

Specifica Tecnica

v0.1



7Last



Versioni

Ver.	Data	Autore	Verificatore	Descrizione
0.1	02/06/2024	Matteo Tiozzo		Stesura struttura del documento

Indice

1	Introduzione	6
1.1	Scopo della specifica tecnica	6
1.2	Scopo del prodotto	6
1.3	Glossario	6
1.4	Riferimenti	6
1.4.1	Normativi	6
1.4.2	Informativi	7
2	Tecnologie	8
2.1	Docker	8
2.1.1	Ambienti	8
2.1.2	Immagini Docker	8
2.2	Linguaggi e formato dati	10
2.3	Librerie	11
2.4	Servizi	12
2.4.1	Redpanda	12
2.4.1.1	Vantaggi	12
2.4.1.2	Casi d'uso	13
2.4.1.3	Impiego nel progetto	13
2.4.2	ClickHouse	13
2.4.2.1	Vantaggi	14
2.4.2.2	Casi d'uso	14
2.4.2.3	Impiego nel progetto	14
2.4.3	Apache Flink	15
2.4.3.1	Vantaggi	15
2.4.3.2	Casi d'uso	16
2.4.3.3	Impiego nel progetto	16
2.4.4	Grafana	16
2.4.4.1	Vantaggi	17
2.4.4.2	Casi d'uso	17
2.4.4.3	Impiego nel progetto	17
3	Architettura di sistema	19
3.1	<i>Data processing architectures</i>	19
3.1.1	<i>Architettura lambda</i>	19



3.1.1.1	Vantaggi e svantaggi	20
3.1.1.2	Casi d'uso	20
3.1.2	Architettura <i>kappa</i>	20
3.1.2.1	Vantaggi e svantaggi	20
3.1.2.2	Casi d'uso	21
3.2	Architettura scelta	21
3.2.1	Componenti di sistema	21
3.2.2	Flusso di dati	22
3.3	Architettura dei simulatori	23
3.3.1	Modulo <i>models</i>	24
3.3.2	Modulo <i>producers</i>	26
3.3.3	Modulo <i>serializers</i>	26
3.3.4	Modulo <i>simulators</i>	26
3.4	Redpanda	26
3.4.1	<i>Topic</i>	26
3.4.2	Partizioni e chiavi	27
3.4.3	Redpanda schema registry	27
3.4.3.1	<i>Compatibility mode</i>	27
3.4.3.2	Serializzazione dei dati	28
3.4.3.2.1	Chiavi	28
3.4.3.2.2	Valori	28
3.4.3.3	Formato dei dati	29
3.4.3.4	<i>Topic name strategy</i>	29
3.4.4	Redpanda Connect	29
3.4.4.1	<i>Sink connector</i> per ClickHouse	29
3.4.4.2	Avro converter	31
3.4.5	Redpanda Console	32
3.5	Flink - <i>Processing layer</i>	32
3.5.1	Introduzione	32
3.5.2	Componenti Flink e <i>processing layer</i>	33
3.5.3	<i>Processing layer data-flow</i>	33
3.5.4	<i>Job</i>	33
3.5.4.1	<i>Heat Index</i>	33
3.5.4.2	Efficienza delle colonnine elettriche	33
3.6	Database ClickHouse	33
3.6.1	Funzionalità utilizzate	33



3.6.1.1	Materialized View	33
3.6.1.2	MergeTree	34
3.6.2	Trasferimento dati tramite Materialized View	35
3.6.3	Misurazioni isole ecologiche	35
3.6.4	Misurazioni temperatura	36
3.6.5	Misurazioni traffico	36
3.7	Grafana	37
3.7.1	Dashboard	37
3.7.2	ClickHouse datasource plugin	37
3.7.2.1	Configurazione del Datasource	37
3.7.3	Variabili Grafana	37
3.7.3.1	Documentazione	37
3.7.4	Grafana Alerts	38
3.7.4.1	Configurazione delle regole di alert	38
3.7.4.2	Configurazione canale di notifica	39
3.7.5	Altri plugin	39
3.7.5.1	Orchestra Cities Map plugin	39
4	Architettura di deployment	41
5	Requisiti	42
5.1	Requisiti funzionali	42

Indice delle tabelle

1	Linguaggi e formato dati	11
2	Librerie utilizzate	12
3	Requisiti funzionali	47

Indice delle immagini

1	Architettura <i>lambda</i>	19
2	Architettura <i>kappa</i>	20
3	Componenti di sistema ad alto livello	22
4	Flusso di dati all'interno del sistema. I sensori di precipitazioni, isole ecologiche, livello dei fiumi e traffico sono stati omessi per chiarezza, ma il percorso di tali dati è analogo a quello dei sensori di qualità dell'aria.	22
5	Diagramma delle classi del modulo <code>models</code>	25
6	Diagramma delle classi del modulo <code>producers</code>	26
7	Diagramma delle classi del modulo <code>serializers</code>	26
8	Percentuale di soddisfacimento dei requisiti funzionali	47
9	Percentuale di soddisfacimento dei requisiti totale	48



1 Introduzione

1.1 Scopo della specifica tecnica

Questo documento è rivolto a tutti gli *stakeholder* coinvolti nel progetto **SyncCity** - *A smart city monitoring platform*. Esso ha lo scopo di fornire una visione dettagliata riguardo l'architettura del sistema, i *design pattern* utilizzati, le tecnologie adottate e le scelte progettuali effettuate. Inoltre, contiene diagrammi UML delle classi e delle attività.

1.2 Scopo del prodotto

Lo scopo del prodotto è realizzare un prototipo di una piattaforma di monitoraggio per una *Smart City*, la quale permetta di raccogliere e analizzare dati provenienti da sensori IoT posizionati nelle città. Questi dati, una volta elaborati, devono essere visualizzati in maniera chiara e intuitiva, tramite grafici e mappe, per permettere alle autorità locali della città di prendere decisioni tempestive e mirate per migliorare la qualità della vita dei cittadini.

1.3 Glossario

Per evitare qualsiasi ambiguità o malinteso sui termini utilizzati nel documento, verrà adottato un glossario. Questo glossario conterrà varie definizioni. Ogni termine incluso nel glossario sarà indicato applicando uno stile specifico:

- aggiungendo una "G" al pedice della parola;
- fornendo il link al glossario online.

1.4 Riferimenti

1.4.1 Normativi

- Standard **ISO 8601**: <https://www.iso.org/iso-8601-date-and-time-format.html>;
-



1.4.2 Informativi

-
-



2 Tecnologie

Questa sezione si occupa di fornire una panoramica delle tecnologie utilizzate per implementare il sistema software. In particolare, delinea le piattaforme, gli strumenti, i linguaggi di programmazione, i framework e altre risorse tecnologiche che sono state impiegate durante lo sviluppo.

2.1 Docker

È una piattaforma di virtualizzazione leggera che semplifica lo sviluppo, il testing e il rilascio delle applicazioni fornendo un ambiente isolato e riproducibile. È utilizzato per creare ambienti di sviluppo standardizzati, facilitare la scalabilità delle applicazioni e semplificare la gestione delle risorse.

2.1.1 Ambienti

Per lo sviluppo di questo progetto sono stati ipotizzati i due seguenti scenari di esecuzione, separati grazie all'utilizzo di profili diversi di Docker Compose:

- **local**: utilizzato dagli sviluppatori per testare e sviluppare le funzionalità dell'applicazione sui propri computer. Questo ambiente permette di eseguire tutti i componenti del sistema all'interno di un container Docker, ad eccezione del simulatore Python. Esso viene eseguito direttamente sul sistema operativo dell'utente, in modo da facilitare il debugging e il testing delle funzionalità, senza dover necessariamente eseguire la *build* dell'immagine Docker ad ogni modifica del codice;
- **release**: utilizzato quando si desidera simulare un ipotetico ambiente di produzione o non è necessario modificare il codice Python. Consente di non dover manualmente installare le dipendenze o configurare l'ambiente di esecuzione. In questo caso, tutti i componenti del sistema vengono eseguiti all'interno di container Docker.

2.1.2 Immagini Docker

Nello sviluppo di questo progetto *7Last* ha utilizzato diverse immagini Docker di seguito elencate.

- **Simulator - Python**
 - **Immagine**: python:3.11.9-alpine;



- **Riferimento:** Python Docker Image [Ultima consultazione: 2024-06-02].
- **Ambiente:** `release`;
- **Redpanda Init:** l'immagine di `alpine` viene utilizzata per creare un container che si occupa di inizializzare il broker Redpanda.
 - **Immagine:** `alpine:3.20.1`;
 - **Riferimento:** Alpine [Ultima consultazione: 2024-06-25].
 - **Ambiente:** `local`, `release`.
- **Redpanda**
 - **Immagine:** `docker.redpanda.com/redpandadata/redpanda:v23.3.11`;
 - **Riferimento:** Redpanda Docker Image [Ultima consultazione: 2024-06-02].
 - **Ambiente:** `local`, `release`.
- **Redpanda console**
 - **Immagine:** `docker.redpanda.com/redpandadata/console:v2.4.6`;
 - **Riferimento:** Redpanda Console Docker Image [Ultima consultazione: 2024-06-02].
 - **Ambiente:** `local`, `release`.
- **Connectors**
 - **Immagine:** `docker.redpanda.com/redpandadata/connectors:v1.0.27`;
 - **Riferimento:** Redpanda Connectors Docker Image [Ultima consultazione: 2024-06-02].
 - **Ambiente:** `local`, `release`.
- **ClickHouse**
 - **Immagine:** `clickhouse/clickhouse-server:24-alpine`;
 - **Riferimento:** ClickHouse Docker Image [Ultima consultazione: 2024-06-02].
 - **Ambiente:** `local`, `release`.
- **Grafana**



- **Immagine:** grafana/grafana-oss:10.3.0;
- **Riferimento:** [Grafana Docker Image](#) [Ultima consultazione: 2024-06-02].
- **Ambiente:** local, release.

- **Apache Flink**

- **Immagine:** flink:1.18.1-java17;
- **Riferimento:** [Flink Docker Image](#) [Ultima consultazione: 2024-06-02].
- **Ambiente:** local, release.

2.2 Linguaggi e formato dati

Nome	Versione	Descrizione	Impiego
Python	3.11.9	Linguaggio di programmazione ad alto livello, interpretato e multiparadigma.	Simulatore di sensori, <i>testing</i> , <i>script</i> per automatizzare il <i>deployment</i> dei <i>job</i> di Flink.
JSON	-	Formato di dati semplice da interpretare e generare, ampiamente utilizzato per lo scambio di dati tra applicazioni.	Configurazione <i>dashboard</i> Grafana, definizione di schemi Avro.
YAML	-	Linguaggio di serializzazione dei dati leggibile sia per gli esseri umani sia per le macchine.	Docker Compose, provisioning Grafana e configurazione <i>alert</i> , file di <i>workflow</i> per le GitHub Actions.
SQL	Ansi SQL	Linguaggio di programmazione specificamente progettato per la gestione e la manipolazione di dati all'interno di sistemi di gestione di database.	<i>Query</i> e gestione database ClickHouse.



Nome	Versione	Descrizione	Impiego
TOML	1.0.0	Linguaggio di <i>markup</i> progettato per essere più leggibile e facile da scrivere rispetto ad altri formati di configurazione come JSON e YAML.	Configurazione e gestione dei sensori simulati.
Java	17	Linguaggio di programmazione ad alto livello, orientato agli oggetti.	Creazione di <i>job</i> per le aggregazioni dei dati in Apache Flink.

Tabella 1: Linguaggi e formato dati

2.3 Librerie

Python		
Nome	Versione	Impiego
<code>confluent_avro</code>	1.8.0	Serializzazione dei dati in formato Avro.
<code>coverage</code>	7.5.1	Strumento per misurare la percentuale di linee di codice e rami coperti dai test.
<code>isodate</code>	0.6.1	Libreria per la manipolazione delle date e delle ore in formato ISO8601.
<code>kafka-python-ng</code>	2.2.2	Client Kafka per Python.
<code>ruff</code>	0.3.5	Libreria per l'analisi statica del codice.
<code>toml</code>	0.10.2	Libreria per effettuare il parsing dei file di configurazione in formato TOML.
Java		
<code>flink-streaming-java</code>	1.18.0	Utilizzo di DataStream API di Flink.
<code>flink-connector-kafka</code>	3.1.0-1.18	Connessione di Flink a Kafka.
<code>flink-clients</code>	1.18.0	Creazione di <i>job</i> di Flink.
<code>flink-java</code>	1.18.0	Creazione di <i>job</i> di Flink.
<code>flink-avro-confluent-registry</code>	1.18.0	Connessione di Flink a uno <i>schema registry</i> che utilizza Avro.



Nome	Versione	Impiego
flink-shaded-guava	31.1-jre-17.0	Gestione delle dipendenze di Flink.
slf4j-simple	1.7.36	Implementazione di SLF4J.
lombok	1.18.32	Libreria per la generazione di codice <i>boilerplate</i> .
maven-assembly-plugin	3.7.1	Plugin Maven per la creazione di un <i>fat jar</i> .

Tabella 2: Librerie utilizzate

2.4 Servizi

2.4.1 Redpanda

Redpanda è una piattaforma di streaming sviluppata in C++. Il suo obiettivo è fornire una soluzione leggera, semplice e performante, pensata per essere un'alternativa ad Apache Kafka. Viene utilizzato per disaccoppiare i dati provenienti dal simulatore.

- **Versione:** v23.3.11;
- **documentazione:** <https://docs.redpanda.com/current/home/> [Ultima consultazione: 2024-06-02].

2.4.1.1 Vantaggi

I vantaggi nell'utilizzo di questo strumento consistono in:

- **performance:** è scritto in C++ e utilizza il *framework* Seastar, offrendo un'architettura *thread-per-core* ad alte prestazioni. Ciò permette di ottenere un'elevata *throughput* e latenze costantemente basse, evitando cambi di contesto e blocchi. Inoltre, è progettato per sfruttare l'*hardware* moderno, tra cui unità NVMe, processori *multi-core* e interfacce di rete ad alta velocità;
- **semplicità di configurazione:** oltre al *message broker*, contiene anche un *proxy* HTTP e uno *schema registry*;
- **minore richiesta di risorse:** rispetto ad Apache Kafka, richiede meno risorse per l'esecuzione in locale, rendendolo più adatto per l'esecuzione su *hardware* meno potente;



- **compatibilità con le API di Kafka:** è compatibile con le API di Apache Kafka, consentendo di utilizzare le librerie e gli strumenti esistenti;

2.4.1.2 Casi d'uso

Tra i casi d'uso di Redpanda si possono citare:

- **streaming di eventi**, permettendo la gestione e l'elaborazione di flussi di dati in tempo reale;
- **data integration**, agisce come un intermediario flessibile e robusto per l'integrazione dei dati, consentendo la raccolta, il trasporto e la trasformazione dei dati provenienti da diverse sorgenti verso varie destinazioni;
- **elaborazione di big data**, permette di gestire e processare enormi volumi di dati in modo efficiente e scalabile;
- **messaggistica real time**, supporta la messaggistica in tempo reale tra applicazioni e sistemi distribuiti.

2.4.1.3 Impiego nel progetto

Il **broker** Redpanda gestisce i dati provenienti dai simulatori e li rende disponibili per i due consumatori. Inoltre, con lo *schema registry* integrato è possibile garantire la compatibilità tra i dati prodotti dai simulatori e i consumatori. I consumatori sono:

- Il **connector sink ClickHouse**, che salva i dati nelle tabelle di ClickHouse;
- **Apache Flink**, che elabora i dati in tempo reale.

2.4.2 ClickHouse

ClickHouse è un sistema di gestione di database colonnare *open-source* progettato per l'analisi dei dati in tempo reale e l'elaborazione di grandi volumi di dati.

- **Versione:** v24-alpine;
- **documentazione:** <https://clickhouse.com/docs/en/intro> [Ultima consultazione: 2024-06-02].



2.4.2.1 Vantaggi

I vantaggi nell'utilizzo di questo strumento consistono in:

- **alte prestazioni**, è progettato per eseguire query analitiche complesse in modo estremamente rapido;
- **scalabilità orizzontale**, può essere scalato orizzontalmente su più nodi, permettendo di gestire grandi volumi di dati;
- **elaborazione in tempo reale**, è in grado di gestire l'ingestione e l'elaborazione dei dati in tempo reale, rendendolo ideale per applicazioni che richiedono l'analisi immediata dei dati appena arrivano;
- **compressione efficiente**, utilizza algoritmi di compressione avanzati per ridurre lo spazio di archiviazione e migliorare l'efficienza I/O;
- **facilità di integrazione**, si integra facilmente con molti strumenti di visualizzazione dei dati e piattaforme di business intelligence come Grafana;
- **partizionamento e indici**, supporta il partizionamento dei dati e l'uso di indici per ottimizzare le query;

2.4.2.2 Casi d'uso

ClickHouse è utilizzato in una varietà di casi d'uso, tra cui:

- **analisi dei log e monitoraggio**, utilizzato per l'analisi e il monitoraggio dei log in tempo reale;
- **business intelligence**, impiegato in applicazioni di BI per eseguire analisi approfondite dei dati aziendali, supportando la presa di decisioni basata sui dati;
- **data warehousing**, funziona come data warehouse per memorizzare e analizzare grandi volumi di dati.

2.4.2.3 Impiego nel progetto

ClickHouse viene utilizzato per memorizzare i dati grezzi provenienti dai simulatori; attraverso il *connector sink* di Redpanda, i *record* pubblicati nei *topic* vengono salvati in tabelle di ClickHouse. Inoltre, tramite l'utilizzo di Materialized Views, vengono effettuate delle



semplici aggregazioni sui dati, come ad esempio la media oraria o giornaliera, le quali sono poi memorizzate in apposite tabelle, in modo da poterne monitorare l'andamento nel tempo.

Le aggregazioni più complesse che coinvolgono dati provenienti da sensori differenti sono invece effettuate utilizzando Apache Flink, come meglio descritto nella sezione 2.4.3.

ClickHouse si integra semplicemente con Grafana, attraverso l'utilizzo del plugin `datasource-clickhouse`, fornito da Grafana Labs.

2.4.3 Apache Flink

Apache Flink è un framework open-source per l'elaborazione dei dati in tempo reale e in *batch*. Sviluppato in Java e Scala, è progettato per gestire *data stream* in modo efficiente, consentendo l'elaborazione di grandi volumi di dati in tempo reale. Flink si distingue per la sua capacità di fornire elaborazione a bassa latenza, esecuzione *fault-tolerant* e scalabilità orizzontale.

- **Versione:** v1.18.1;
- **documentazione:** <https://flink.apache.org> [Ultima consultazione: 2024-06-25].

2.4.3.1 Vantaggi

- **Elaborazione a bassa latenza:** Flink è progettato per elaborare i dati in tempo reale con latenza estremamente bassa, rendendolo ideale per applicazioni che richiedono risposte rapide ai cambiamenti dei dati.
- **Fault Tolerance:** Flink utilizza una tecnologia chiamata *Stateful Stream Processing* che garantisce che lo stato dell'applicazione venga memorizzato in modo sicuro e possa essere recuperato in caso di guasti. Questo consente un'elaborazione affidabile e continua anche in presenza di errori hardware o software.
- **Scalabilità:** Flink può scalare orizzontalmente su cluster di grandi dimensioni, distribuendo il carico di lavoro tra molteplici nodi per gestire volumi di dati crescenti senza compromettere le prestazioni.
- **Modello di programmazione flessibile:** Flink offre due tipologie di API per l'elaborazione dei dati: *DataStream API*, utilizzato per l'elaborazione di flussi di dati non strutturati



in tempo reale, e Table API, un'astrazione di livello superiore per manipolare dati strutturati come tabelle, facilitando l'uso di operazioni simili a SQL;

- **Supporto per analisi complesse:** Flink fornisce potenti funzionalità di analisi come aggregazioni, join e *windowing*, che consentono di realizzare analisi complesse sui flussi di dati.

2.4.3.2 Casi d'uso

Tra i principali casi d'uso di Apache Flink si trovano:

- **Applicazioni *event-driven*:** applicazioni *stateful* che elaborano eventi provenienti da uno o più flussi di eventi e reagiscono agli eventi in ingresso attivando calcoli, aggiornamenti di stato o azioni esterne;
- ***data analytics*:** estrazione di informazioni e *insight* a partire dall'elaborazione dei dati grezzi, sia in *real time* che in modalità *batch*;
- ***data pipeline*:** ad esempio per la costruzione di *Extract-transform-load* (ETL) o integrazione di dati provenienti da sorgenti differenti.

2.4.3.3 Impiego nel progetto

Vengono utilizzate le *Data Streaming API* per elaborare ed aggregare dati provenienti da sensori di tipologie differenti. Nello specifico, per ciascuno degli indici in seguito elencati è stato sviluppato un *job*; i dettagli implementativi di ciascuno di essi sono meglio discussi nella sezione 3.5.

- **Heat Index:** una misura che combina la temperatura dell'aria e l'umidità relativa per determinare la temperatura percepita dall'uomo. Questa misura riflette meglio il livello di disagio che una persona potrebbe sperimentare rispetto alla sola temperatura dell'aria;
- **Efficienza delle colonnine elettriche**

2.4.4 Grafana

È una potente piattaforma di visualizzazione dei dati progettata per creare, esplorare e condividere *dashboard* interattive che visualizzano metriche, *log* e altri dati di monitoraggio in tempo reale.



- **Versione:** v10.3.0;
- **documentazione:** <https://grafana.com/docs/grafana/v10.4/> [Ultima consultazione: 2024-06-02].

2.4.4.1 Vantaggi

- **Facilità d'uso:** possiede un'interfaccia intuitiva che rende facile la creazione e la gestione delle dashboard;
- **flessibilità:** La capacità di integrarsi con molteplici sorgenti dati e l'ampia gamma di plugin disponibili la rendono estremamente flessibile;
- **personalizzazione:** permette una personalizzazione completa delle dashboard, soddisfacendo ogni possibile necessità di visualizzazione dei dati;
- **gestione degli accessi:** offre funzionalità avanzate di gestione degli accessi e delle autorizzazioni, consentendo di controllare chi può accedere alle *dashboard* e quali azioni possono eseguire.

2.4.4.2 Casi d'uso

- **Monitoraggio delle infrastrutture:** utilizzato per monitorare le prestazioni e la disponibilità delle infrastrutture IT, inclusi server, database, servizi cloud e altro;
- **analisi delle performance delle applicazioni:** utilizzato per monitorare le prestazioni delle applicazioni e identificare eventuali problemi di prestazioni;
- **analisi delle serie temporali:** utilizzato per visualizzare e analizzare dati di serie temporali, come metriche di monitoraggio, log e dati di sensori;
- **business intelligence:** utilizzato per creare *dashboard* personalizzate per l'analisi dei dati aziendali e la visualizzazione delle metriche chiave.

2.4.4.3 Impiego nel progetto

- **Visualizzazione dei dati:** creazione *dashboard* interattive che visualizzano i dati salvati su ClickHouse;



- **notifiche superamento soglie:** invio di notifiche nel caso in cui vengano superate delle soglie prestabilite, che rappresentano situazioni di eventuale pericolo, forte disagio o disservizio per i cittadini.



3 Architettura di sistema

3.1 Data processing architectures

Le architetture di tipo *data processing* sono progettate per gestire l'*ingestion*, *processing* e memorizzazione di grandi quantità di dati. Esse permettono di analizzare e ottenere informazioni utili (*insight*) da questi dati, consentendo di ottimizzare i processi decisionali e migliorare le prestazioni aziendali. Esistono diverse architetture, ciascuna con le proprie caratteristiche e vantaggi. Tra le più comuni troviamo l'architettura *lambda* e l'architettura *kappa*.

3.1.1 Architettura *lambda*

L'architettura *lambda* è costituita dalle seguenti quattro componenti:

- **sorgente di dati**: responsabile dell'acquisizione dei dati grezzi da diverse sorgenti;
- **batch layer**: responsabile dell'elaborazione e persistenza di dati storici in *batch* di grandi dimensioni; il suo scopo è fornire risposte complete e accurate, anche se con una latenza più elevata rispetto allo *speed layer*. Tale componente è tipicamente rappresentata da *framework* come Apache Hadoop o Apache Spark;
- **speed (real-time) layer**: responsabile dell'elaborazione e persistenza di dati in tempo reale. I dati vengono elaborati in modo rapido e con una latenza molto bassa, fornendo tuttavia risposte elaborate rispetto al *batch layer*. Questa componente è tipicamente rappresentata da *framework* come Apache Storm o Apache Flink;
- **serving layer**: responsabile della fornitura dei dati elaborati in modo veloce ed affidabile, indipendentemente dal *layer* di elaborazione utilizzato.

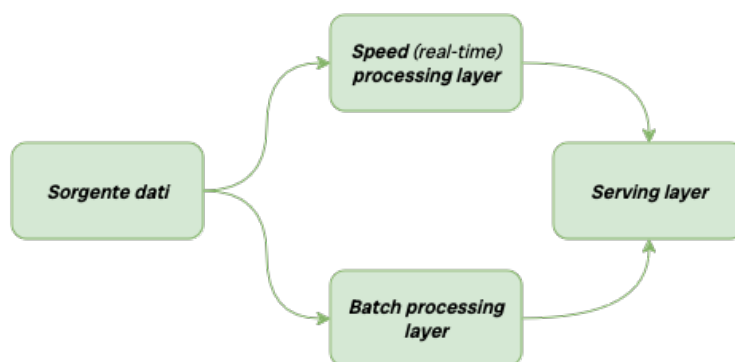


Figura 1: Architettura *lambda*

3.1.1.1 Vantaggi e svantaggi

L'architettura *lambda* offre diversi vantaggi, tra cui la **scalabilità orizzontale**, la **tolleranza ai guasti** e la **flessibilità**. Tuttavia, la presenza di due *layer* di elaborazione separati può portare a problemi di coerenza dei dati, duplicazione della logica di aggregazione e complessità aggiuntiva nella gestione del sistema. Inoltre, rispetto all'architettura *kappa*, l'architettura *lambda* può avere una latenza più elevata.

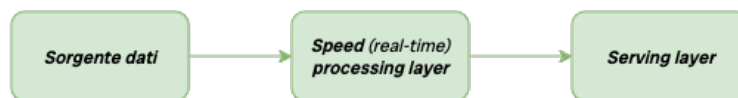
3.1.1.2 Casi d'uso

L'architettura *lambda* è particolarmente adatta per applicazioni che richiedono sia un'analisi sui dati in tempo reale che un'analisi storica.

3.1.2 Architettura *kappa*

L'architettura *kappa* è stata introdotta per semplificare l'architettura *lambda*, eliminando la necessità di gestire due *layer* di elaborazione separati per il *batch processing* e il *real-time processing*. Essa si divide in tre componenti principali:

- **sorgente di dati**: responsabile dell'acquisizione dei dati grezzi da diverse sorgenti;
- **processing layer**: responsabile dell'elaborazione dei dati in tempo reale, senza la necessità di separare i dati in *batch* e *real-time*;
- **serving layer**: responsabile della fornitura dei dati elaborati in modo veloce ed affidabile.

Figura 2: Architettura *kappa*

3.1.2.1 Vantaggi e svantaggi

L'architettura *kappa* offre diversi vantaggi, tra cui la **semplicità**, la **riduzione dei costi** e la **bassa latenza**. Tuttavia, può non essere adatta per applicazioni che richiedono un'analisi storica dei dati.



3.1.2.2 Casi d'uso

L'architettura *kappa* è particolarmente adatta per gli scenari in cui sono critici i dati in tempo reale e l'analisi dei dati storici è meno importante. Inoltre, semplifica notevolmente il processo di sviluppo e manutenzione dei sistemi di elaborazione dei dati.

3.2 Architettura scelta

Nello scenario del capitolato proposto da *SyncLab S.r.L.*, è critica l'analisi in tempo reale, in quanto i dati provenienti dai sensori IoT devono fornire informazioni sempre aggiornate ed eventualmente sollevare allarmi in caso di situazioni critiche. Inoltre, non è richiesta l'aggregazione storica di dati, dunque i vantaggi dell'architettura *lambda* non risultano utili per i nostri fini. Per soddisfare tali requisiti, è stata dunque scelta l'architettura *kappa*.

3.2.1 Componenti di sistema

All'interno del sistema da noi progettato sono dunque presenti le seguenti componenti:

- **sorgenti di dati:** costituite dal simulatore di sensori, il quale genera i dati grezzi che in un contesto reale sarebbero provenienti dai sensori IoT;
- **streaming layer:** gestisce il flusso di dati in tempo reale provenienti dai sensori. È composto da *Redpanda* e lo *Schema Registry*;
- **processing layer:** elabora i dati in tempo reale per calcolare metriche e indici. È composto da *Apache Flink*;
- **storage layer:** memorizza i dati elaborati per l'analisi e la visualizzazione. È composto da *ClickHouse*;
- **data visualization layer:** fornisce un'interfaccia utente per visualizzare i dati elaborati. È composto da *Grafana*.

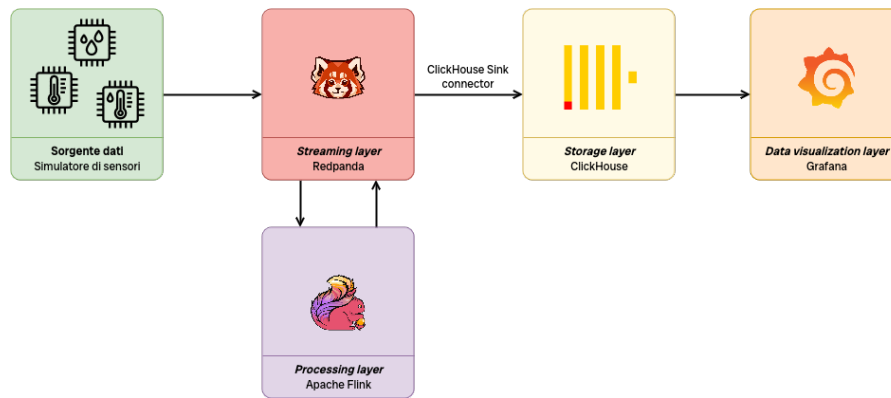


Figura 3: Componenti di sistema ad alto livello

3.2.2 Flusso di dati

Per illustrare il flusso di dati all'interno del sistema, è stato realizzato il seguente diagramma, il quale mostra il percorso che i dati grezzi seguono dal simulatore fino alla visualizzazione tramite Grafana.

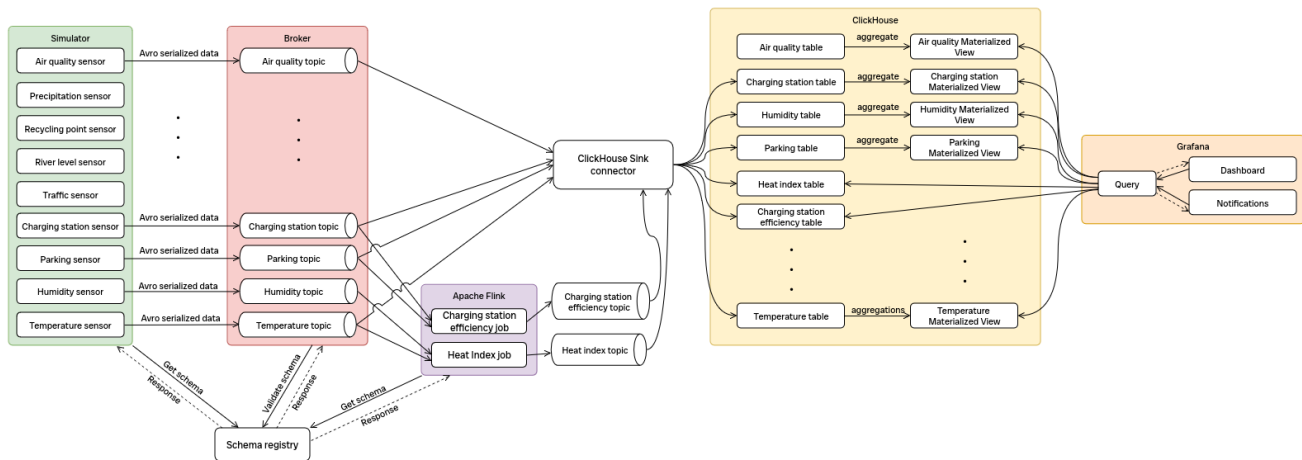


Figura 4: Flusso di dati all'interno del sistema. I sensori di precipitazioni, isole ecologiche, livello dei fiumi e traffico sono stati omessi per chiarezza, ma il percorso di tali dati è analogo a quello dei sensori di qualità dell'aria.

Il flusso seguito dai dati si può riassumere nei seguenti passaggi:

1. **generazione dei dati:** ciascun simulatore emula il comportamento di un singolo sensore IoT, generando ad intervalli periodici o ad eventi (*event-driven*) i dati grezzi relativi alla propria tipologia di dato.



2. **serializzazione e produzione dei dati:** i dati grezzi generati nel punto precedente vengono serializzati utilizzando il formato Confluent Avro e inviati nel *topic* corrispondente al tipo di dato generato;
3. **elaborazione dei dati:** i *topic* contenenti i dati grezzi di temperatura, umidità, occupazione dei parcheggi e colonnine di ricarica vengono consumati da Apache Flink. Due *job* distinti si occupano di calcolare la temperatura percepita e il grado di efficienza delle colonnine di ricarica. Una volta elaborati, i dati vengono inviati rispettivamente nei *topic* `heat_index` e `charging_station_efficiency`;
4. **memorizzazione dei dati:** attraverso il connettore *sink* per ClickHouse, i dati pubblicati in tutti i *topic* vengono memorizzati nel database;
5. **aggregazioni con *materialized view*:** attraverso l'utilizzo di *materialized view* in ClickHouse, vengono calcolate le statistiche relative ai dati memorizzati, come ad esempio la media oraria o giornaliera. Tali aggregazioni sono più semplici rispetto a quelle effettuate da Flink, in quanto non richiedono elaborazioni complesse sui dati;
6. **visualizzazione dei dati:** i dati memorizzati in ClickHouse vengono visualizzati tramite Grafana, che permette di creare *dashboard* personalizzate per monitorare i dati in tempo reale;
7. **notifiche:** Grafana esegue periodicamente delle *query* per verificare se sono state superate delle soglie predeterminate. In caso affermativo, vengono inviate notifiche tramite il canale Discord dedicato, in modo tale da poter avvisare l'autorità locale.

3.3 Architettura dei simulatori

I simulatori vengono utilizzati per produrre dati grezzi che sostituiscono le rilevazioni effettuate dai sensori IoT in un contesto reale. Per tale motivo, questa parte del sistema non è ufficialmente parte del prodotto finale, ma è stata sviluppata per scopi di *test* e dimostrativi nell'ambito del progetto didattico; ai fini di quest'ultimo, il gruppo ha deciso di dedicare alcune risorse per la progettazione.

Nei paragrafi successivi verranno descritti i moduli che compongono i simulatori, le classi e metodi principali e i *design pattern* utilizzati.

Sono stati implementati simulatori per i seguenti tipi di dato:

- qualità dell'aria;



- precipitazioni;
- isole ecologiche;
- livello dei fiumi;
- traffico;
- colonnine di ricarica;
- parcheggi;
- temperatura;
- umidità.

3.3.1 Modulo `models`

Questo modulo contiene le classi che rappresentano i dati grezzi generati dai sensori (`raw_data`) e la configurazione dei sensori stessi (`config`) letta dal file di configurazione `sensors.toml` e dalle variabili d'ambiente.

Ciascun tipo di dato grezzo è rappresentato da una classe che estende la classe astratta `RawData`, la quale contiene i campi comuni a tutti i tipi di dato, i metodi *helper* `topic` e `value_subject()` e il metodo `to_json()` per la serializzazione in formato JSON.

La classe `SensorConfig` riceve nel costruttore la configurazione sotto forma di dizionario, effettua *parsing*, validazione, popola con valori di *default* i campi mancanti (nel caso lo prevedano) ed inizializza i propri attributi, corrispondenti ai campi del file di configurazione.

Allo stesso modo, `EnvConfig` legge le variabili d'ambiente ed espone il metodo `bootstrap_server()` che combina *host* e *port* per formare l'indirizzo del *broker* Kafka.

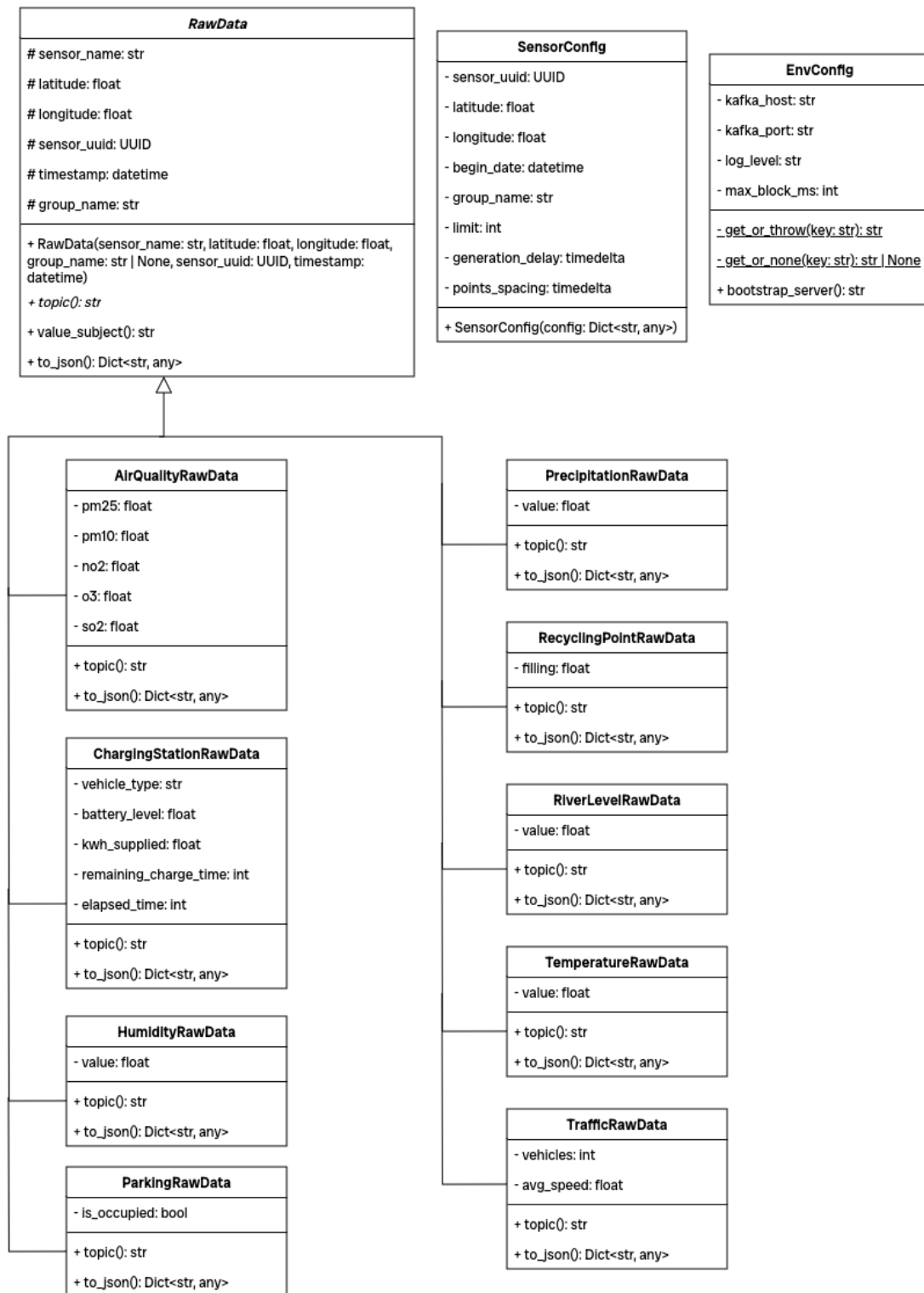


Figura 5: Diagramma delle classi del modulo models



3.3.2 Modulo producers

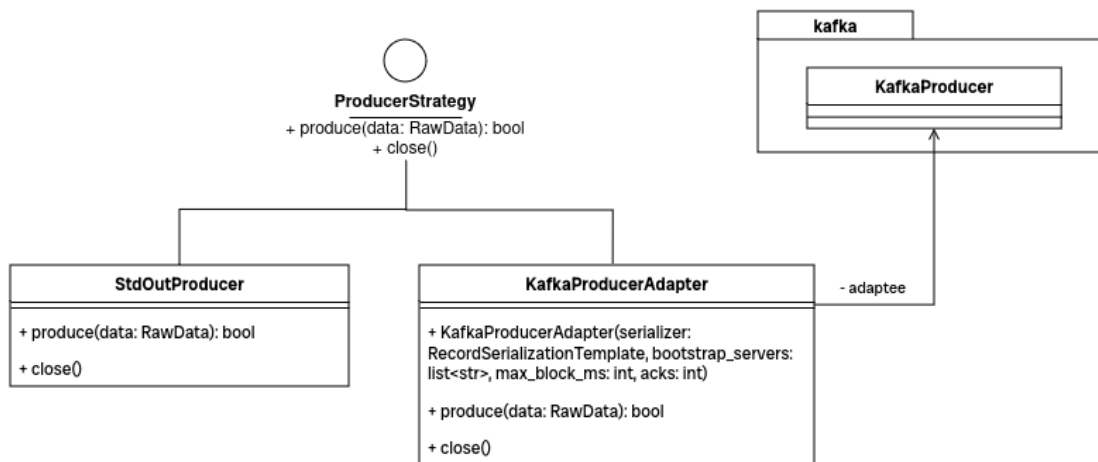


Figura 6: Diagramma delle classi del modulo producers

3.3.3 Modulo serializers

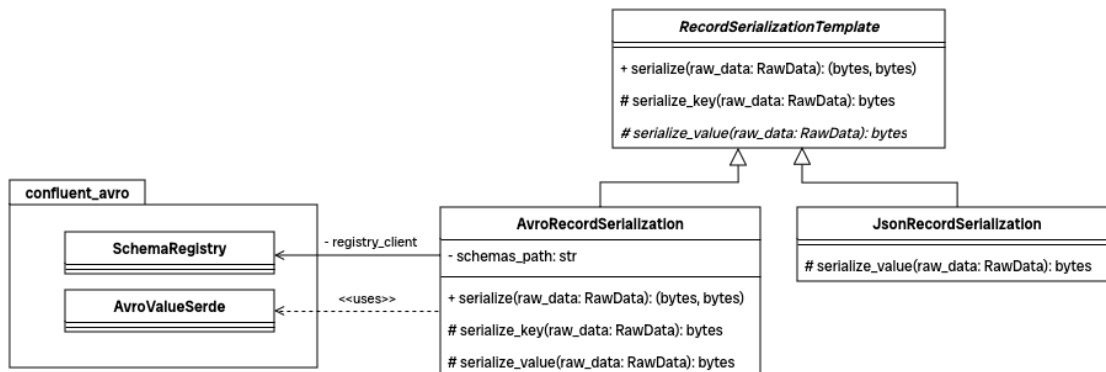


Figura 7: Diagramma delle classi del modulo serializers

3.3.4 Modulo simulators

3.4 Redpanda

3.4.1 Topic

Nel contesto di Redpanda, un *topic* è una categoria o canale a cui vengono inviati i dati. Essi sono utili per organizzare logicamente i diversi tipi di messaggi o eventi. Nel nostro sistema, i dati grezzi provenienti dai simulatori vengono pubblicati in un *topic*



differente per ciascun tipo di dato; ciò consente di elaborare in modo indipendente le varie tipologie di messaggi.

3.4.2 Partizioni e chiavi

I *topic* possono essere suddivisi in più partizioni, le quali consentono la distribuzione del carico di lavoro tra più *broker* Redpanda, allo scopo di migliorare le prestazioni e la scalabilità. Ciascuna partizione di un *topic* viene memorizzata in diversi nodi del *cluster*; la numerosità delle partizioni può essere configurata a seconda delle necessità.

Redpanda garantisce l'ordine degli eventi all'interno della stessa partizione, tuttavia di *default* non è garantito l'ordine tra partizioni diverse. Il partizionamento consente di elaborare i dati in parallelo, infatti i consumatori possono leggere da più partizioni contemporaneamente, distribuendo il carico computazionale e migliorando il *throughput*. Ogni messaggio pubblicato è detto *record* e ha una chiave, che può essere utilizzata per determinare la partizione a cui il messaggio verrà assegnato, ed un valore, che costituisce il vero e proprio *payload*; eventi con la stessa chiave vengono inviati alla stessa partizione.

Nel caso del nostro progetto, abbiamo deciso di utilizzare come chiave il `sensor_uuid`, un identificativo univoco globale per ciascun sensore, affinché i dati siano inviati alla stessa partizione e conseguentemente elaborati nell'ordine in cui sono stati prodotti.

3.4.3 Redpanda schema registry

Lo *schema registry* offre un archivio centralizzato per gestire e convalidare gli schemi associati ai messaggi Kafka, facilitandone la serializzazione e deserializzazione. I produttori e consumatori dei *topic* Kafka possono utilizzare questi schemi per garantire coerenza e compatibilità dei dati durante la loro evoluzione nel tempo.

3.4.3.1 *Compatibility mode*

Le modalità di compatibilità di uno schema sono delle regole che determinano come i cambiamenti ad uno schema influiscono sulla capacità dei dati serializzati con versioni precedenti di essere letti da versioni successive e viceversa; sono essenziali per garantire che i dati rimangano compatibili durante l'evoluzione degli schemi. Di seguito sono descritte le principali modalità di compatibilità supportate dallo *schema registry* di Redpanda:

- **BACKWARD**: i consumatori che utilizzano lo schema più recente possono leggere i dati prodotti con lo schema precedente;



- **BACKWARD_TRANSITIVE**: i consumatori che utilizzano lo schema più recente possono leggere i dati prodotti con tutti gli schemi precedenti;
- **FORWARD**: i consumatori che utilizzano lo schema precedente possono leggere i dati prodotti con lo schema più recente;
- **FORWARD_TRANSITIVE**: i consumatori che utilizzano uno qualsiasi degli schemi precedenti possono leggere i dati prodotti con lo schema più recente;
- **FULL**: i dati prodotti con lo schema più recente possono essere letti da consumatori che utilizzano lo schema precedente e viceversa;
- **FULL_TRANSITIVE**: i dati prodotti con uno qualsiasi degli schemi possono essere letti da consumatori che utilizzano qualsiasi altro schema;
- **NONE**: nessun controllo di compatibilità viene effettuato.

Nel progetto proposto da *SyncLab S.r.L.* l'obiettivo principale è l'elaborazione dei dati in tempo reale piuttosto che di quelli storici, pertanto è importante che i consumatori possano sempre ricevere i messaggi più recenti, anche se prodotti con un nuovo schema. Pertanto questo tipo di applicazioni beneficiano della modalità **FORWARD**, ovvero quella che abbiamo scelto di utilizzare.

3.4.3.2 Serializzazione dei dati

3.4.3.2.1 Chiavi

Come menzionato in precedenza, si utilizza il `sensor_uuid` per popolare il campo chiave dei *record*; tale identificativo viene, prima di essere pubblicato nel *topic*, convertito a stringa e codificato in UTF-8.

3.4.3.2.2 Valori

Il formato Avro consente di definire attraverso JSON uno schema che descriva la struttura dei dati, permettendo di serializzare e deserializzarli in modo affidabile; la serializzazione tramite Avro produce dati binari compatti, che consentono di ridurre l'*overhead* di rete e migliorare le prestazioni di trasmissione dei dati. Solamente i dati che rispettano lo schema definito possono essere inviati nel *topic*, garantendo la coerenza dei dati e facilitando la gestione delle evoluzioni dello schema.



Per la serializzazione dei valori abbiamo stabilito di utilizzare il formato Confluent Avro; la principale differenza rispetto al formato Avro standard è l'inclusione di un *magic byte* e dell'ID dello schema all'inizio del messaggio, seguiti dal *payload* vero e proprio. Ciò consente di evitare di includere lo schema all'interno di ogni messaggio, riducendo la dimensione dei dati trasmessi.

Il produttore consulta lo *schema registry* per ottenere l'ID corretto da utilizzare quando invia un messaggio, mentre il consumatore lo utilizza per ottenere lo schema con cui deserializzare il messaggio.

3.4.3.3 Formato dei dati

3.4.3.4 Topic name strategy

3.4.4 Redpanda Connect

Redpanda Connect è una piattaforma integrata nel sistema Redpanda, progettata per facilitare l'integrazione e il trasferimento dei dati tra Redpanda e altre fonti o destinazioni. Esso consente di gestire dei connettori, componenti *software* che si occupano automatizzare lo spostamento dei dati da e verso Redpanda. Tali connettori si dividono in due categorie:

- **source connector**: si occupano di trasferire i dati da una sorgente esterna a Redpanda;
- **sink connector**: si occupano di trasferire i dati da Redpanda a una destinazione esterna.

3.4.4.1 Sink connector per ClickHouse

All'interno del progetto abbiamo utilizzato Redpanda Connect per persistere su ClickHouse i dati provenienti dai sensori pubblicati nei differenti *topic*. Per poter effettuare questa operazione è stato necessario utilizzare un *sink connector*, che si occupasse di deserializzare i messaggi in formato Confluent Avro, effettuare il *parsing* dei campi di tipo `DateTime` (pubblicati come stringhe in formato ISO 8601) ed infine salvare i dati in ClickHouse. La documentazione relativa è consultabile al seguente [url](#) [Ultima consultazione 2024-07-10].

La configurazione di tale connettore è disponibile all'interno del *repository* del progetto al percorso `redpanda/connectors/configs/clickhouse.json`. Al fine di effettuare il *parsing* delle date è stato necessario definire all'interno di tale file un *transformer*, il quale si occupa di leggere il campo `timestamp` e convertirlo in `DateTime`, tipo riconosciuto da



ClickHouse.

La versione utilizzata è la 1.1.1, scaricabile dal seguente [url](#) [Ultima consultazione 2024-07-10].

La configurazione del *transformer* è la seguente:

```
{
  //...
  "transforms": "TimestampConverter",
  "transforms.TimestampConverter.type": "org.apache.kafka.connect
    .transforms.TimestampConverter$Value",
  "transforms.TimestampConverter.format": "yyyy-MM-dd'T'HH:mm:ss
    ",
  "transforms.TimestampConverter.field": "timestamp",
  "transforms.TimestampConverter.target.type": "Timestamp"
  //...
}
```

Listing 1: Configurazione del *transformer* all'interno del file `clickhouse.json`

La creazione del connettore viene effettuata eseguendo il seguente comando nella radice del progetto:

```
curl "localhost:8083/connectors" -H 'Content-Type: application/json' \
  -d @./redpanda/connectors/configs/clickhouse.json
```

3.4.4.2 Avro converter

È stato inoltre necessario utilizzare un ulteriore *plugin*, *avro-converter* (versione 7.6.1), scaricabile dal seguente [url](#) [Ultima consultazione 2024-07-10], il quale consente di effettuare la deserializzazione dei messaggi in formato Confluent Avro.

Il *connector sink* è stato configurato per utilizzarlo come segue:

```
{
  //...
  "value.converter": "io.confluent.connect.avro.AvroConverter",
  "value.converter.schemas.enable": "true",
  "value.converter.schema.registry.url": "http://redpanda:8081",
  //...
}
```



Listing 2: Utilizzo del *plugin* `avro-converter` all'interno del file `clickhouse.json`

3.4.5 Redpanda Console

Redpanda Console è un'applicazione web che consente di gestire e effettuare *debug* di un'istanza Redpanda. Essa ha diverse funzionalità, tra cui:

- **Visualizzazione dei messaggi:** consente di esplorare i messaggi dei *topic* attraverso query ad-hoc e filtri dinamici, scritti con semplici funzioni JavaScript;
- **Gruppi di consumatori:** permette di visualizzare tutti i gruppi di consumatori attivi, insieme ai relativi offset, modificarli o eliminarli;
- **Panoramica dei *topic*:** permette di visualizzare la lista dei *topic*, controllarne la configurazione, lo spazio utilizzato, la lista dei consumatori e i dettagli delle partizioni;
- **Panoramica del *cluster*:** permette di visualizzare le ACL, i *broker* disponibili, il loro spazio utilizzato, l'ID del rack e altre informazioni per ottenere una panoramica del *cluster*;
- **Schema Registry:** permette di visualizzare tutti gli schemi Avro, Protobuf o JSON all'interno del registro degli schemi;
- **Kafka Connect:** permette di gestire i connettori da più *cluster* di connessione, modificare le configurazioni, visualizzare lo stato corrente o riavviare i task.

3.5 Flink - *Processing layer*

3.5.1 Introduzione

È un *framework* di elaborazione dati distribuito e *open-source* che si distingue per la sua capacità di gestire sia dati di flusso in tempo reale che dati *batch*. Una delle sue caratteristiche principali è la capacità di gestire dati in tempo reale con latenze molto basse. Ciò significa che può elaborare i dati man mano che arrivano, consentendo alle applicazioni di reagire istantaneamente ai cambiamenti nell'*input*. Questa caratteristica è particolarmente importante per le applicazioni che richiedono analisi in tempo reale, come il monitoraggio di sensori, il rilevamento di anomalie o la personalizzazione di contenuti.



3.5.2 Componenti Flink e *processing layer*

È costituito da diverse componenti fondamentali che lavorano insieme per consentire l'elaborazione efficiente e scalabile dei dati in tempo reale e batch. Queste componenti formano il cuore del sistema Flink e forniscono le basi per la sua potente capacità di elaborazione dei dati.

3.5.3 *Processing layer data-flow*

3.5.4 *Job*

3.5.4.1 *Heat Index*

A partire dai dati rilevati dai sensori di temperatura e umidità; esso consente di stimare la percezione della temperatura da parte dell'essere umano. Nella configurazione del Simulatore, oltre a posizione e identificativo per ciascun sensore, è possibile specificare un `group_name`, una stringa che identifica il gruppo o zona di appartenenza; si suppone che sensori situati in posizioni geografiche vicine abbiano lo stesso `group_name`. Il *job* calcola prima separatamente la temperatura e l'umidità media per finestre di un'ora, aggregando i dati provenienti da sensori dello stesso gruppo. Successivamente con i valori ottenuti computa lo *Heat Index*, utilizzando la formula empirica ideata da Blazejczyk [Ultima consultazione 2024-06-25]. Nel risultato finale, oltre al valore dello *Heat Index*, vengono restituiti anche i valori di temperatura e umidità medi, il centro di massa del gruppo di sensori (utilizzando la formula Haversine [Ultima consultazione 2024-06-25] per il calcolo della distanza) e la distanza dal centro di massa al sensore più lontano. Questi ultimi due dati sono impiegati in una mappa interattiva su Grafana per poter disegnare un cerchio, rappresentante la zona di influenza del gruppo di sensori.

3.5.4.2 *Efficienza delle colonnine elettriche*

3.6 Database ClickHouse

3.6.1 Funzionalità utilizzate

3.6.1.1 *Materialized View*

Sono una potente funzionalità di *ClickHouse* per migliorare le prestazioni delle query e semplificare l'analisi dei dati. In sostanza, una *materialized view* è una vista pre-



calcolata o una copia di una query, memorizzata fisicamente su disco in forma tabellare. Ciò consente di evitare il calcolo ripetuto dei risultati della query ogni volta che viene eseguita.

Documentazione

<https://clickhouse.com/docs/en/guides/developer/cascading-materialized-views>
[Ultima consultazione 2024-06-05]

Utilizzi

3.6.1.2 MergeTree

MergeTree è uno dei principali motori di archiviazione di ClickHouse, progettato per gestire grandi volumi di dati e fornire elevate prestazioni di lettura e scrittura. È particolarmente adatto per applicazioni in cui i dati vengono aggiunti in modo incrementale e le query vengono eseguite su intervalli di tempo specifici. Le caratteristiche principali sono:

- **partizionamento**, in cui i dati vengono partizionati in base a una colonna di data o di tempo, in modo che i dati più recenti siano memorizzati in partizioni separate e possano essere facilmente eliminati o archiviati;
- **ordine dei dati**, dove i dati vengono ordinati in base a una colonna di ordinamento, in modo che i dati siano memorizzati in modo sequenziale e possano essere letti in modo efficiente;
- **indice primario**, tramite il quale i dati vengono indicizzati in base a una colonna di chiave primaria, in modo che le query di ricerca e di join siano veloci ed efficienti;
- **merging dei dati**, in questo modo i dati vengono uniti in modo incrementale in background, in modo che le query di aggregazione e di analisi siano veloci ed efficienti;
- **compressione**, i dati vengono compressi in modo efficiente per ridurre lo spazio di archiviazione e migliorare le prestazioni di lettura e scrittura;
- **replica e distribuzione**, i dati possono essere replicati e distribuiti su più nodi per garantire l'affidabilità e la disponibilità del sistema.



Documentazione

<https://clickhouse.com/docs/en/engines/table-engines/mergetree-family/mergetree>
[Ultima consultazione 2024-06-05]

3.6.2 Trasferimento dati tramite Materialized View

Le Materialized View in ClickHouse sono viste che memorizzano fisicamente i risultati di una query specifica in modo da permettere un accesso rapido e efficiente ai dati pre-elaborati. Quando vengono create, le Materialized View eseguono la query definita e archiviano i risultati in una struttura di dati ottimizzata per l'accesso veloce. Questo consente di evitare il calcolo ripetuto dei risultati della query ogni volta che viene eseguita, migliorando notevolmente le prestazioni complessive del sistema. I vantaggi derivanti da questo approccio sono molteplici, tra questi troviamo:

- **prestazioni ottimizzate:** grazie alla memorizzazione fisica dei risultati delle query, le Materialized View consentono un accesso rapido ai dati pre-elaborati, riducendo i tempi di risposta delle query complesse;
- **riduzione del carico di lavoro:** trasferendo i dati pre-elaborati in Materialized View, si riduce il carico di lavoro sul sistema sorgente, consentendo una maggiore scalabilità e riducendo il rischio di sovraccarico del sistema durante le operazioni di estrazione dei dati;
- **sempre aggiornate:** possono essere progettate per aggiornarsi automaticamente in risposta alle modifiche nei dati sottostanti, garantendo che i risultati siano sempre aggiornati e coerenti con lo stato attuale dei dati;
- **semplificazione dell'architettura:** è possibile semplificare l'architettura complessiva del sistema eliminando la necessità di eseguire query complesse e costose ogni volta che si accede ai dati.

3.6.3 Misurazioni isole ecologiche

Di seguito viene riportata la configurazione della tabella per le misurazioni delle isole ecologiche. Le misurazioni includono:

- **sensor_uuid:** identificativo univoco del sensore (formato UUID);
- **sensor_name:** nome del sensore (formato String);



- **timestamp**: data e ora della misurazione (formato DateTime64);
- **latitude**: latitudine del sensore (formato Float64);
- **longitude**: longitudine del sensore (formato Float64);
- **filling_value**: percentuale di riempimento dell'isola ecologica (formato Float32).

3.6.4 Misurazioni temperatura

Di seguito viene riportata la configurazione della tabella per le misurazioni della temperatura. Le misurazioni includono:

- **sensor_uuid**: identificativo univoco del sensore (formato UUID);
- **sensor_name**: nome del sensore (formato String);
- **timestamp**: data e ora della misurazione (formato DateTime64);
- **value**: valore della temperatura rilevata (formato Float32);
- **latitude**: latitudine del sensore (formato Float64);
- **longitude**: longitudine del sensore (formato Float64);

3.6.5 Misurazioni traffico

Di seguito viene riportata la configurazione della tabella per le misurazioni della traffico. Le misurazioni includono:

- **sensor_uuid**: identificativo univoco del sensore (formato UUID);
- **sensor_name**: nome del sensore (formato String);
- **timestamp**: data e ora della misurazione (formato DateTime64);
- **latitude**: latitudine del sensore (formato Float64);
- **longitude**: longitudine del sensore (formato Float64);
- **vehicles**: numero di veicoli rilevati (formato Int32);
- **avg_speed**: velocità media del traffico (formato Float32).



3.7 Grafana

Grafana è uno strumento di analisi e monitoraggio che permette di visualizzare dati provenienti da una varietà di fonti. È sviluppato principalmente in Go e Typescript ed è noto per la sua capacità di creare dashboard personalizzabili e intuitive.

3.7.1 Dashboard

3.7.2 ClickHouse datasource plugin

Il plugin ClickHouse per Grafana è un'implementazione che consente di utilizzare ClickHouse come fonte di dati per Grafana. Questo plugin facilita la connessione e l'interrogazione dei dati archiviati in ClickHouse direttamente da Grafana, permettendo di creare dashboard dinamiche e interattive.

Documentazione

<https://grafana.com/grafana/plugins/grafana-clickhouse-datasource/>
[Ultima consultazione 2024-06-05]

3.7.2.1 Configurazione del Datasource

La configurazione grafana/provisioning/datasources/default.yaml

3.7.3 Variabili Grafana

3.7.3.1 Documentazione

<https://grafana.com/docs/grafana/latest/dashboards/variables/>
[Ultima consultazione 2024-06-05]

Variabili nella dashboard principale

Le variabili presenti nella dashboard principale sono:

- **tipo sensore:** permette di selezionare il tipo di sensore da visualizzare (temperatura, traffico, isola ecologica...);
- **nome sensore:** permette di selezionare il nome del sensore da visualizzare (ad esempio sensore1, sensore2, ecc.);



Variabili nella dashboard dettagliata

Le variabili presenti nelle dashboard dettagliate sono:

- **nome sensore:** permette di selezionare il nome del sensore da visualizzare (es. sensore1, sensore2, ecc.);

3.7.4 Grafana Alerts

Sono una funzionalità che permettono di definire, configurare e gestire avvisi basati su condizioni specifiche rilevate nei dati monitorati. Questi avvisi consentono agli utenti di essere informati tempestivamente su eventuali problemi o cambiamenti critici nei loro sistemi, applicazioni o infrastrutture.

Documentazione

<https://grafana.com/docs/grafana/latest/alerting/> [Ultima consultazione 2024-06-05]

3.7.4.1 Configurazione delle regole di alert

Definiscono le condizioni che devono essere soddisfatte per attivare un alert. Gli eventi che generano un alert sono:

- temperatura maggiore di 40°C per più di 30 minuti;
- isola ecologica piena al 100% per più di 24 ore;
- superamento dell'indice 3 dell'EAQI (indice di qualità dell'aria);
- livello di precipitazioni superiore a 10 mm in 1 ora.

Gli alert possono possedere tre diversi tipi di stati:

- **normal**, indica che l'alert non è attivo perché le condizioni definite per l'attivazione dell'avviso non sono soddisfatte;
- **pending**, indica che le metriche monitorate stanno iniziando a deviare dalle condizioni normali ma non hanno ancora soddisfatto completamente le condizioni per attivare l'alert;
- **firing**, significa che le condizioni definite per l'avviso sono state soddisfatte e l'alert è attivo.



3.7.4.2 Configurazione canale di notifica

Per configurare un canale di notifica è necessario:

1. nel menù di sinistra, cliccare sull'icona "Alerting";
2. selezionare la voce "Notification channels";
3. cliccare sul pulsante "Add channel" per aggiungere un nuovo canale di notifica;
4. selezionare il tipo di canale di notifica desiderato tra quelli disponibili;
5. configurare le impostazioni del canale di notifica in base alle proprie esigenze;
6. cliccare sul pulsante "Save" per salvare le impostazioni del canale di notifica.

3.7.5 Altri plugin

3.7.5.1 Orchestra Cities Map plugin

Progettato per facilitare la visualizzazione e l'analisi dei dati geospaziali all'interno di piattaforme di pianificazione urbana e sviluppo territoriale.

Le principali funzionalità offerte da questo plugin sono:

- **visualizzazione dei dati geospaziali:** consente agli utenti di visualizzare dati geografici, come mappe, strati di dati GIS (Geographic Information System), punti di interesse e altre informazioni territoriali;
- **interfaccia interattiva:** offre un'interfaccia utente intuitiva e interattiva che consente agli utenti di esplorare e interagire con i dati geospaziali in modo dinamico;
- **personalizzazione:** offre opzioni di personalizzazione per adattarsi alle esigenze specifiche dell'utente o dell'applicazione;
- **analisi dei dati:** oltre alla semplice visualizzazione dei dati geospaziali, il plugin può anche supportare funzionalità avanzate di analisi dei dati, come l'identificazione di cluster, la creazione di heatmap e l'esecuzione di analisi spaziali per identificare tendenze o pattern significativi nei dati territoriali;
- **integrazione:** è progettato per integrarsi facilmente con altre componenti dell'ecosistema Orchestra Cities e con altre piattaforme software di pianificazione urbana e sviluppo territoriale.



Documentazione

<https://grafana.com/grafana/plugins/orchestracities-map-panel/?tab=installation>
[Ultima consultazione 2024-06-05]



4 Architettura di deployment

Per implementare ed eseguire l'intero stack tecnologico e i livelli del modello architetturale, viene creato un ambiente *Docker* che riproduce la suddivisione e la distribuzione dei servizi. In particolare, per l'ambiente di produzione, sono stati creati i seguenti container:

- **Data feed**

- Container: **Simulator**;
- Descrizione: simula la generazione di dati;

- **Streaming layer**

- Container: **Redpanda**;
- Descrizione: definisce il flusso di dati in tempo reale;
- Componenti di supporto: schema registry;
- Porta: 18082.

- **Processing Layer**

- Container: **Flink**;
- Descrizione: pianifica, assegna e coordina l'esecuzione dei task di elaborazione dei dati su un cluster di nodi, garantendo prestazioni elevate, scalabilità e affidabilità nell'elaborazione dei dati.

- **Storage Layer**

- Container: **Clickhouse**;
- Descrizione: memorizza i dati;
- Porta: 8123.

- **Data Visualization Layer**

- Container: **Grafana**;
- Descrizione: visualizza i dati;
- Porta: 3000.



5 Requisiti

5.1 Requisiti funzionali

Codice	Importanza	Stato	Descrizione
RF-1	Obbligatorio	Soddisfatto	La parte <i>IoT</i> dovrà essere simulata attraverso tool di generazione di dati casuali che tuttavia siano verosimili.
RF-2	Obbligatorio	Soddisfatto	Il sistema dovrà permettere la visualizzazione dei dati in tempo reale.
RF-3	Obbligatorio	Soddisfatto	Il sistema dovrà permettere la visualizzazione dei dati storici.
RF-4	Obbligatorio	Soddisfatto	L'utente deve poter accedere all'applicativo senza bisogno di autenticazione.
RF-5	Obbligatorio	Soddisfatto	L'utente dovrà poter visualizzare su una mappa la posizione geografica dei sensori.
RF-6	Obbligatorio	Soddisfatto	I tipi di dati che il sistema dovrà visualizzare sono: temperatura, umidità, qualità dell'aria, precipitazioni, traffico, stato delle colonnine di ricarica, stato di occupazione dei parcheggi, stato di riempimento delle isole ecologiche e livello di acqua.
RF-7	Obbligatorio	Soddisfatto	I dati dovranno essere salvati su un database OLAP.
RF-8	Obbligatorio	Soddisfatto	I sensori di temperatura rilevano i dati in gradi Celsius
RF-9	Obbligatorio	Soddisfatto	I sensori di umidità rilevano la percentuale di umidità nell'aria.



Codice	Importanza	Stato	Descrizione
RF-10	Obbligatorio	Soddisfatto	I sensori livello acqua rilevano il livello di acqua nella zona di installazione
RF-11	Obbligatorio	Soddisfatto	I dati provenienti dai sensori dovranno contenere i seguenti dati: id sensore _G , data, ora e valore.
RF-12	Obbligatorio	Soddisfatto	Sviluppo di componenti quali widget _G e grafici per la visualizzazione dei dati nelle dashboard _G .
RF-13	Obbligatorio	Soddisfatto	Il sistema deve permettere di visualizzare una dashboard _G generale con tutti i dati dei sensori.
RF-14	Obbligatorio	Soddisfatto	Il sistema deve permettere di visualizzare una dashboard _G specifica per ciascuna categoria di sensori.
RF-15	Obbligatorio	Soddisfatto	Nella dashboard _G dei dati grezzi dovranno essere presenti: una mappa interattiva, un widget _G con il conteggio totale dei sensori divisi per tipo, una tabella contenente tutti i sensori e la data in cui essi hanno trasmesso l'ultima volta. Inoltre verranno mostrate delle tabelle con i dati filtrabili suddivisi per sensore _G e un grafico time series _G con tutti i dati grezzi.



Codice	Importanza	Stato	Descrizione
RF-16	Obbligatorio	Soddisfatto	Nella dashboard _G della temperatura dovranno essere visualizzati: un grafico time series _G , una mappa interattiva, la temperatura media, minima e massima di un certo periodo di tempo, la temperatura in tempo reale e la temperatura media per settimana e mese.
RF-17	Obbligatorio	Soddisfatto	Nella dashboard _G dell'umidità dovranno essere visualizzati: un grafico time series _G , una mappa interattiva, l'umidità media, minima e massima di un certo periodo di tempo e l'umidità in tempo reale.
RF-18	Obbligatorio	Soddisfatto	Nella dashboard _G della qualità dell'aria dovranno essere visualizzati: un grafico time series _G , una mappa interattiva, la qualità media dell'aria in un certo periodo e in tempo reale, i giorni con la qualità dell'aria migliore e peggiore in un certo periodo di tempo.
RF-19	Obbligatorio	Soddisfatto	Nella dashboard _G delle precipitazioni dovranno essere visualizzati: un grafico time series _G , una mappa interattiva, la quantità media di precipitazioni in un certo periodo e in tempo reale, i giorni con la quantità di precipitazioni maggiore e minore in un certo periodo di tempo.



Codice	Importanza	Stato	Descrizione
RF-20	Obbligatorio	Soddisfatto	Nella dashboard _G del traffico dovranno essere visualizzati: un grafico time series _G , il numero di veicoli e la velocità media in tempo reale, il calcolo dell'ora di punta sulla base del numero di veicoli e velocità media.
RF-21	Obbligatorio	Soddisfatto	Nella dashboard _G delle colonnine di ricarica dovranno essere visualizzati: una mappa interattiva contenente anche lo stato e il numero di colonnine di ricarica suddivise per stato in tempo reale.
RF-22	Obbligatorio	Soddisfatto	Nella dashboard _G dei parcheggi dovranno essere visualizzati: una mappa interattiva con il rispettivo stato di occupazione e il conteggio di parcheggi suddivisi per stato di occupazione in tempo reale.
RF-23	Obbligatorio	Soddisfatto	Nella dashboard _G delle isole ecologiche dovranno essere visualizzati: una mappa interattiva con il rispettivo stato di riempimento e il conteggio di isole ecologiche suddivise per stato di riempimento in tempo reale.
RF-24	Obbligatorio	Soddisfatto	Nella dashboard _G del livello di acqua dovranno essere visualizzati: un grafico time series _G , una mappa interattiva, il livello medio di acqua in un certo periodo e in tempo reale.



Codice	Importanza	Stato	Descrizione
RF-25	Obbligatorio	Soddisfatto	Nel caso in cui non ci siano dati visualizzabili, il sistema deve notificare l'utente mostrando un opportuno messaggio.
RF-26	Obbligatorio	Soddisfatto	I sensori di qualità dell'aria inviano i seguenti dati: <i>PM10</i> , <i>PM2.5</i> , <i>NO2</i> , <i>CO</i> , <i>O3</i> , <i>SO2</i> in $\mu g/m^3$.
RF-27	Obbligatorio	Soddisfatto	I sensori di precipitazioni inviano la quantità di pioggia caduta in mm.
RF-28	Obbligatorio	Soddisfatto	I sensori di traffico inviano il numero di veicoli rilevati e la velocità in km/h.
RF-29	Obbligatorio	Soddisfatto	Le colonnine di ricarica inviano lo stato di occupazione e il tempo mancante alla fine della ricarica (se occupate) o il tempo passato dalla fine dell'ultima ricarica (se libere).
RF-30	Obbligatorio	Soddisfatto	I sensori di parcheggio inviano lo stato di occupazione del parcheggio (1 se occupato, 0 se libero) e il timestamp dell'ultimo cambiamento di stato.
RF-31	Obbligatorio	Soddisfatto	Le isole ecologiche inviano lo stato di riempimento come percentuale.
RF-32	Obbligatorio	Soddisfatto	I sensori di livello di acqua inviano il livello di acqua in cm.
RF-33	Obbligatorio	Soddisfatto	Il sistema deve permettere di filtrare i dati visualizzati in base a un intervallo di tempo.
RF-34	Obbligatorio	Soddisfatto	Il sistema deve permettere di filtrare i dati visualizzati in base al sensore _G che li ha generati.



Codice	Importanza	Stato	Descrizione
RF-37	Obbligatorio	Soddisfatto	Deve essere implementato almeno un simulatore di dati.
RF-39	Obbligatorio	Soddisfatto	I simulatori devono produrre dei dati verosimili.
RF-40	Obbligatorio	Soddisfatto	Per ciascuna tipologia di sensore _G dev'essere sviluppata almeno una dashboard _G .
RF-50	Obbligatorio	Soddisfatto	Il sistema deve permettere di filtrare i dati visualizzati in base al tipo di sensore che li ha prodotti.

Tabella 3: Requisiti funzionali

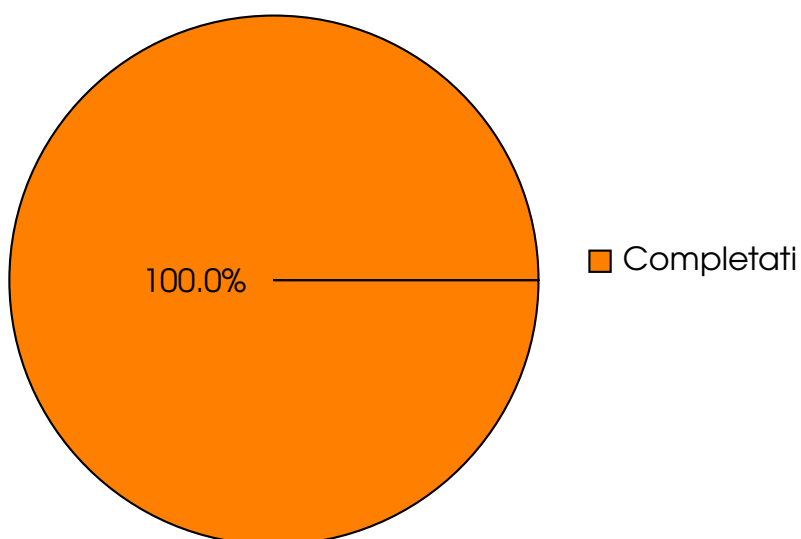


Figura 8: Percentuale di soddisfacimento dei requisiti funzionali

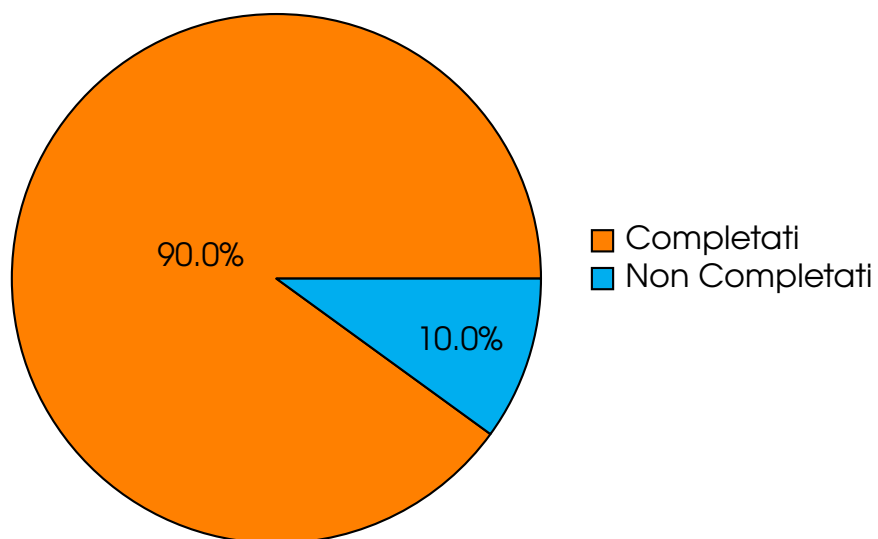


Figura 9: Percentuale di soddisfacimento dei requisiti totale