27] (<i>JSON Web Tokens</i>) gestisce l'autenticazione <i>stateless</i> delle sessioni API. OAuth2 [28] delega l'identificazione degli utenti a <i>provider</i> scientifici fidati, specificamente ORCID [29], standard <i>de-facto</i> per l'identità dei ricercatori.
Validazione Input	AJV (Another JSON Validator)	AJV [30] è un componente critico per l'integrità dei dati. Esegue una validazione ad alta velocità dei <i>payload</i> JSON [31] in ingresso contro gli schemi DataCite, respingendo dati malformati prima che raggiungano la logica di <i>business</i> .
Testing	Mocha & Chai	Mocha [32] e Chai [33], compongono una <i>suite</i> completa per il <i>Quality Assurance</i> , coprendo test unitari, funzionali e di integrazione. La libreria <i>chai-http</i> è utilizzata specificamente per verificare la conformità delle risposte HTTP dell'API.
Migrazioni	Umzug	Umzug [34] è lo strumento per la gestione programmatica dell'evoluzione dello schema <i>database</i> . Assicura che le modifiche alla struttura dei dati siano applicate in modo versionato e riproducibile tra ambienti di sviluppo e produzione.
Orchestrazione	Docker & Nginx	Docker [35] garantisce l'isolamento dell'ambiente di esecuzione. Docker Compose orchestra i servizi (<i>Backend</i> , DB, Proxy). Nginx [36] funge da <i>reverse proxy</i> per la gestione del traffico, il <i>load balancing</i> e la terminazione SSL/HTTPS.
Documentazione	Swagger / OpenAPI 3.0	L'adesione allo standard OpenAPI [37] e l'adozione di Swagger [38] permette la generazione automatica di documentazione interattiva e Client SDK, facilitando l'adozione e l'integrazione del <i>software</i> da parte di terzi.

Tabella 1 Stack Tecnologico di MetaCat.
Table 1 Metacat Technological Stack.

3. MetaCat: Architettura *software*

MetaCat è stato concepito adottando un'architettura *software* modulare e orientata alla scalabilità, fondata sui principi di disaccoppiamento e singola responsabilità - *Single Responsibility Principle* [39]. Questa scelta architetturale, organizzata in livelli ben definiti seguendo il pattern di *Layered Architecture* [40], assicura manutenibilità, testabilità e la capacità di far

⁴ Un ODM è un *software* che funge da ponte tra i concetti della programmazione orientata agli oggetti (OOP) e i *database* non relazionali (NoSQL), permettendo agli sviluppatori di manipolare i documenti del *database* come se fossero oggetti di codice nativo, semplificando la gestione dei dati, astruendo le *query* del database.

evolvere il sistema in modo incrementale, isolando la *business logic* dai dettagli implementativi infrastrutturali. L'applicazione rigorosa del principio di *Single Responsibility* ha guidato la definizione di ogni modulo del sistema: ogni componente (classe, funzione, *middleware*) è stato progettato per assolvere a un unico compito ben definito e circoscritto nel suo dominio di appartenenza, riducendo drasticamente l'accoppiamento (*coupling*) tra le parti e facendo sì che la modifica a un componente non generi effetti collaterali imprevisti su altri. Nel contesto di MetaCat, questo si traduce in una netta separazione operativa: il codice che gestisce le chiamate di rete è completamente distinto da quello che applica le regole di *business*, il quale a sua volta ignora i dettagli di connessione al *database*.

Parallelamente, l'adozione di una architettura a livelli (*Layered Architecture*) organizza questi componenti in strati logici orizzontali e gerarchici. Questo approccio agisce come una barriera protettiva per il "cuore" del sistema: la logica di business (*Business Logic*), che rappresenta il valore centrale dell'applicazione, viene isolata e protetta dalle variazioni dei dettagli implementativi infrastrutturali (come *driver* di accesso al *database* o *framework web*). Questa stratificazione garantisce non solo una manutenibilità a lungo termine, ma anche una testabilità granulare, permettendo di verificare il comportamento di ogni livello in isolamento. L'immagine di seguito (Figura 1) rappresenta graficamente l'adozione del principio di *Single Responsibility* dai componenti *software* e la loro suddivisione secondo il *pattern* di architettura a livelli (*layers*).

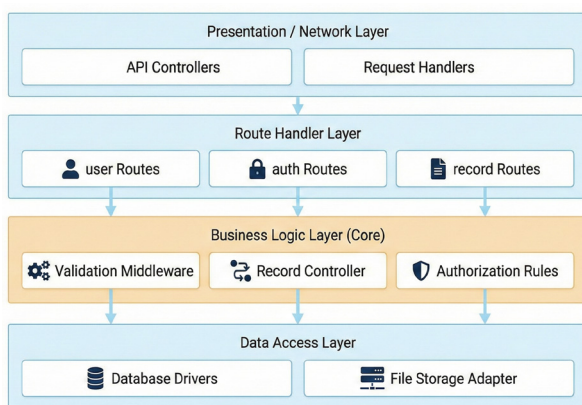


Figura 1 I principi di Singola Responsabilità ed il pattern di Architettura *Multi Layer* applicati nel contesto dello *stack software* di MetaCat. La divisione verticale dei diversi box colorati, rappresenta la divisione a strati (*layers*). All'interno dei livelli ogni componente assume una singola responsabilità (ruolo).

Figure 1 The principles of *Single Responsibility* and the *Multi-Layer Architecture* pattern applied in the context of the MetaCat software stack. The vertical division of the different colored boxes represents the division into layers. Within the layers, each component assumes a single responsibility (role).

3.1 Livello di Validazione e Persistenza: la "Doppia Difesa"

La gestione dei dati in MetaCat si basa su un doppio meccanismo di validazione sequenziale, progettato per garantire l'integrità e la conformità di ogni entità, lungo tutto il percorso del dato, dall'ingresso nel sistema tramite API fino alla persistenza fisica nel *database* (Figura 2).

3.1.1 Validazione Esterna (AJV: Il *Gatekeeper* dell'API)

Il primo livello di controllo è situato alla frontiera del sistema. Prima che qualsiasi *payload* JSON in ingresso (operazioni POST, PUT e PATCH) raggiunga la logica applicativa, viene intercettato e validato contro un insieme rigoroso di schemi. Questa fase è gestita dalla libreria **AJV** (**A**nother **J**SON **S**chema **V**alidator), integrata nell'architettura tramite un *middleware* personalizzato [41] di Express.js (`src/v1/services/ajvService.js`).

Tecnicamente, questo *middleware* agisce interponendosi nel ciclo di vita della richiesta HTTP: compila gli schemi JSON (che mappano fedelmente il DataCite Metadata Schema v4.6) in funzioni JavaScript ad alta efficienza e verifica il corpo della richiesta in ingresso. Questo

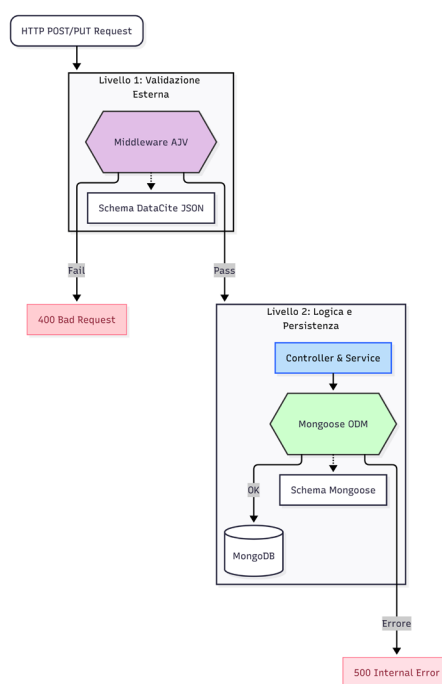
componente funge da *gatekeeper* ad alte prestazioni: adottando il paradigma “Fail Fast”, respinge cioè immediatamente le richieste malformate o sintatticamente non conformi. Ciò comporta un significativo beneficio architetturale, poiché riduce il carico computazionale sui livelli interni e protegge il *database* da “sporcizia” sintattica ancor prima che venga istanziata la logica di *business*.

3.1.2 Validazione Interna e Modellazione (Mongoose)

Superata la barriera esterna, i dati accedono al livello di dominio gestito tramite **Mongoose**, un ODM (*Object Data Mapper*) per MongoDB. Se AJV garantisce la correttezza della forma (sintassi), Mongoose garantisce la coerenza della struttura e delle relazioni (semantica). In questo livello, il *software* non interagisce più con JSON grezzi, ma con “Modelli” strutturati. Mongoose funge da ponte tra l'ambiente di *runtime* Node.js e il *database* documentale, imponendo una tipizzazione forte (es. assicurando che una data sia un oggetto *Date* valido e non una stringa arbitraria) e astruendo le complessità delle operazioni di I/O. Aspetto cruciale di questo livello è la gestione dell'integrità relazionale: attraverso gli schemi definiti (*UserSchema*, *RecordSchema*, *RecordMetadataSchema*), il sistema impone vincoli logici, come la relazione di proprietà (*owner*) che lega indissolubilmente un *record* all'utente che lo ha creato. Questa modellazione è fondamentale per supportare le logiche di sicurezza e autorizzazione descritte nei paragrafi successivi.

Figura 2 Illustra il diagramma di flusso della validazione a doppio livello implementata: le richieste in arrivo (POST, PUT) vengono inizialmente sottoposte a validazione tramite il *middleware* AJV, successivamente, il metadata viene validato sullo schema Mongoose a livello di *database*.

Figure 2 Illustrates the flow chart of the implemented two-level validation: incoming requests (POST, PUT) are initially subjected to validation via the AJV middleware, subsequently, the metadata is validated against the Mongoose schema at the database level.



3.2 Livello Logico: il *Pattern Controller-Service-DAO*

Per gestire la complessità crescente della logica applicativa, il core di MetaCat adotta rigorosamente il *pattern* architetturale a tre livelli *Controller-Service-DAO*. Questa organizzazione impone una netta “Separazione delle Preoccupazioni” (*Separation of Concerns- SoC*⁵), assicurando che ogni livello operi su un dominio di responsabilità isolato (Figura 3). Il risultato è un flusso dei dati unidirezionale e prevedibile, che facilita il *testing* isolato dei componenti e la manutenibilità del codice nel tempo.

⁵ La “Separazione delle Preoccupazioni” (*Separation of Concerns, SoC*) è un principio di progettazione in informatica che sostiene la divisione di un programma in sezioni distinte, ognuna con un compito specifico e indipendente, al fine di rendere il codice più modulare, manutenibile, testabile e riutilizzabile, separando logica di business, interfaccia utente, accesso ai dati.

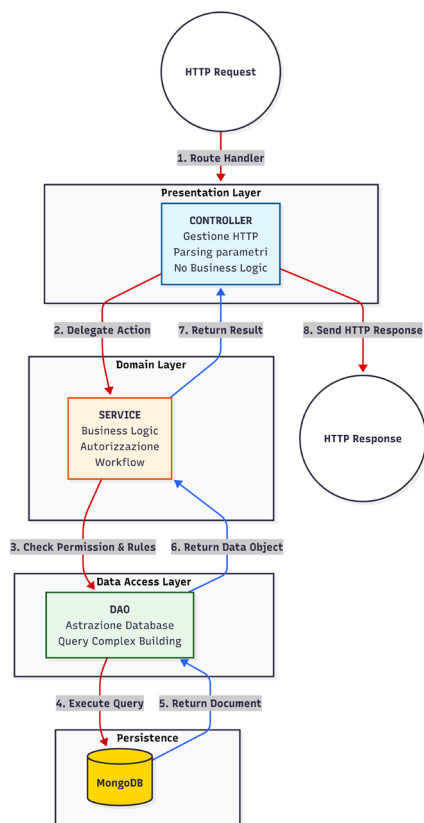


Figura 3 Illustra il diagramma di flusso del *pattern* architetturale *Controller-Service-DAO*: 1. la richiesta HTTP viene inoltrata dal gestore delle rotte al *controller*, il quale 2) delega il *service* appropriato; successivamente 3) dopo aver verificato le necessarie verifiche di sicurezza sulla richiesta, il controllo passa al DAO che 4) esegue le effettive *query* sul DB. La catena inversa vede come l'esito dell'operazione sul DB venga 5) restituita al DAO che la incapsulerà 6) in un oggetto di risposta per il *service* che a sua volta 7) la tornerà al *controller*, che infine la userà per generare 8) la *response* HTTP.

Figure 3 Illustrates the flow diagram of the *Controller-Service-DAO* architectural pattern: 1. the HTTP request is forwarded by the route handler to the controller, which 2) delegates to the appropriate service; subsequently 3) after performing the necessary security checks on the request, control passes to the DAO which 4) executes the actual queries on the DB. The reverse chain sees how the result of the operation on the DB is 5) returned to the DAO which 6) encapsulates it in a response object for the service, which in turn 7) returns it to the controller, which finally uses it to generate 8) the HTTP response.

3.2.1 Controller: Gestione del Protocollo HTTP

Il *Controller* (implementato in moduli come `src/v1/controllers/[auth Controller.js, recordController.js, userController.js]`) rappresenta il punto di ingresso e di uscita dell'applicazione, fungendo da interfaccia diretta con il *client*. La sua responsabilità è strettamente limitata alla gestione del protocollo di trasporto (in questo caso HTTP). Nello specifico, il *Controller* riceve l'oggetto della richiesta (*req*), ne estrae i parametri (*body*, *query params*, *path variables*) e li "pulisce" prima di passarli agli strati sottostanti. Agendo come un puro orchestratore, delega l'esecuzione effettiva delle operazioni al *Service Layer* e attende il risultato. Una volta ottenuta l'elaborazione, il *Controller* si occupa di formattare la risposta (*res*), selezionando il codice di stato HTTP semanticamente corretto (es. 200 OK, 201 Created, 404 Not Found) e serializzando i dati in JSON per il *client*. È fondamentale notare che questo livello è privo di qualsiasi logica di *business*: un *Controller* non "decide" mai, si limita a tradurre le richieste e le risposte (*req*, *res*) che arrivano tramite protocollo HTTP nel dominio dell'applicazione.

3.2.2 Service: Logica di Business e Dominio

Il *Service Layer* (`src/v1/services/[authService.js, recordService.js]`) costituisce il cuore nevralgico del sistema, dove risiede l'intelligenza applicativa. A differenza del *Controller*, il *Service* è completamente agnostico rispetto al protocollo HTTP (non conosce *req* o *res*), ricevendo in *input* semplici oggetti JavaScript. In questo livello vengono implementate le regole operative del catalogo di metadati, come i complessi *workflows* di pubblicazione (che richiedono validazioni incrociate) o le logiche di ricerca avanzata. Inoltre, il *Service* è il guardiano della sicurezza a livello logico: è qui che avvengono i controlli di autorizzazione granulare, verificando ad esempio se l'utente che sta tentando un'azione possiede i privilegi necessari (ruolo `admin`

o proprietà del *record*) prima di procedere. Questa indipendenza dal protocollo HTTP rende la logica di *business* estremamente testabile e potenzialmente riutilizzabile da interfacce diverse (es. una *CLI* o un *consumer* di messaggi) senza modifiche.

3.2.3 DAO (*Data Access Object*): Astrazione dello *Storage*

Il livello più profondo è rappresentato dai *DAO* (*Data Access Objects* [42]), responsabili dell'interazione fisica con il *database*. Il compito del *DAO* è disaccoppiare la logica di *business* dalla tecnologia di persistenza sottostante (MongoDB). Questo livello traduce le intenzioni di alto livello del *Service* (es. "trova tutti i *record* pubblici dell'utente X") nel dialetto specifico del *database*, costruendo *query*, aggregazioni e proiezioni ottimizzate.

La scelta di adottare MongoDB per l'intero strato di persistenza rappresenta un preciso compromesso progettuale. Da un lato, l'impiego di un *database* NoSQL *document-oriented* è la soluzione ingegneristica ottimale per accogliere la natura nativamente gerarchica, e profondamente annidata dello standard DataCite, evitando la frammentazione del dato in decine di tabelle e garantendo un'altissima efficienza in fase di *ingestion*. Dall'altro lato, il sistema presenta intrinseche necessità relazionali, quali la gestione dell'identità (utenti), dei ruoli (RBAC) e della paternità (*ownership*) dei record. Sebbene MongoDB non sia nativamente progettato per architetture fortemente relazionali, tali vincoli vengono risolti efficacemente a livello applicativo tramite il *framework* Mongoose e incapsulati nel *DAO*, che gestisce le relazioni simulando le *join* attraverso l'operatore di aggregazione *\$lookup* [43]. Con questa operazione vengono combinati i documenti tra due collezioni (*collections*) filtrandoli in base a una corrispondenza su un campo specifico.

Essendo le *query* relazionali di MetaCat limitate principalmente al controllo degli accessi, l'impatto prestazionale di tali aggregazioni risulta trascurabile. Tale asserzione è supportata empiricamente dai risultati della campagna di *benchmark* (§4.3), che certificano l'elevata tolleranza del sistema sotto stress e la sua capacità di mantenere latenze contenute. Inoltre, il disaccoppiamento garantito dal *pattern* *DAO* assicura una totale sostenibilità evolutiva: qualora in futuro le logiche relazionali dovessero ramificarsi, l'infrastruttura sarebbe già predisposta per migrare verso un approccio di persistenza ibrido (ad esempio, delegando la gestione utenti ad un *database* relazionale e mantenendo un *database* di tipo NoSQL per i metadati), richiedendo unicamente la riscrittura dei metodi esposti dal *DAO*, senza alcun impatto sui *layer* superiori (§3).

3.3 Livello API: *Endpoint* pragmatici *REST-like*

L'interfaccia pubblica di MetaCat è costituita da un insieme di *endpoint* progettati secondo un approccio pragmatico REST. Questi *endpoints* non si limitano a esporre le operazioni CRUD (*Create, Read, Update, Delete*) sul *database*, ma riflettono semanticamente il ciclo di vita dei metadati e i processi di gestione dell'utente. Le definizioni delle rotte sono organizzate modularmente nella directory `'src/v1/routes/'`, facilitando la navigazione nel codice e la separazione dei domini funzionali. Di seguito un elenco delle funzionalità implementate dalle API REST.

3.3.1. Autenticazione ibrida (*authRoutes.js*)

Questo modulo gestisce l'ingresso nel sistema. Oltre alle rotte standard per la gestione delle utenze locali - `POST /auth/register`, `POST /auth/login` - il sistema implementa un

flusso OAuth2 - GET /auth/orcid - permettendo l'accesso federato tramite identificativo ORCID. In particolare, il flusso che descrive l'inserimento di una nuova identità nel sistema (operazione di registrazione o *sign-up*) è descritto visivamente nel diagramma di sequenza per la registrazione di un nuovo utente (Figura 4).

L'utente invia 1) una richiesta POST con le credenziali (*username, email, password*). Il *payload* viene intercettato 2) dal *middleware* di validazione (AJV) che ne conferma la struttura e i requisiti obbligatori 3). Il controllo passa 4) al *Controller* che delega 5) al *Service* la logica di *business*, inclusa la cifratura (*hashing*) della *password* per sicurezza. Il *DAO* viene invocato 6) per l'accesso ai dati ed esegue 7) l'inserimento fisico nel *database* MongoDB. A seguito della conferma di scrittura 8), l'oggetto utente creato (privato dei dati sensibili) risale la catena architetturale 9) 10) portando un *DTO* (*Data Transfer Object*) fino al *Controller*, che 11) restituisce al client un codice HTTP 201 (*Created*) tramite il 12) *RouteHandler*.

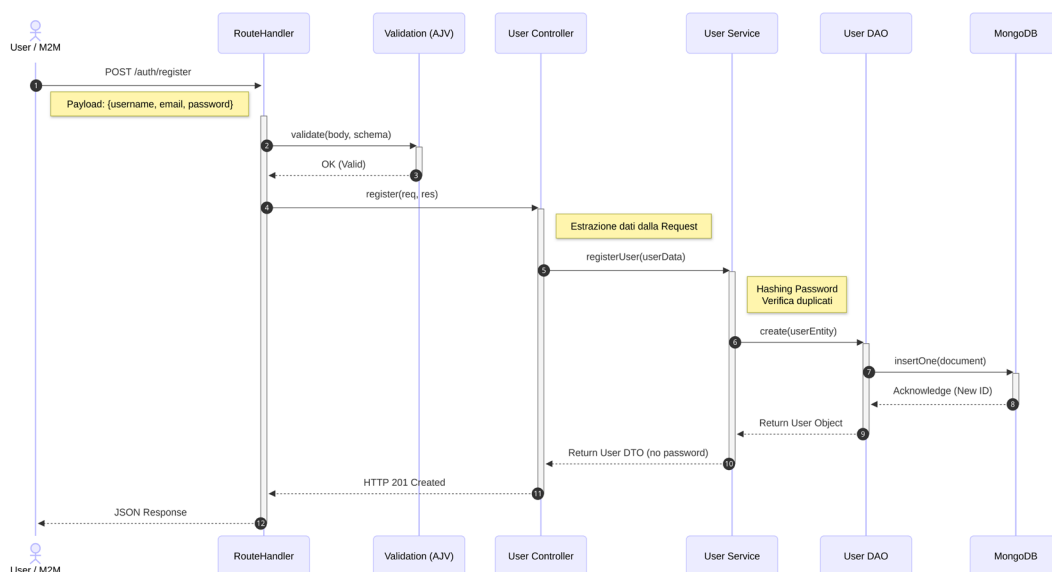


Figura 4 Diagramma di sequenza per la registrazione utente. Il flusso illustra l'interazione tra i layer architetturali (*Middleware, Controller, Service, DAO*), dalla validazione iniziale del *payload* (AJV) fino alla persistenza su MongoDB.

Figure 4 Sequence diagram for user registration. The flow illustrates the interaction between architectural layers (*Middleware, Controller, Service, DAO*), from the initial *payload* validation (AJV) to MongoDB persistence.

Allo stesso modo nel diagramma di sequenza del processo di autenticazione (Figura 5) è visivamente illustrato il flusso per la negoziazione dell'accesso sicuro (operazione di *login*). L'utente invia 1) le credenziali (*email e password*) all'*endpoint* di *login*. La richiesta viene immediatamente sottoposta 2) alla validazione sintattica (AJV) che ne conferma la correttezza formale 3). Il controllo passa 4) al *Controller* che invoca 5) il *Service* per la verifica sostanziale. Il *Service* recupera i dati dell'utente tramite il *DAO* 6) interagendo con il *database* 7) 8) ed esegue il confronto crittografico dell'*hash* della *password*. A verifica avvenuta con successo, il flusso torna 9) al *Controller* che esegue l'operazione critica: 10) la generazione e firma digitale del *Token JWT* (*JSON Web Token*). Infine, il *token* viene incapsulato 11) nella risposta HTTP 200 (OK) restituita al client 12) dal *RouteHandler*.

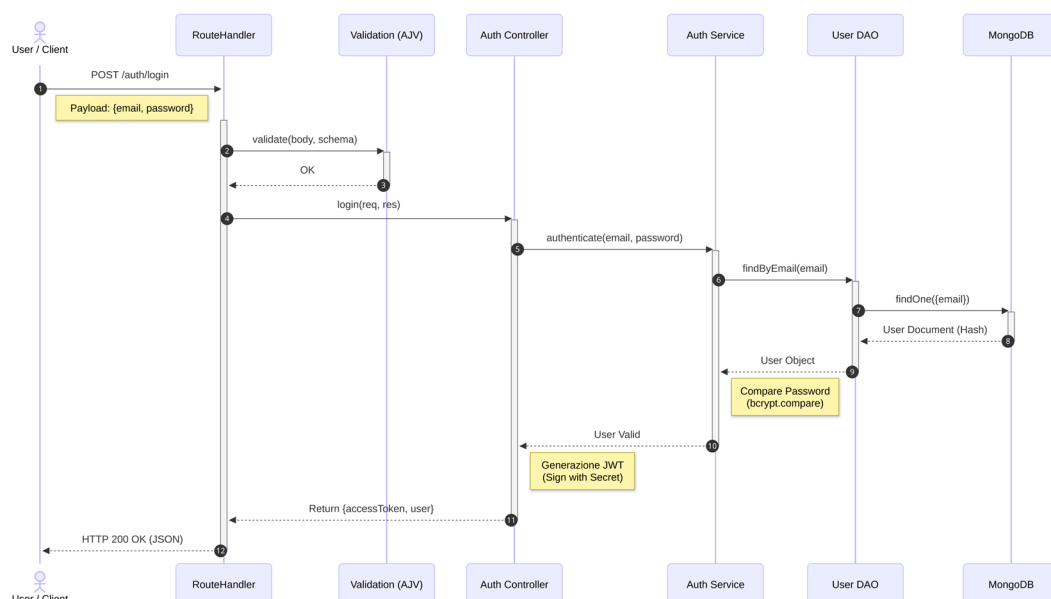


Figura 5 Diagramma di sequenza dell'autenticazione. Il flusso illustra l'interazione tra i livelli architetturali per la validazione delle credenziali, la verifica crittografica del database e l'emissione del token JWT.

Figure 5 Authentication sequence diagram. The flow illustrates the interaction between architectural layers for credential validation, cryptographic database verification, and JWT token issuance.

La scelta di limitare l'attuale modello di autenticazione a un approccio ibrido basato su utenze locali e OAuth2 tramite ORCID va inquadrata nella natura di dimostratore tecnologico - *Proof of Concept* - di MetaCat. In questa fase sperimentale, l'integrazione di ORCID non ha la pretesa di coprire in modo esaustivo gli scenari di adozione istituzionale, ma funge piuttosto da implementazione di riferimento per validare la robustezza, la flessibilità e l'interoperabilità della pipeline di sicurezza. Essendo l'infrastruttura di autenticazione concepita in modo rigorosamente modulare, il sistema è già nativamente predisposto per essere esteso in futuro ad altri provider di identità. Ciò include meccanismi di *Single Sign-On*⁶ e autenticazione federata ampiamente utilizzati nel contesto universitario e della ricerca, nonché l'integrazione diretta con Google Workspace⁷, attualmente alla base dell'operatività quotidiana e degli account istituzionali dell'INGV. L'eventuale implementazione futura di questi sistemi richiederebbe unicamente l'aggiunta delle relative logiche nei *middleware*, *controller* e *service* di autenticazione, lasciando del tutto inalterata la successiva logica di *business* e di gestione autorizzativa (RBAC) del nucleo applicativo.

3.3.2 Ciclo di vita del Record (*recordRoutes.js*)

È il core funzionale del catalogo. Le operazioni esposte dalle API implementate, mappano gli stati del *record* e ne governano l'intera evoluzione, dalla fase di inserimento fino alla pubblicazione definitiva o alla rimozione. Questo modulo, definito in `src/v1/routes/recordRoutes.js`, implementa le rotte principali per la creazione di nuove bozze tramite `POST`, l'aggiornamento integrale con `PUT`, e la modifica granulare di singoli attributi attraverso il verbo `PATCH`.

⁶ *Single Sign-On* è la proprietà di un sistema di controllo d'accesso che consente ad un utente di effettuare un'unica autenticazione valida per più sistemi software o risorse informatiche alle quali è abilitato.

⁷ Google Workspace è una suite software di produttività personale comprendente strumenti per il cloud computing e per la collaborazione.

3.3.2.1 Creazione Bozza: POST /records

L'invocazione tramite verbo `POST` dell'*endpoint* `/records`, inizializza un nuovo *record* in stato di *draft* (bozza), permettendo l'inserimento dei dati. Nel diagramma di sequenza in Figura 6, viene illustrata la *pipeline* dei *middleware* in azione: 1) la richiesta HTTP `POST` all'*endpoint* di creazione (con *payload* incluso) attraversa prima il livello di sicurezza, con 2) verifica del *token* e 3) iniezione dell'utente nella proprietà `req.user`. Successivamente, 4) e 5) avviene la validazione dello *schema* (AJV), per giungere infine 6) al *Controller*. Questo elabora il *payload* e invoca 7) il *Service* dedicato, che 8) inizializza la risorsa come bozza (*draft*) assegnandone la proprietà (*owner*). Inizia la fase di persistenza: il *DAO* gestisce la transazione verso MongoDB inviando i dati 9) e trasformando l'oggetto di dominio in un documento *BSON* permanente. La catena di esecuzione risale 10) 11) 12) con il *DTO* (*Data Transfer Object*) di response fino al *Controller*, che 13) compone la risposta di successo con codice `HTTP 201 (Created)` restituita infine 14) al *client* dal *RouteHandler*.

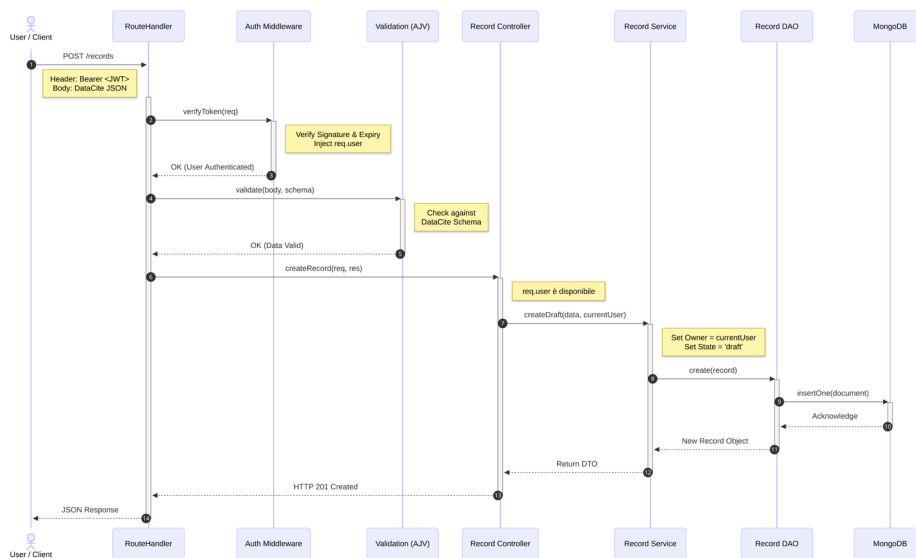


Figura 6 Diagramma di sequenza: creazione di un nuovo *record*. Il flusso illustra l'attraversamento della *pipeline* dei *middleware* (autenticazione JWT e validazione AJV), l'inizializzazione della risorsa in formato bozza e la definitiva persistenza su MongoDB e risposta `HTTP 201`.

Figure 6 Sequence diagram: new record creation. The flow illustrates the *middleware pipeline* traversal (JWT authentication and AJV validation), *draft resource* initialization, and final MongoDB persistence and `HTTP 201` response.

3.3.2.2 Modifica e aggiornamento *record*

L'utilizzo del verbo `PUT` verso l'*endpoint* `/records/{recordId}` consente l'evoluzione del contenuto e quindi dei metadati registrati. È fondamentale notare che ogni richiesta di aggiornamento viene intercettata nuovamente dal *middleware* di validazione (AJV), garantendo che il *record* non possa mai trovarsi in uno stato invalido, nemmeno durante le modifiche parziali.

3.3.2.3 Modifica di un attributo

L'invocazione tramite verbo `PATCH` verso l'*endpoint* `/records/{recordId}/{attribute}` aggiorna un singolo attributo di un *record*. Permette di modificare un singolo attributo (es.

creators, titles) di un *record* specifico. Un *user* può modificare solo i propri *record* e solo se non sono ancora pubblicati, *curator* e *admin* possono modificare qualsiasi *record*. Il diagramma di sequenza in Figura 7 evidenzia la rigorosa *pipeline* di sicurezza a tre livelli: 1) la richiesta raggiunge il *Router* e viene intercettata prima 2) dal *middleware* di Autenticazione che valida il *token* e 3) identifica l'utente. Successivamente, 4) interviene il *middleware* di Autorizzazione (RBAC) che applica la logica *CheckOwnershipOrRole*, 5) 6) verificando che l'utente sia l'effettivo proprietario della risorsa o un amministratore. Solo se autorizzato, il *payload* parziale passa 7) 8) alla validazione schema (AJV). Il flusso prosegue 9) verso il *Controller* e 10) il *Service*, che applica 11) le modifiche ai soli campi specificati. La persistenza è gestita 12) dal *DAO* che esegue l'aggiornamento atomico su MongoDB. Il documento aggiornato risale la catena 13) 14) 15) portando il *DTO* (*Data Transfer Object*) fino alla 16) composizione della risposta *HTTP 200* (OK) inviata 17) al *client*.

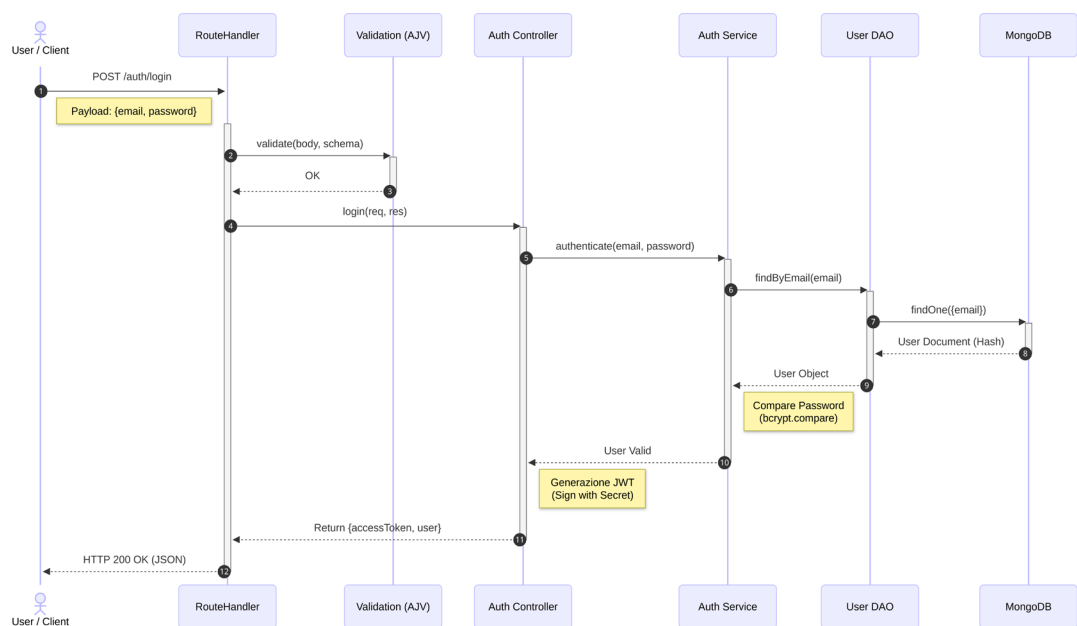


Figura 7 Diagramma di sequenza: aggiornamento parziale di un *record*. Rappresentazione dei tre livelli di sicurezza preventiva (Autenticazione, RBAC, Validazione) e del conseguente aggiornamento atomico su MongoDB.

Figure 7 Sequence diagram: partial record update. Representation of the three preventive security levels (Authentication, RBAC, Validation) and the subsequent atomic update on MongoDB.

3.3.2.4 Altre operazioni

Di seguito una lista delle altre operazioni implementate:

- **Consultazione:** GET /records e GET /records/{recordId} permettono il recupero dei metadati pubblici. L'*endpoint* puntuale tramite la specifica di un id implementa la restituzione di uno specifico *record*; una logica condizionale (definita nel *Controller*) gestisce la risposta con i dettagli privati del *record* in stato *draft* solo se il richiedente è il proprietario.
- **Pubblicazione:** POST /records/{recordId}/publish rappresenta una transizione di stato critica. Accessibile solo a ruoli privilegiati (*admin*, *curator*), innesca il congelamento della versione corrente e la sua pubblicazione sul catalogo.

- **Gestione personale:** La rotta `GET /me/records` offre una vista filtrata sulle risorse dell'utente autenticato, includendo le bozze non visibili pubblicamente.
- **Cancellazione di un record:** La rotta `DELETE /records/{recordId}` offre un *endpoint* amministrativo per la rimozione logica o fisica.

L'insieme di API sono documentate secondo lo standard OpenAPI e tramite l'interfaccia SwaggerUI possono essere consultate seguendo la rotta `<MetaCat_server_url>/api-docs/`.

3.4 Sicurezza: Pipeline di Middleware

La sicurezza in MetaCat non è implementata come logica sparsa nei *Controller*, ma attraverso una *pipeline* di *middleware* intercettori (`src/v1/middlewares/`). Questo approccio assicura che le *policy* di sicurezza siano applicate uniformemente prima che la richiesta raggiunga la logica di *business*.

3.4.1 Autenticazione (*authenticationMiddleware.js*)

Il primo livello di difesa verifica l'identità digitale dell'attore che effettua una *request* HTTP. Il *middleware* intercetta ogni richiesta protetta, estraendo e validando la firma crittografica e la scadenza del *JSON Web Token (JWT)* presente nella sezione *header Authorization*. In caso di successo, il *middleware* estende l'oggetto della richiesta iniettando l'istanza aggiornata dell'utente (`req.user`), rendendo il contesto di sicurezza (ID, ruolo, email) immediatamente disponibile ai livelli successivi secondo la logica di funzionamento dei *middleware*.

3.4.2 Autorizzazione (*authorizationMiddleware.js*)

Il secondo strato gestisce le autorizzazioni (permessi) e stabilisce quali operazioni un utente autenticato può effettivamente eseguire. Il modello di sicurezza di MetaCat implementa una gerarchia dei ruoli a tre livelli (implementando una politica RBAC⁸), progettata per rispecchiare il flusso di lavoro di validazione scientifica:

- **user** (Ricercatore, utente, processo macchina): È il livello di accesso base. L'utente possiede i diritti di creazione (*create*) e modifica (*update*) limitatamente alle risorse di sua proprietà (*ownership*). Può gestire i propri *record* in stato di bozza ma non ha i privilegi per renderli pubblici in autonomia.
- **curator** (*Data Steward*): Rappresenta la figura garante della qualità del dato. Oltre ai permessi base, possiede il privilegio elevato di transizione di stato: è l'unico ruolo (insieme all'*admin*) abilitato all'*endpoint* di pubblicazione (*publish*), sancendo il passaggio da bozza privata a record pubblicamente accessibile.
- **admin** (Amministratore di Sistema): Possiede privilegi globali su tutte le risorse. È l'unico ruolo abilitato all'esecuzione di operazioni distruttive (*delete*) e alla gestione dell'infrastruttura, garantendo la manutenzione del catalogo.

⁸ RBAC è l'acronimo di *Role-Based Access Control*, Controllo degli Accessi Basato sui Ruoli, un modello di sicurezza informatica che gestisce l'accesso alle risorse di un sistema (dati, applicazioni) assegnando permessi non direttamente agli utenti, ma a ruoli specifici (es. "Amministratore", "Utente"), e poi associando gli utenti a questi ruoli.

Il controllo dei permessi avviene durante la *pipeline* di autorizzazione tramite l'uso di due *middlewares*:

- Middleware **checkRole**: Protegge le rotte sensibili (es. cancellazione) verificando staticamente l'appartenenza dell'utente a uno dei gruppi privilegiati.
- Middleware **checkOwnershipOrRole**: Implementa una logica ibrida e dinamica, consentendo la modifica di una risorsa solo se l'utente ne è il proprietario oppure se possiede un ruolo amministrativo. Questo meccanismo garantisce che i ricercatori possano alterare solo i propri dati, preservando l'integrità del lavoro altrui.

4. Risultati: *deployment* e validazione

Dopo aver definito i principi architetturali e lo *stack* tecnologico, questo capitolo illustra l'implementazione del sistema MetaCat. La trattazione si articola in due macro-sezioni:

- In prima istanza verrà descritto il modello di *deployment*⁹ adottato per garantire la portabilità e l'isolamento del *software*.
- Successivamente, verrà analizzato il caso d'uso reale scelto per validare le funzionalità e le prestazioni dell'infrastruttura.

4.1 Modello di *deployment*

Per garantire la riproducibilità degli ambienti di esecuzione e contestualmente l'isolamento dei processi, il modello di *deployment* di MetaCat è basato interamente sul paradigma dei *software container* tramite Docker. L'orchestrazione dello *stack* applicativo è demandata a **Docker Compose** [44], che permette di definire l'infrastruttura come codice - *Infrastructure as Code* [45] - tramite un singolo *file* manifesto nel formato di serializzazione di dati YAML [46].

L'architettura di *runtime* è stata concepita come un ecosistema di tre servizi interconnessi, ciascuno con una responsabilità specifica:

- Servizio **mongodb**: preposto alla persistenza dei dati tramite un *database* di tipo NoSQL. È configurato con un volume persistente (`docker volume` [47]) esterno al ciclo di vita del *container*, garantendo la durabilità e l'integrità delle informazioni anche in caso di riavvio o aggiornamento del servizio.
- Servizio **metacat**: Costituisce il nucleo applicativo del sistema, ossia un sistema di *backend* implementato su un *runtime* con Node.js. L'immagine del *container* è generata attraverso una operazione di *multi-stage build* [48] ottimizzata, che riduce la dimensione finale dell'immagine stessa e separa le dipendenze di compilazione da quelle di *runtime*.
- Servizio **nginx**: Agisce come *Reverse Proxy* e *API Gateway*. Rappresenta l'unico punto di accesso esposto verso l'esterno (porte TCP 80/443), gestendo la terminazione SSL/TLS per la cifratura del traffico HTTPS e l'instradamento delle richieste verso il *backend*, che rimane protetto nella rete interna virtuale (`docker network` [49]).

4.2 Caso d'uso: *Ingestion* del Catalogo Metadati INGV

La validazione del sistema MetaCat non si è limitata alla verifica della correttezza della logica applicativa tramite test unitari (il *framework* mocha e la libreria chai, vedi Tabella 1). Per attestare la reale efficacia e robustezza dell'architettura, è stata eseguita una "campagna di

⁹ Il "*deployment*" del software comprende tutte le attività che rendono un sistema *software* disponibile per l'utilizzo: dalla configurazione dei vari componenti, all'installazione su una infrastruttura, fino ai test funzionali.

test" funzionali e prestazionali basata su un caso d'uso reale: l'ingestione (caricamento) massiva - **data ingestion** - del catalogo dei metadati associati ai *dataset* prodotti del INGV. Questo scenario, che impiega l'intero patrimonio informativo attualmente esposto dal Portale *Open Data* dell'Istituto, ha permesso di verificare simultaneamente:

1. Correttezza Funzionale: La capacità del sistema di validare e persistere *record* di metadati reali e complessi.
2. Resilienza sotto Carico: Il comportamento delle API quando sottoposte ad un carico non indifferente di richieste concorrenti (*stress test*).

Per orchestrare ed eseguire questa campagna di *benchmark* è stato selezionato k6 di Grafana [50], un *framework open-source* moderno per i test di carico (*load test*) e la misurazione delle prestazioni. La scelta di k6 è stata dettata dalla sua natura *developer-centric*: esso permette di definire scenari di test complessi scrivendo *script* in JavaScript, mantenendo così una totale coerenza con lo *stack* tecnologico del progetto. Inoltre, grazie al suo motore ad altissime prestazioni scritto nel linguaggio Go, k6 è in grado di simulare un'elevata concorrenza di utenti virtuali (*Virtual Users*) minimizzando l'impatto sulle risorse della macchina *host* dove è eseguito, garantendo così misurazioni affidabili sulle latenze e sul *throughput* del sistema sotto test.

4.2.1 Data Preparation e Strategia di recupero Metadati

In prima istanza, è stata formalizzata una fase preliminare di *Data Preparation* (Preparazione dei dati), necessaria al fine di automatizzare l'estrazione e la formattazione dei *record* di metadati (alla data di stesura di questo documento, relativi a più di 700 *dataset*) selezionati dal Portale *Open Data* istituzionale. Questa fase ha previsto un processo in due passaggi fondamentali:

1. **Recupero dei DOI e Normalizzazione input:** Interrogando le API del Portale *Open Data* Istituzionale (basato sul *software* CKAN) [51], sono stati recuperati tutti i DOI relativi ai *dataset* pubblicati, selezionati per comporre il *dataset* di input. Tali identificativi sono stati memorizzati in un file CSV strutturato con un *header* contenente i campi `ID`, `DOI` e `check` (rispettivamente: l'`ID` del *record* nel Portale *Open Data*, il DOI univoco associato al *dataset* e un *flag booleano* per tracciare lo stato di processamento di *ingestion*).
2. **Recupero dei Metadati (DOI Content Negotiation):** Per l'effettivo recupero dei metadati nello standard DataCite 4.6, è stato sviluppato uno script in linguaggio Javascript - `fetchDOI.js` [52], che interroga il risolutore ufficiale di DOI della DOI Foundation [53]. L'obiettivo di questo strumento è stato quello di recuperare i metadati originali e successivamente aggregarli in un formato ottimizzato per l'operazione di *ingestion* massiva da parte degli *script* di test (*framework* k6). Invece di ricorrere a tecniche di *web scraping* sulle *landing page* dei *dataset* (non strutturate), è stata implementata una strategia basata sulla *Content Negotiation* [54]. Il protocollo HTTP consente al *client* di negoziare il formato della risorsa tramite un *header* HTTP `Accept`. Normalmente, la risoluzione di un DOI reindirizza il *browser* verso una pagina HTML (`text/html`). Il meccanismo di richiesta implementato nello script `fetchDOI.js` invece, interroga il risolutore globale (`https://doi.org/<DOI>`) specificando l'*header*: `'Accept: application/vnd.datacite.datacite+json'`. Questa direttiva istruisce il risolutore a restituire il *payload* dei metadati in formato JSON, già validato secondo lo schema DataCite. Questo approccio garantisce l'acquisizione di dati ad alta fedeltà direttamente dalla fonte autoritativa, eliminando errori di *parsing* e garantendo la compatibilità nativa con gli schemi di validazione di MetaCat.

A conclusione del processo lo *script* ha provveduto sia a scaricare, normalizzare e archiviare i singoli *payload* in formato JSON, che unirli in un unico *array* JSON consolidato che sarà l'*input*

dello script K6. Questo *dataset* finale, del peso di circa 11 MB, rappresenta una fotografia esatta del catalogo di metadati registrati sul Portale Open Data istituzionale - comprensiva sia di *record* leggeri che di *payload* complessi superiori ai 500 KB - fornendo così una base di test estremamente realistica per la fase di *benchmark*.

4.2.2 Obiettivi e Metodologia del *Benchmark*

L'obiettivo primario della campagna dei test di validazione è stato determinare i limiti operativi, la stabilità architetturale e la scalabilità di MetaCat affrontando uno scenario basato su dati reali. Come già accennato, al fine di garantire la massima attendibilità dei risultati, è stato scelto deliberatamente di non utilizzare dati sintetici, bensì di sottoporre il carico del sistema all'intero *dataset* dei dati reali ricavato in fase di preparazione (§4.2.1), che riflette la reale eterogeneità del catalogo di metadati INGV.

La metodologia di esecuzione dei *benchmarks* è stata descritta da due distinti *scripts* [52], gestiti tramite specifici esecutori (*executors*) del *framework* k6¹⁰:

- **Load Test** (Test di Carico - *Baseline*): Finalizzato a stabilire una *performance baseline* di riferimento del sistema in condizioni nominali. Utilizzando l'esecutore *shared-iterations* [55], è stata simulata l'ingestione massiva dell'intero *dataset* (più di 750 iterazioni), distribuendo il carico su un *pool* di utenti virtuali concorrenti. Lo scopo di questa fase è misurare il tempo di smaltimento di una coda di lavoro predefinita e analizzare la distribuzione statistica delle latenze (*http_req_duration*).
- **Stress Test** (Test di Sforzo e Saturazione): Mirato a identificare il "punto di rottura" (*breaking point*) dell'architettura. Tramite l'esecutore *ramping-arrival-rate*, il sistema è stato sottoposto a un volume di richieste al secondo (req/s) in costante aumento, forzando l'infrastruttura a elaborare traffico indipendentemente dai propri tempi di risposta, al fine di valutare il comportamento degradato e la resilienza sotto pressione estrema.

4.3 Esecuzione del *Benchmark*

L'esecuzione dei *benchmark* è stata correlata ad un monitoraggio continuo dell'utilizzo delle risorse infrastrutturali (RAM, CPU) durante i test, tramite un campionamento telemetrico con il comando 'docker stats' (con granularità di 1 secondo). Infine, per garantire la significatività statistica e mitigare le fluttuazioni, tutte le sessioni di test sono state eseguite sulla medesima infrastruttura *hardware*. I risultati sono stati ottenuti eseguendo la media su tre cicli di iterazioni.

4.3.1 *Baseline Performance: Load Test*

Per questa sessione è stato utilizzato l'esecutore k6 *shared-iterations* [55]. Questa configurazione permette di definire un numero fisso di iterazioni totali (più di 700, pari all'esatto numero di *record* del *dataset* di *input*) distribuite tra un *pool* di *Virtual Users* (VUs) concorrenti (10 nel test). Tale approccio simula fedelmente un processo di caricamento massivo (*bulk ingestion*), in cui il sistema deve processare una coda di richieste predefinita nel minor tempo possibile. L'analisi dei risultati ottenuti si è focalizzata sulla distribuzione della latenza delle richieste HTTP (*http_req_duration*). I risultati sono stati sintetizzati graficamente tramite un istogramma

¹⁰ Gli *executors* nel *framework* K6, sono dei motori che oltre ad eseguire il test, ne guidano la strategia ed il modo in cui il carico di lavoro viene distribuito nel tempo, definendo il comportamento degli Utenti Virtuali (VUs).

a gradoni su scala logaritmica (Figura 8), che espone il conteggio assoluto delle richieste per ciascun intervallo di tempo.

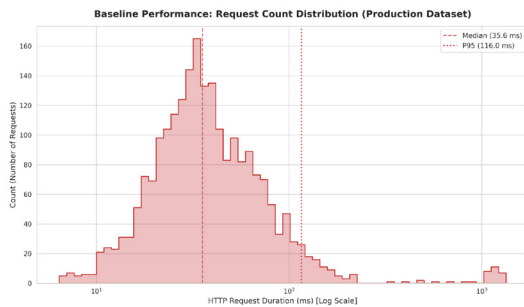


Figura 8 Istogramma a gradoni della distribuzione della latenza aggregata su 3 esecuzioni del Load Test (Dataset di produzione).

Figure 8 Histogram of the aggregate latency distribution over 3 executions of the Load Test (Production Dataset).

I risultati empirici confermano la solidità dell'architettura, pur evidenziando l'impatto fisiologico durante il processamento di *payload* di grandi dimensioni. La latenza mediana risulta estremamente contenuta (circa 35.6 ms), corrispondente al picco di concentrazione del grafico (la forma a campana). Questo dimostra che il core applicativo elabora la stragrande maggioranza delle richieste standard con grande rapidità. L'istogramma evidenzia una pronunciata "coda lunga" nella sua porzione terminale destra: la distribuzione presenta blocchi isolati di richieste che si spingono oltre i 1000 ms. Questi "gradini" estremi coincidono con l'elaborazione dei *record* più complessi (superiori ai 500 KB). A causa della natura sincrona del *parsing* JSON in Node.js, l'elaborazione di questi flussi dati trattiene momentaneamente il *thread* logico principale, rallentando non solo la transazione corrente, ma innescando un lieve ritardo a cascata sulle richieste in attesa nella coda di smistamento.

Nonostante l'allungamento dei tempi di risposta causato dai *record* più onerosi, il 95° percentile (P95) si mantiene su valori ampiamente accettabili (116.0 ms). Soprattutto, il sistema ha elaborato con successo il 100% delle richieste, senza registrare alcun errore di connessione o *timeout*, confermando l'assoluta stabilità della *pipeline* di *ingestion* in condizioni di carico a dimensione fissa distribuito su un numero di VUs.

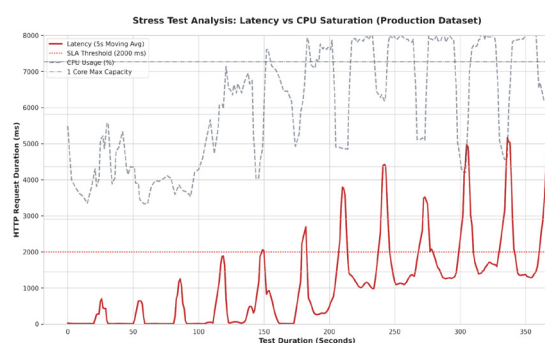
4.3.2 Saturazione e Resilienza: Stress Test

La seconda fase della validazione ha avuto lo scopo di spingere l'infrastruttura oltre le normali condizioni operative, identificando il punto di saturazione (*breaking point*) e valutando la resilienza del sistema sotto un carico estremo. Il test è stato progettato per forzare l'architettura a elaborare un volume di richieste al secondo (req/s) in costante aumento, utilizzando l'esecutore *k6 ramping-arrival-rate* [56]. Per simulare un incremento di carico realistico, l'esecutore è stato configurato secondo un profilo "a gradini". Il test avvia l'esecuzione iniettando un carico base di 20 richieste al secondo, innalzando progressivamente il volume di traffico attraverso quattro fasi di intensità crescente, con *target* prefissati rispettivamente a 50, 85, 120 e 160 req/s; ognuna delle fasi è della durata di 60 secondi, scalando fino a un massimo di 150 Utenti Virtuali (*maxVUs*) concorrenti.

L'analisi congiunta della latenza applicativa (req/s) e dell'utilizzo delle risorse *hardware* (utilizzo % CPU), riportata in Figura 9, evidenzia in modo inequivocabile la correlazione tra il carico di richieste e il limite fisico del *runtime* Node.js in esecuzione su un core CPU.

Figura 9 Analisi dello Stress Test: correlazione tra il degrado della latenza (durata delle HTTP req/s in millisecondi) e la saturazione della CPU.

Figure 9 Stress Test Analysis: correlation between latency degradation (HTTP req/s duration in milliseconds) and CPU saturation.



Osservando le metriche nel corso del tempo, è possibile distinguere due fasi comportamentali nettamente separate:

- Fase di Tolleranza (0 - 150 secondi): Nella prima metà dell'esperimento, il sistema gestisce l'incremento di traffico con notevole efficacia. Il consumo di CPU oscilla fisiologicamente per far fronte all'elaborazione dei *payload*, ma l'architettura riesce a smaltire il carico senza accumulare ritardi critici. I picchi di latenza dovuti al *parsing* dei *record* più pesanti vengono riassorbiti rapidamente e si mantengono costantemente al di sotto della soglia di allerta (definita a 2000 ms). È interessante notare la natura ciclica ed equidistante di tali picchi: questo andamento "a onde" è la diretta conseguenza visiva dell'iterazione continua dello *script k6* sul medesimo *dataset* (§4.2.1). A ogni nuovo ciclo, infatti, il sistema si ritrova fisiologicamente a processare lo stesso sottoinsieme di *payload*, innescando i rallentamenti periodici.
- Punto di Saturazione e relativo degrado (Oltre i 150 secondi): All'aumentare ulteriore della pressione, la richiesta di risorse computazionali supera la capacità elaborativa del sistema. Intorno al 150° secondo, la linea di monitoraggio della CPU raggiunge per la prima volta il limite architetturale del 100% su singolo *core*. A partire da questo *breaking point*, il sistema entra in una fase di saturazione cronica. L'*Event Loop* di Node.js [57], bloccato dalle onerose operazioni sincrone di elaborazione degli oggetti JSON più grandi, non è più in grado di servire tempestivamente le nuove connessioni in ingresso. Ciò innesca un accodamento delle richieste (*queueing*) che si traduce in un'esplosione delle latenze, le quali superano ripetutamente la soglia di allerta con picchi che si spingono oltre i 4000 ms.

I dati quantitativi dimostrano che il vero collo di bottiglia dell'attuale modello di *deployment* non risiede nel livello di persistenza (MongoDB) o nella rete, bensì nel limite intrinseco dell'architettura *single-threaded* di Node.js di fronte a operazioni fortemente *CPU-intensive*.

Tuttavia, sotto il profilo della resilienza, il test è da considerarsi un successo. Nonostante l'evidente degrado prestazionale indotto dall'ingente carico di richieste, il sistema ha dimostrato di saper reggere: durante l'intera fase di "asfissia" della CPU, MetaCat non ha registrato alcun crash applicativo, non ha manifestato errori di memoria esaurita e ha continuato a processare e salvare con successo l'intero traffico in ingresso, mantenendo un tasso di errori HTTP pari a zero.

5. Conclusioni

Il presente rapporto tecnico ha illustrato la progettazione e l'implementazione di **MetaCat**, un servizio *backend* dedicato alla catalogazione di metadati relativi a produzioni scientifiche coerente con lo standard DataCite 4.6. Più che un prodotto commerciale finito, MetaCat si configura come un caso di studio di **Ricerca e Sviluppo (R&D)**, nato con l'obiettivo di dimostrare

la validità e l'efficacia di un'architettura moderna, disaccoppiata e basata su microservizi nel contesto della gestione dei dati della ricerca.

I risultati ottenuti confermano che l'adozione di uno *stack* tecnologico basato su Node.js, MongoDB e Docker rappresenta un'alternativa robusta e performante rispetto alle tradizionali piattaforme monolitiche. Le scelte implementative adottate – in particolare la separazione netta dei livelli architetturali (*Layered Architecture*), la doppia validazione degli schemi e la *containerizzazione* nativa – hanno permesso di realizzare un sistema che è al contempo leggero, scalabile e rigorosamente conforme ai principi di integrità dei dati.

In linea con la filosofia *Open Science* che permea il progetto, MetaCat è rilasciato alla comunità come *software* libero sotto licenza **GNU Affero General Public License (AGPL) v3.0**. Il codice sorgente è reperibile sul servizio di *hosting* GitHub al url <https://github.com/fpistagna/metacat>. Questa scelta garantisce che qualsiasi evoluzione o servizio derivato rimanga aperto e accessibile, incoraggiando la collaborazione e il riutilizzo del codice all'interno della comunità scientifica.

Bibliografia

- Budapest Open Access Initiative - BOAI (2002). Interlending & Document Supply, 30(2). <https://doi.org/10.1108/ilds.2002.12230bab.012>
- Cousijn, H., Braukmann, R., Fenner, M., Hellström, M., Ignat, T., Kraus, P., McEntyre, J., & Roussakis, K. (2021). Connected Research: The Potential of the PID Graph. *Patterns*, 2(1), 100180. <https://doi.org/10.1016/j.patter.2020.100180>
- DataCite Metadata Working Group. (2024). DataCite Metadata Schema for the Publication and Citation of Research Data and Other Research Outputs (Version 4.6). DataCite e.V. <https://doi.org/10.14454/csba-e454>
- DISS Working Group (2025). Database of Individual Seismogenic Sources (DISS), Version 3.3.1: A compilation of potential sources for earthquakes larger than M 5.5 in Italy and surrounding areas. Istituto Nazionale di Geofisica e Vulcanologia (INGV). <https://doi.org/10.13127/diss3.3.1>
- INGV Data Management Office. (2020). INGV Open Data Registry, the metadata catalogue of the Istituto Nazionale di Geofisica e Vulcanologia. Istituto Nazionale di Geofisica e Vulcanologia (INGV). <https://doi.org/10.13127/data-registry>
- Locati, M., Mazza, S., Montalto, P., Romano, V., Lauciani, V., Vallone, R., & Cacciaguerra, S. (2026). The INGV data registry as a curated metadata infrastructure for Earth Science data stewardship. *Scientific Data*, 13(1). <https://doi.org/10.1038/s41597-026-06980-3>
- Neumann, J., & Brase, J. (2014). DataCite and DOI names for research data. *Journal of Computer-Aided Molecular Design*, 28, 1035–1041. <https://doi.org/10.1007/s10822-014-9776-5>
- Pistagna, F. (2025). MetaCat: A Backend Service for DataCite Metadata (Version 0.1.4-alpha) [Software]. *Zenodo*. <https://doi.org/10.5281/zenodo.17436861>
- Pistagna, F., Torrisi, M., Cassisi, C., Locati, M., & Montalto, P. (2023). OEDataRep: the new version of the Osservatorio Etneo Open Data Repository. *Rapporti Tecnici INGV*, 476. <https://doi.org/10.13127/rpt/476>
- Puglisi, G., Basili, R., Chiodetti, A., Cianchi, A., Drudi, M., Freda, C., Locati, M., Pignone, M., & Sangianantoni, A. (2018). La Politica dei Dati dell'INGV / The INGV Data Policy. Istituto Nazionale di Geofisica e Vulcanologia (INGV). <https://hdl.handle.net/2122/14886>
- Rovida A., Locati M., Camassi R., Lolli, B., Gasperini P., Antonucci A., 2022. Catalogo Parametrico dei Terremoti Italiani (CPTI15), versione 4.0. Istituto Nazionale di Geofisica e Vulcanologia (INGV). <https://doi.org/10.13127/cpti/cpti15.4>

- Stallman, R. (s.d.). Why Open Source misses the point of Free Software. GNU Operating System. <https://www.gnu.org/philosophy/open-source-misses-the-point.html>
- Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., et al. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3, 160018. <https://doi.org/10.1038/sdata.2016.18>
- Yeats, D. (2007). Morality and Pragmatism in Free Software and Open Source. In K. St.Amant, & B. Still (Eds.), *Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives*. Idea Group Inc (IGI). <https://doi.org/10.4018/978-1-59140-999-1.ch003>

Sitografia

- [1] https://en.wikipedia.org/wiki/Open_data - Wikipedia (2011). Open data. Wikipedia
Data di consultazione: 23/07/2025.
- [2] <https://www.go-fair.org/fair-principles/> - GO FAIR (n.d.). FAIR Principles. GO FAIR
Data di consultazione: 23/07/2025.
- [3] <https://www.fsf.org/> - Free Software Foundation (n.d.). FSF - Free Software Foundation.
Data di consultazione: 23/07/2025.
- [4] <https://www.paristechreview.com/2013/03/29/brief-history-open-data/> - Chignard, S. (2023). A brief history of Open Data. Paris Tech Review. Data di consultazione: 23/07/2025.
- [5] <https://www.doi.org/> - DOI Foundation(n.d.). DOI. DOI.org. Data di consultazione: 23/07/2025.
- [6] <https://www.doi.org/the-identifier/what-is-a-doi/> - DOI Foundation (n.d.). What is a DOI?. DOI.org. Data di consultazione: 23/07/2025.
- [7] <https://www.openaccess.nl/en/about-open-access/what-is-open-access> - Openaccess.nl (n.d.). What is open access?. Openaccess.nl. Data di consultazione: 28/07/2025.
- [8] <https://datacite.org/mission/> - DataCite (n.d.). Mission & Vision. DataCite.
Data di consultazione: 28/07/2025.
- [9] <https://datacite-metadataschema.readthedocs.io/en/4.6/properties/relatedidentifier/#relatedidentifier> DataCite (n.d.). 12. RelatedIdentifier - DataCite Metadata Schema 4.6 documentation. Data di consultazione: 05/06/2026.
- [10] <https://scienceeurope.org/our-priorities/open-science/research-data-management/> Science Europe (n.d.). Research Data Management. Science Europe.
Data di consultazione: 19/08/2025.
- [11] <https://inveniordm.docs.cern.ch/features/> - CERN (n.d.). Overview - Turn-key research data management repository. InvenioRDM. Data di consultazione: 19/08/2025.
- [12] <https://ckan.org/> - CKAN (n.d.). CKAN - The open source data management system CKAN. Data di consultazione: 19/08/2025.
- [13] <https://dspace.org/> - DSpace (n.d.). DSpace. DSpace.org. Data di consultazione: 20/08/2025.
- [14] <https://dataverse.org/> - The Dataverse Project (n.d.). The Dataverse Project. Dataverse.org. Data di consultazione: 20/08/2025.
- [15] <https://oedatarep.ct.ingv.it/> - Istituto Nazionale di Geofisica e Vulcanologia [INGV] (n.d.). INGV - OEDatarep. OEDatarep. Data di consultazione: 15/12/2025.
- [16] <https://aws.amazon.com/compare/the-difference-between-frontend-and-backend> Amazon Web Services [AWS] (n.d.). Front End vs Back End - Difference Between Application Development. AWS. Data di consultazione: 12/11/2025.
- [17] <https://github.com/fpistagna/metacat> - Pistagna, F. (n.d.). fpistagna/metacat: A simple CRUD REST backend for Datacite Metadata. GitHub: . Data di consultazione: 10/08/2025.
- [18] <https://datacite-metadataschema.readthedocs.io/en/4.6/> - DataCite Metadata Working Group (n.d.). DataCite Metadata Schema Documentation for the Publication and Citation of Research Data and Other Research Outputs DataCite. Data di consultazione 20/08/2025.

- [19] <https://data.ingv.it/docs/principi/index.html> - Istituto Nazionale di Geofisica e Vulcanologia [INGV] (n.d.). Principi della Politica dei Dati dell'INGV. Data INGV. Data di consultazione: 20/08/2025.
- [20] <https://data.ingv.it> - Istituto Nazionale di Geofisica e Vulcanologia [INGV] (n.d.). Data INGV. INGV.it. Data di consultazione: 20/08/2025.
- [21] <https://emidius.mi.ingv.it> - Istituto Nazionale di Geofisica e Vulcanologia [INGV] (n.d.). CPTI15 v4.0 Catalogo Parametrico dei Terremoti Italiani. CPTI15-DBMI15 v4.0: CPTI15 DBMI15/. Data di consultazione: 05/06/2026.
- [22] <https://it.wikipedia.org/wiki/DevOps> - Wikipedia (n.d.). DevOps. Wikipedia: Data di consultazione: 12/11/2025.
- [23] <https://nodejs.org/en> - OpenJS Foundation (n.d.). Node.js. Node.js. Data di consultazione: 12/11/2025.
- [24] <https://expressjs.com/> - OpenJS Foundation (n.d.). Express.js. Express.js. Data di consultazione: 12/11/2025.
- [25] <https://www.mongodb.com/> - MongoDB, Inc. (n.d.). MongoDB. MongoDB. Data di consultazione: 12/11/2025.
- [26] <https://mongoosejs.com/> - LearnBoost (n.d.). Mongoose. Mongoose.js. Data di consultazione: 12/11/2025.
- [27] <https://www.jwt.io/> - Auth0 (n.d.). JSON Web Tokens. JWT.io. Data di consultazione: 12/11/2025.
- [28] <https://oauth.net/2/> - OAuth (n.d.). OAuth 2.0. OAuth.net. Data di consultazione: 12/11/2025.
- [29] <https://orcid.org/> - ORCID (n.d.). ORCID. ORCID.org. Data di consultazione: 12/11/2025.
- [30] <https://ajv.js.org/> - Ajv (n.d.). Ajv JSON schema validator. Ajv.js.org. Data di consultazione: 12/11/2025.
- [31] <https://www.json.org/json-en.html> - JSON (n.d.). Introducing JSON. JSON.org. Data di consultazione: 12/11/2025.
- [32] <https://mochajs.org/> - Mocha (n.d.). MochaJS. MochaJS.org. Data di consultazione: 12/11/2025.
- [33] <https://www.chaijs.com/> - Chai (n.d.). Chai Assertion Library. Chaijs.com. Data di consultazione: 12/11/2025.
- [34] <https://github.com/sequelize/umzug> - Sequelize (n.d.). sequelize/umzug: Framework agnostic migration tool for Node.js. GitHub. Data di consultazione: 12/11/2025.
- [35] <https://docs.docker.com/get-started-docker-overview/> - Docker Inc. (n.d.). What is Docker?. Docker Docs. Data di consultazione: 12/11/2025.
- [36] <https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/> - F5, Inc. (n.d.). NGINX Reverse Proxy. NGINX Documentation. Data di consultazione: 12/11/2025.
- [37] <https://www.openapis.org> - OpenAPI Initiative (n.d.). OpenAPI Initiative. OpenAPIs.org: Data di consultazione: 12/11/2025.
- [38] <https://swagger.io/> - SmartBear Software (n.d.). Swagger. Swagger.io. Data di consultazione: 12/11/2025.
- [39] https://en.wikipedia.org/wiki/Single-responsibility_principle Wikipedia (n.d.). Single responsibility principle. Wikipedia. Data di consultazione: 22/12/2025.
- [40] https://en.wikipedia.org/wiki/Multitier_architecture - Wikipedia (n.d.). Multitier architecture. Wikipedia. Data di consultazione: 22/12/2025.
- [41] <https://expressjs.com/en/guide/using-middleware.html> - OpenJS Foundation (n.d.). Using Express middleware. Express.js. Data di consultazione: 23/12/2025.
- [42] https://en.wikipedia.org/wiki/Data/access_object - Wikipedia: - Wikipedia (n.d.). Data access object. Data di consultazione: 22/12/2025.
- [43] <https://www.mongodb.com/docs/manual/reference/operator/aggregation/lookup> MongoDB, Inc. (s.d.). \$lookup (aggregation). MongoDB Manual. Data di consultazione: 21/04/2026.

- [44] <https://docs.docker.com/compose/intro/compose-application-model/> Docker - Inc. (n.d.). How Compose works. Docker Docs. Data di consultazione: 22/12/2025.
- [45] <https://www.redhat.com/it/topics/automation/what-is-infrastructure-as-code-iac> - Red Hat (n.d.). Cos'è l'Infrastructure as Code (IaC)?. Data di consultazione: 05/06/2026.
- [46] <https://www.redhat.com/it/topics/automation/what-is-yaml> - Red Hat (n.d.). Cos'è YAML?. Data di consultazione: 05/06/2026.
- [47] <https://docs.docker.com/engine/storage/volumes/> Docker - Inc. (n.d.). Volumes|Docker Docs. Data di Consultazione: 23/04/2026.
- [48] <https://docs.docker.com/build/building/multi-stage/> Docker - Inc. (n.d.). Multi-stage Docker Docs. Data di consultazione: 22/12/2025.
- [49] <https://docs.docker.com/engine/network/> - Docker Inc. (n.d.). Networking Overview. Data di consultazione: 23/04/2026.
- [50] <https://grafana.com/docs/k6/latest/> Grafana - Labs (n.d.). Grafana k6 | Grafana k6 documentation: . Data di consultazione: 07/05/2026.
- [51] <https://data.ingv.it/api/3/> - Istituto Nazionale di Geofisica e Vulcanologia (n.d.). INGV Open Data Portal - CKAN API. Data di consultazione: 05/06/2026.
- [52] <https://gist.github.com/fpistagna/d390a2ca54397fcf3338d83bdbfba8c3> - Pistagna, F. (n.d.). MetaCat Data Ingestion. GitHub Gist. Data di consultazione: 08/01/2026.
- [53] <https://doi.org/> - DOI Foundation (n.d.). Digital Object Identifier System. Data di consultazione: 05/06/2026.
- [54] <https://citation.doi.org/docs.html> - DOI Foundation (n.d.). DOI Citation Formatter. Data di consultazione: 20/04/2026.
- [55] <https://grafana.com/docs/k6/latest/using-k6/scenarios/executors/shared-iterations> Grafana Labs (n.d.). Shared iterations. Data di consultazione: 05/06/2026.
- [56] <https://grafana.com/docs/k6/latest/using-k6/scenarios/executors/ramping-arrival-rate> Grafana Labs (n.d.). Ramping arrival rate: . Data di consultazione: 05/06/2026.
- [57] <https://nodejs.org/learn/asynchronous-work/event-loop-timers-and-nexttick#event-loop-explained> - The Node.js Event Loop | Node.js Learn. Data di Consultazione: 21/04/2026.



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

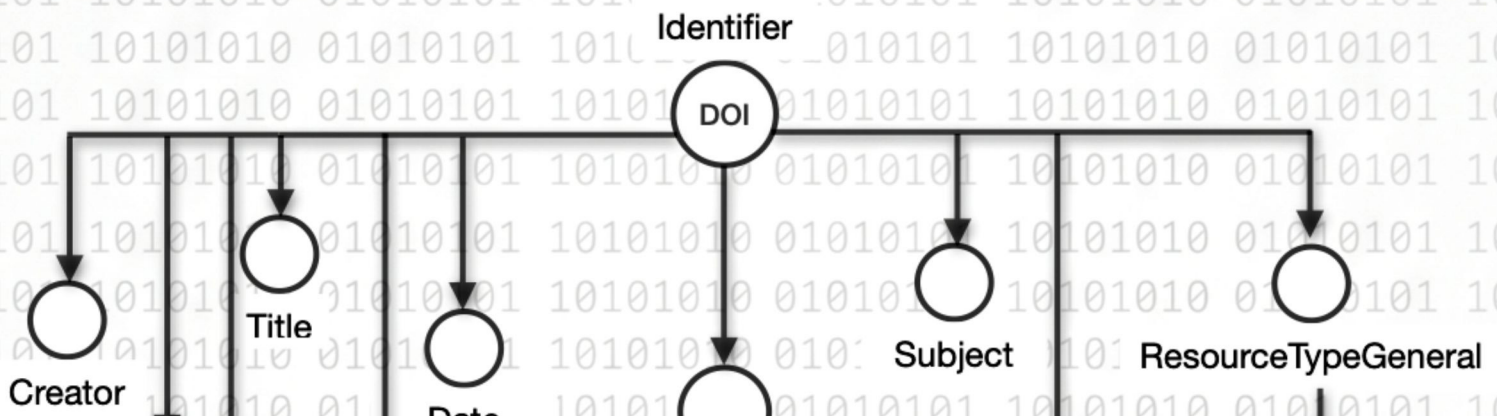
© 2026 the Author(s). Open Access.
This article is licensed under a Creative Commons Attribution 4.0 International License.



Creative Commons Attribution 4.0 International

MetaCat

< MetaData Catalog/ >



ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA

Contributor

Publisher

GeoLocation

Software