Specifica Tecnica

v0.1

7Last

Versioni

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

Indice

Elenco delle tabelle

Elenco delle figure

1 Introduzione

1.1 Scopo della specifica tecnica

Fornisce una descrizione dettagliata dei requisiti tecnici e delle funzionalità del sistema software in fase di sviluppo. È un punto di riferimento per gli sviluppatori e gli stakeholder, delineando le specifiche necessarie per la realizzazione del prodotto finale. Include una descrizione delle funzionalità del software, i requisiti hardware e software, le specifiche dei componenti, i criteri di performance e sicurezza, e le linee guida per la verifica e la validazione del sistema.

1.2 Scopo del prodotto

Lo scopo principale del prodotto è consentire a *Sync Lab S.r.l.* di valutare la fattibilità di investire tempo e risorse nell'implementazione del progetto ***SyncCity*** *- A smart city monitoring platform*. Questa soluzione, grazie all'utilizzo di dispositivi IoT, permette un monitoraggio costante delle città. SyncCity avrà l'obiettivo di raccogliere e analizzare dati provenienti da sensori posizionati nelle città, fornendo informazioni utili per la gestione urbana. Il prodotto finale sarà un prototipo funzionale che consentirà la visualizzazione dei dati raccolti su un cruscotto.

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

-
-

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 deployment 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

Ambiente di sviluppo locale

- Ambiente in cui gli sviluppatori lavorano sui propri computer;
- permette di creare e gestire i container localmente;
- utilizzato per testare le modifiche e sviluppare le funzionalità senza influenzare gli altri membri del gruppo.

Ambiente di sviluppo condiviso

- Ambiente utilizzato quando più membri del team devono collaborare su un progetto;
- è possibile utilizzare strumenti di orchestrazione come Docker Compose per definire un set di container che costituiscono l'applicazione in sviluppo;

- tutti i membri del team possono utilizzare lo stesso set di container per sviluppare e testare le loro modifiche.

Ambiente di test

- Questo ambiente viene utilizzato per testare l'applicazione in un ambiente simile a quello di produzione prima di rilasciarla;
- può includere una replica dell'infrastruttura di produzione, con configurazioni simili e dati di test;
- gli ambienti di test possono essere automaticamente aggiornati con le ultime versioni dell'applicazione dai rami di sviluppo tramite integrazione continua (CI) o integrazione continua/produzione (CI/CD).

Ambiente di produzione

- Questo è l'ambiente in cui viene eseguita effettivamente l'applicazione per gli utenti finali;
- gli ambienti di produzione sono solitamente più rigorosamente controllati e configurati rispetto agli ambienti di sviluppo e di test;
- le applicazioni vengono distribuite qui dopo essere state testate e validate in ambienti di test.

2.1.2 Docker images

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

- **Simulatore - Python**
  - Immagine: 3.11.9-alpine;
  - Riferimento: Python Official Docker Image (Consultato 2024-06-02).
  - Ambiente:
    - sviluppo;
    - produzione.

- **Broker - Redpanda**
  - Immagine: docker.redpanda.com/redpandadata/redpanda:v23.3.11;
  - Riferimento: Redpanda Official Docker Image (Consultato 2024-06-02).
  - Ambiente:
    * sviluppo;
    * produzione;
    * test.
- **Redpanda console**
  - Immagine: docker.redpanda.com/redpandadata/console:v2.4.6;
  - Riferimento: Redpanda Official Docker Image(Consultato 2024-06-02).
  - Ambiente:
    * sviluppo;
    * produzione;
    * test.
- **Database - ClickHouse**
  - Immagine: clickhouse/clickhouse-server:24-alpine;
  - Riferimento: ClickHouse Official Docker Image (Consultato 2024-06-02).
  - Ambiente:
    * sviluppo;
    * produzione;
    * test.
- **Grafana**
  - Immagine: grafana/grafana-oss:10.3.0;
  - Riferimento: Grafana Official Docker Image (Consultato 2024-06-02).
  - Ambiente:
    * sviluppo;
    * produzione.

2.2 Linguaggi e formato dati

2.2.1 Python

Python è un linguaggio di programmazione versatile, noto per la sua semplicità, leggibilità e vasta gamma di applicazioni.

2.2.1.1 Versione

- Versione utilizzata: 3.11.9

2.2.1.2 Documentazione

`https://docs.python.org/3.11/` (Consultato 2024-06-02).

2.2.1.3 Utilizzo nel progetto

- Sviluppo dei simulatori di sensori;
- sviluppo di moduli di elaborazione dei dati;
- automazione e testing.

2.2.1.4 Librerie

- **Astroid**:
  - Versione: 3.1.0;
  - Documentazione: `https://pypi.org/project/astroid/` (Consultato 2024-06-02);
  - Descrizione: utilizzata per rappresentare e manipolare l'Abstract Syntax Tree (AST) del codice Python. È ampiamente utilizzata da strumenti di analisi del codice come pylint per effettuare analisi statiche del codice Python. Astroid fornisce una rappresentazione astratta del codice sorgente che permette di esplorare e navigare la struttura del codice, compresi moduli, classi, funzioni e variabili, in modo dettagliato e programmabile.
- **Avro**:
  - Versione: 1.11.3;

  - Documentazione: `https://avro.apache.org/docs/1.11.1/getting-started-python/` (Consultato 2024-06-02);
  - Descrizione: è progettata per essere utilizzata da applicazioni che necessitano di una rappresentazione compatta, veloce e binaria dei dati, con un robusto supporto per l'evoluzione degli schemi dei dati. Avro è particolarmente utilizzata in ambienti di big data e di elaborazione distribuita.

- **Certifi**:
  - Versione: 2024.2.2;
  - Documentazione: `https://certifi.io/` (Consultato 2024-06-02);
  - Descrizione: è una libreria Python che fornisce una versione aggiornata del pacchetto di certificati CA (Certificate Authority) per la verifica dei certificati SSL/TLS.

- **Charset-normalizer**:
  - Versione: 3.3.2;
  - Documentazione: `https://charset-normalizer.readthedocs.io/en/stable/` (Consultato 2024-06-02);
  - Descrizione: è progettata per rilevare e normalizzare automaticamente la codifica del testo. È particolarmente utile quando si lavora con dati di origine sconosciuta o con file di testo che potrebbero avere diverse codifiche.

- **Dill**:
  - Versione: 0.3.8;
  - Documentazione: `https://dill.readthedocs.io/en/latest/` (Consultato 2024-06-02);
  - Descrizione: estende le funzionalità del modulo standard pickle, permettendo di serializzare e deserializzare una gamma più ampia di oggetti.

- **Idna**:
  - Versione: 3.7;
  - Documentazione: `https://pypi.org/project/idna/` (Consultato 2024-06-02);

  - Descrizione: implementa l'Internationalized Domain Names in Applications (IDNA) standard. Questo standard consente l'uso di caratteri Unicode nei nomi di dominio, permettendo l'inclusione di caratteri non-ASCII come quelli utilizzati in molte lingue del mondo.

- **Isolate**:
  - Versione: 0.6.1;
  - Documentazione: `https://docs.python.org/3/howto/isolating-extensions.html` (Consultato 2024-06-02);
  - Descrizione: progettata per eseguire codice Python in un ambiente isolato, separato dal contesto globale dell'applicazione.

- **Isort**:
  - Versione: 5.13.2;
  - Documentazione: `https://pycqa.github.io/isort/` (Consultato 2024-06-02);
  - Descrizione: utilizzata per ordinare automaticamente le dichiarazioni di import in un file Python in base a determinati criteri. Mantenere un ordine coerente e ben strutturato delle importazioni può rendere il codice più leggibile e facilitare la manutenzione.

- **Kafka-python**:
  - Versione: 2.2.2;
  - Documentazione: `https://kafka-python.readthedocs.io/en/master/` (Consultato 2024-06-02);
  - Descrizione: libreria Python ampiamente utilizzata per interagire con Apache Kafka, una piattaforma di streaming distribuita.

- **McCabe**:
  - Versione: 0.7.0;
  - Documentazione: `https://here-be-pythons.readthedocs.io/en/latest/python/mccabe.html` (Consultato 2024-06-02);
  - Descrizione: fornisce strumenti per misurare la complessità ciclomatica del codice sorgente Python.

- **Pip-autoremove**:
  - Versione: 0.10.0;
  - Documentazione: `https://pypi.org/project/pip-autoremove/` (Consultato 2024-06-02);
  - Descrizione: è uno strumento aggiuntivo per pip che consente di rimuovere automaticamente i pacchetti Python non utilizzati da un ambiente virtuale.
- **Platformdirs**:
  - Versione: 4.2.0;
  - Documentazione: `https://platformdirs.readthedocs.io/en/latest/` (Consultato 2024-06-02);
  - Descrizione: fornisce un'interfaccia semplice e multi-piattaforma per ottenere i percorsi di directory standard per l'archiviazione di dati specifici dell'applicazione su diverse piattaforme.
- **Python-dotenv**:
  - Versione: 1.0.1;
  - Documentazione: `https://pypi.org/project/python-dotenv/` (Consultato 2024-06-02);
  - Descrizione: carica le variabili d'ambiente da file .env nel sistema operativo. Questo è particolarmente utile durante lo sviluppo per mantenere segrete e configurabili le variabili d'ambiente utilizzate nel progetto.
- **Requests**:
  - Versione: 2.31.0;
  - Documentazione: `https://requests.readthedocs.io/en/latest/` (Consultato 2024-06-02);
  - Descrizione: semplifica notevolmente l'invio di richieste HTTP e la gestione delle risposte, fornendo un'API semplice e intuitiva per interagire con servizi web.
- **Ruff**:
  - Versione: 0.3.5;

  - Documentazione: `https://docs.astral.sh/ruff/` (Consultato 2024-06-02);
  - Descrizione: impiegato per l'analisi statica del codice sorgente.

- **Setuptools**:
  - Versione: 69.5.1;
  - Documentazione: `https://setuptools.pypa.io/en/latest/` (Consultato 2024-06-02);
  - Descrizione: fornisce strumenti per la creazione, la distribuzione e l'installazione di pacchetti Python.

- **Six**:
  - Versione: 1.16.0;
  - Documentazione: `https://six.readthedocs.io/` (Consultato 2024-06-02);
  - Descrizione: progettata per semplificare la scrittura di codice Python che deve essere compatibile con entrambe le versioni di Python 2 e 3.

- **Toml**:
  - Versione: 0.10.2;
  - Documentazione: `https://pypi.org/project/toml/` (Consultato 2024-06-02);
  - Descrizione: utilizzata per la lettura e la scrittura di file di configurazione nel formato TOML.

- **Urllib3**:
  - Versione: 2.2.1;
  - Documentazione: `https://urllib3.readthedocs.io/en/2.2.1/` (Consultato 2024-06-02);
  - Descrizione: è una libreria Python che fornisce funzionalità avanzate per effettuare richieste HTTP in modo sicuro e affidabile.

2.2.2 JSON

JSON (JavaScript Object Notation) è un formato di scambio dati leggero e facile da leggere e scrivere sia per gli esseri umani che per le macchine. È ampiamente utilizzato per

trasmettere dati tra un server e un client, in particolare nelle applicazioni web. JSON è basato su un sottoinsieme del linguaggio di programmazione JavaScript, ma è indipendente dal linguaggio, il che significa che può essere utilizzato con quasi tutti i linguaggi di programmazione. Questo linguaggio è composto da due strutture fondamentali, **oggetti** e **array**.

- **Oggetti**:
  - racchiusi tra parentesi graffe;
  - contengono coppie chiave-valore, dove le chiavi sono le stringhe e i valori possono essere di vari tipi;
  - le coppia chiave-valore sono separate dal simbolo ":" e ogni coppia è separata dal simbolo ",".
- **Array**:
  - racchiusi tra parentesi quadre;
  - contengono una lista ordinata di valori separati da virgole.

2.2.2.1 Impiego nel progetto

- Configurazione dashboard Grafana;
- struttura dei dati inviati dai simulatori dei sensori al broker Kafka.

2.2.3 YAML

YAML (YAML Ain't Markup Language) è un linguaggio di serializzazione dei dati, concepito per essere leggibile sia per gli esseri umani sia per le macchine. YAML viene spesso utilizzato per configurare file e scambiare dati tra applicazioni.

2.2.3.1 Impiego nel progetto

- Configurazione Docker Compose per l'avvio del progetto;
- configurazione provisioning Grafana.

2.2.4 SQL

SQL (Structured Query Language) è un linguaggio di programmazione specificamente progettato per la gestione e la manipolazione di dati all'interno di sistemi di gestione di database.

2.2.4.1 Impiego nel progetto

- Configurazione e gestione database ClickHouse.

2.2.5 Toml

TOML (Tom's Obvious, Minimal Language) è un linguaggio di serializzazione configurabile utilizzato principalmente per i file di configurazione.

2.2.5.1 Impiego nel progetto

- Configurazione e gestione dei sensori simulati.

2.2.6 Java

È un linguaggio di programmazione ad alto livello, orientato agli oggetti e progettato per avere il minor numero possibile di dipendenze di implementazione.

2.2.6.1 Versione

- Versione utilizzata: 21

2.2.6.2 Documentazione

`https://docs.oracle.com/en/java/` (Consultato 2024-06-02).

2.2.6.3 Impiego nel progetto

- Creazione di job per le aggregazioni dei dati di Flink.

2.3 Database e servizi

2.3.1 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. Ciascun tipo di dato viene inviato su un topic specifico, in modo da poter essere elaborato in modo indipendente.

2.3.1.1 Versione

La versione impiegata al momento dello sviluppo del progetto è la seguente: v23.3.11.

2.3.1.2 Documentazione

`https://docs.redpanda.com/current/home/` (Consultato 2024-06-02).

2.3.1.3 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;
- **costi**: rispetto ad altre tecnologie il carico di lavoro può essere ridotto fino a 5 volte in meno;
- **semplicità di configurazione**: oltre al *message broker*, contiene anche un *proxy* HTTP e uno *schema registry*;
- **BYOC (Bring Your Own Cluster)**: consente agli utenti finali di implementare una soluzione parzialmente gestita dal fornitore nella propria infrastruttura (come il proprio *data center* o il proprio *VPC cloud*);
- **compatibilità con le API di Kafka**: è compatibile con le API di Apache Kafka, consentendo di utilizzare le librerie e gli strumenti esistenti;

- **self-healing**: redistribuisce continuamente i dati e la *leadership* tra i nodi per mantenere il *cluster* in uno stato ottimale mentre evolve o quando i nodi falliscono.

2.3.1.4 Casi d'Uso

Esistono molteplici casi d'uso associati all'uso di *Redpanda*, tra cui:

- **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.

2.3.1.5 Impiego nel progetto

- **Broker**: riceve i dati prodotti dal simulatore e li rende disponibili ai consumatori.

All'interno di questo progetto i dati vengono trasmessi in formato JSON. In questo momento il consumatore è rappresentato dal database *CLickHouse*, il quale salva i dati resi disponibili da *Redpanda*.

2.3.2 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.

2.3.2.1 Versione

La versione impiegata al momento dello sviluppo del progetto è la seguente: 24.3.2.23.

2.3.2.2 Documentazione

`https://clickhouse.com/docs/en/intro` (Consultato 2024-06-02).

2.3.2.3 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 petabyte 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;
- **replica ad alta disponibilità**, supporta la replica dei dati tra diversi nodi, offrendo tolleranza ai guasti e alta disponibilità;
- **compressione efficiente**, utilizza algoritmi di compressione avanzati per ridurre lo spazio di archiviazione e migliorare l'efficienza I/O;
- **supporto SQL avanzato**, supporta un dialetto SQL ricco di funzionalità, permettendo agli utenti di eseguire query complesse e di sfruttare funzioni avanzate per l'analisi dei dati;
- **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;
- **costo efficacia**, essendo open source non ha costi di licenza, il che lo rende una soluzione economica.

2.3.2.4 Casi d'Uso

Esistono molteplici casi d'uso associati all'uso di *Redpanda*, 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.

2.3.2.5 Impiego nel progetto

- **Organizzazione efficiente dei dati**: il tipo di architettura di *ClickHouse* permette di comprimere i dati in modo più efficace e di leggere solo le colonne necessarie durante l'esecuzione delle query.
- **Integrazione con Redpanda**: può essere utilizzato in sinergia con Redpanda, una piattaforma di streaming dati compatibile con Apache Kafka. Questa integrazione permette di ingestire, elaborare e analizzare flussi di dati in tempo reale.
- **Aggregazione rapida dei dati**: è progettato per eseguire aggregazioni di dati in modo estremamente veloce. Grazie alle sue capacità di elaborazione colonnare, alle tecniche avanzate di compressione e all'uso di indici, ClickHouse può eseguire calcoli aggregati su grandi dataset in tempi molto ridotti.
- **Integrazione con Grafana**: si integra facilmente con Grafana, una delle piattaforme di visualizzazione dei dati più popolari. Questa integrazione permette di creare dashboard interattivi e personalizzati che visualizzano i dati in tempo reale provenienti da ClickHouse.

2.3.3 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.

2.3.3.1 Versione

La versione impiegata al momento dello sviluppo del progetto è la seguente: 10.3.0.

2.3.3.2 Documentazione

`https://grafana.com/docs/grafana/v10.4/` (Consultato 2024-06-02).

2.3.3.3 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, soddisfando 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.

2.3.3.4 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.

2.3.3.5 Impiego nel progetto

- **Visualizzazione dei dati**: utilizzato per creare dashboard interattive che visualizzano i dati provenienti da ClickHouse;
- **analisi dei dati**: utilizzato per analizzare i dati e identificare tendenze, pattern e anomalie;
- **monitoraggio degli allarmi**: utilizzato per monitorare e visualizzare gli allarmi generati dai simulatori e dagli altri componenti del sistema;
- **notifiche**: utilizzato per inviare notifiche in caso di superamento di soglie critiche.

3 Architettura di sistema

3.1 Modello architetturale

Il prodotto necessita di un'architettura in grado di gestire in tempo reale un vasto flusso di dati provenienti da sensori, offrendo contemporaneamente strumenti di visualizzazione efficaci per rendere comprensibili e utili queste informazioni. A tal fine è stata scelta la *k-architecture*.

3.1.1 k-architecture

Introdotta come alternativa alla *Lambda Architecture*, la *Kappa Architecture* semplifica questo modello. In questa architettura i dati vengono acquisiti, elaborati e analizzati in tempo reale senza la necessità di separare il flusso di dati in due percorsi distinti per il batch processing e il processing in tempo reale.

3.1.1.1 Vantaggi

- **Semplicità**: semplifica notevolmente il processo di sviluppo e manutenzione dei sistemi di data processing in tempo reale, in quanto elimina la necessità di gestire due percorsi separati per il batch processing e il processing in tempo reale;
- **costi ridotti**: la gestione di un'unica pipeline di dati riduce i costi di sviluppo e manutenzione;
- **reattività**: grazie alla capacità di elaborare i dati in arrivo immediatamente senza attendere il completamento di finestre di tempo predefinite, consente di ottenere risposte e feedback in tempo reale sui dati;
- **scalabilità**: i sistemi basati su quest'architettura possono essere facilmente scalati per gestire grandi volumi di dati e picchi di carico improvvisi, poiché è progettata per essere distribuita e può essere facilmente adattata alle esigenze di crescita del sistema;
- **flessibilità**: è altamente flessibile e può essere implementata utilizzando una varietà di strumenti e tecnologie.

3.1.2 Componenti di sistema

Sorgenti di dati

Costituite dai simulatori di sensori IoT distribuiti per la città.

Streaming layer

Gestisce il flusso di dati in tempo reale provenienti dai sensori. È composto da *Redpanda* e *Schema Registry*.

Processing layer

Elabora i dati in tempo reale per calcolare i KPI richiesti. È composto da *Flink*.

Storage layer

Memorizza i dati elaborati per l'analisi e la visualizzazione. È composto da *ClickHouse*.

Data visualization layer

Visualizza i dati elaborati in modo chiaro e intuitivo. È composto da *Grafana*.

3.2 Flusso di dati

3.3 Architettura dei simulatori

3.3.1 Modulo producers

3.3.2 Modulo serializers

3.3.3 Modulo simulators

- Classe astratta `Simulator`: rappresenta un simulatore generico. Contiene i seguenti metodi e attributi:
  - Metodi:
    - `start() None [public]`: avvia il simulatore;
    - `stop() None [public]`: ferma il simulatore;

    * `stream() Iterable[RawData]`: metodo astratto che verrà implementato dalle sottoclassi per generare i dati;
  - Attributi:
    * `sensor_uuid UUID [public]`: identificativo univoco del sensore;
    * `sensor_name str [public]`: nome del sensore;
    * `latitude float [public]`: latitudine del sensore;
    * `longitude float [public]`: longitudine del sensore;
    * `generation_delay timedelta [public]`: tempo effettivamente atteso tra la generazione di un dato e il successivo;
    * `points_spacing timedelta [public]`: distanza temporale dei dati generati;
    * `limit int [public]`: limite di dati da produrre;
    * `timestamp datetime [public]`: data da cui iniziare a produrre dati;
    * `running bool [private]`: flag che indica se il simulatore è in esecuzione.

- Classe concreta `AirQualitySimulator`: estende `Simulator` e rappresenta un simulatore per la generazione di dati relativi alla qualità dell'aria. Contiene i seguenti metodi e attributi:
  - Metodi:
    * `stream() Iterable[AirQualityRawData] [public]`: genera i dati relativi alla qualità dell'aria;
  - Attributi aggiuntivi rispetto a `Simulator`:
    * `o3_coefficient float [private]`: coefficiente generato con distribuzione uniforme tra -50 e 50 che viene sommato ogni volta ai valori generati per far si che varino a seconda del sensore che li ha generati;
    * `no2_coefficient float [private]`: coefficiente generato con distribuzione uniforme tra -50 e 50 che viene sommato ogni volta ai valori generati per far si che varino a seconda del sensore che li ha generati;
    * `so2_coefficient float [private]`: coefficiente generato con distribuzione uniforme tra -50 e 50 che viene sommato ogni volta ai valori generati per far si che varino a seconda del sensore che li ha generati;
    * `pm10_coefficient float [private]`: coefficiente generato con distribuzione uniforme tra -50 e 50 che viene sommato ogni volta ai valori generati per far si che varino a seconda del sensore che li ha generati;

    * `pm25_coefficient float [private]`: coefficiente generato con distribuzione uniforme tra -50 e 50 che viene sommato ogni volta ai valori generati per far si che varino a seconda del sensore che li ha generati.
  - Funzioni di supporto:
    * `daily_variation(timestamp: datetime) float [private]`: genera una variazione giornaliera in base alla data e ora passata come argomento;
    * `weekly_variation(timestamp: datetime) float [private]`: genera una variazione settimanale in base alla data e ora passata come argomento;
    * `seasonal_variation(timestamp: datetime) float [private]`: genera una variazione stagionale in base alla data e ora passata come argomento;
    * `sinusoidal_value(timestamp: datetime) float [private]`: genera un valore sinusoidale in base alla data e ora passata come argomento, richiamando le funzioni di variazione giornaliera, settimanale e stagionale e sommandole tra di loro.

- Classe concreta `RecyclingPointSimulator`: estende `Simulator` e rappresenta un simulatore per la generazione di dati relativi alle isole ecologiche. Contiene i seguenti metodi e attributi:
  - Metodi:
    * `stream() Iterable[RecyclingPointRawData] [public]`: genera i dati relativi alle isole ecologiche;
    * `calculate_fill_rate() None [private]`: calcola la velocità di riempimento dell'isola ecologica dall'ultimo dato generato;
    * `filling() float [private]`: calcola la percentuale di riempimento dell'isola ecologica;
  - Attributi aggiuntivi rispetto a `Simulator`:
    * `last_value float [private]`: percentuale di riempimento dell'isola ecologica dell'ultimo dato generato;
    * `prev_timestamp datetime [private]`: data e ora dell'ultimo dato generato;
    * `fill_rate float [private]`: velocità di riempimento dell'isola ecologica;
    * `emptying_hours List[Tuple[int, int]] [private]`: lista di giorni della settimana in cui l'isola ecologica verrà svuotata;
    * `noise_limit float [private]`: numero casuale generato con distribuzione uniforme che rappresenta il rumore;

    * `partial_emptying_chance float [private]`: probabilità che l'isola ecologica venga svuotata parzialmente;
    * `partial_emptying_max_percentage float [private]`: percentuale di rifiuti rimanente dopo uno svuotamento parziale;
  - Funzioni di supporto:
    * `generate_emptying_hours() List[Tuple[int, int]] [private]`: genera una lista di giorni della settimana in cui l'isola ecologica verrà svuotata;

- Classe concreta `TemperatureSimulator`: estende `Simulator` e rappresenta un simulatore per la generazione di dati relativi alla temperatura. Contiene i seguenti metodi e attributi:
  - Metodi:
    * `stream() Iterable[TemperatureRawData] [public]`: genera i dati relativi alla temperatura;
  - Funzioni di supporto:
    * `seasonal_coefficient(timestamp: datetime) float [private]`: genera un coefficiente stagionale in base alla data e ora passata come argomento;
    * `thermal_excursion(timestamp: datetime) float [private]`: genera un'escursione termica giornaliera in base alla data e ora passata come argomento;
    * `sinusoidal_value(timestamp: datetime) float [private]`: genera un valore sinusoidale in base alla data e ora passata come argomento, richiamando le funzioni di variazione stagionale e di escursione termica.

- Classe concreta `TrafficSimulator`: estende `Simulator` e rappresenta un simulatore per la generazione di dati relativi al traffico. Contiene i seguenti metodi e attributi:
  - Metodi:
    * `stream() Iterable[TrafficRawData] [public]`: genera i dati relativi al traffico;
  - Costanti:
    * `VEHICLES_MULTIPLICATIVE_FACTOR int [private]`: costante moltiplicativa per il numero di veicoli;

    * `SPEED_MULTIPLICATIVE_FACTOR int [private]`: costante moltiplicativa per la velocità media.
  - Funzioni di supporto:
    * `multimodal_gauss_value(x: float, modes: list[tuple[float, float]]) float [private]`: genera un valore gaussiano multimodale in base ai secondi passati dall'inizio della giornata e ai modi passati come argomento;

3.3.4 Modulo models

3.3.4.1 config

Contiene le classi e le funzioni utilizzate per leggere ed effettuare il parsing del file di configurazione *sensor.toml*, utilizzato per costruire i vari simulatori.

3.3.4.1.1 Classi e metodi

- Classe concreta `EnvConfig`: legge le variabili d'ambiente e nel caso siano assenti ma ammettano valore di default lo assegna.
  - Metodi
    * `get_or_throw(key: str) str [private]`: ritorna il valore della variabile d'ambiente con quella chiave o se non esiste lancia un'eccezione;
    * `get_or_none(key: str) str | None [private]`: ritorna il valore della variabile d'ambiente con quella chiave o se non esiste ritorna `None`;
    * `bootstrap_server() str [public]`: ritorna concatenati i valori delle variabili d'ambiente `KAFKA_HOST` e `KAFKA_PORT` separati da il simbolo ":";
  - Attributi
    * `kafka_host str [public]`: host del broker Kafka;
    * `kafka_port str [public]`: porta del broker Kafka;
    * `log_level str [public]`: livello di log;
    * `max_block_ms str [public]`: massimo numero di millisecondi che il producer kafka può rimanere bloccato durante `send()`.
- Classe concreta `SensorConfig`: legge il file di configurazione *sensor.toml* e ne effettua il parsing.
  - Attributi

    * `uuid UUID [public]`: identificativo univoco del sensore;
    * `limit int [public]`: limite di dati da produrre;
    * `begin_date datetime [public]`: data da cui iniziare a produrre dati. Opzionale e nel caso non sia presente viene presa la data attuale;
    * `latitude float [public]`: latitudine del sensore;
    * `longitude float [public]`: longitudine del sensore;
    * `generation_delay timedelta [public]`: tempo effettivamente atteso tra la generazione di un dato e il successivo. Espresso in formato ISO 8601 e nel caso non sia conforme viene sollevata un'eccezione;
    * `points_spacing timedelta [public]`: distanza temporale dei dati generati. Espresso in formato ISO 8601 e nel caso non sia conforme viene sollevata un'eccezione.

3.3.4.2 raw_data

- Classe astratta `RawData`: rappresenta un dato grezzo generato da un sensore. Contiene i seguenti metodi e attributi:
  - Metodi
    * `accept(visitor: SerializerVisitor) Dict [str,any] [public]`: metodo astratto che verrà utilizzato dalle sottoclassi per poter implementare il pattern Visitor;
    * `topic() str [public]`: ritorna il nome del topic Kafka a cui il dato deve essere inviato;
    * `subject() str [public]`: ritorna un identificativo associato allo schema del raw data. Nello specifico ritorna il nome del topic seguito da "-value";
  - Attributi
    * `sensor_uuid UUID [public]`: identificativo univoco del sensore;
    * `sensor_name str [public]`: nome del sensore;
    * `latitude float [public]`: latitudine del sensore;
    * `longitude float [public]`: longitudine del sensore;
    * `timestamp datetime [public]`: data e ora della misurazione;
- Classe concreta `AirQualityRawData`: estende `RawData` e rappresenta un dato grezzo generato da un sensore di qualità dell'aria. Contiene i seguenti metodi e attributi:

  - Metodi
    * `accept(visitor: SerializerVisitor) Dict [str,any] [public]`: implementa il metodo della classe padre, invocando il metodo per la serializzazione della classe AirQualityRawData;
    * `topic() str [public]`: ritorna "air_quality".
  - Attributi aggiuntivi rispetto a `RawData`:
    * `pm25 float [public]`: quantità di particolato atmosferico con un diametro aerodinamico inferiore o uguale a 2.5 micrometri e misurato in $\mu g/m^3$;
    * `pm10 float [public]`: quantità di particolato atmosferico con un diametro aerodinamico inferiore o uguale a 10 micrometri e misurato in $\mu g/m^3$;
    * `no2 float [public]`: concentrazione di biossido di azoto misurata in $\mu g/m^3$;
    * `o3 float [public]`: concentrazione di ozono misurata in $\mu g/m^3$;
    * `so2 float [public]`: concentrazione di biossido di zolfo misurata in $\mu g/m^3$.

- Classe concreta `RecyclingPointRawData`: estende `RawData` e rappresenta un dato grezzo generato da un sensore di isola ecologica. Contiene i seguenti metodi e attributi:
  - Metodi:
    * `accept(visitor: SerializerVisitor) Dict [str,any] [public]`: implementa il metodo della classe padre, invocando il metodo per la serializzazione della classe RecyclingPointRawData;
    * `topic() str [public]`: ritorna "recycling_point".
  - Attributi aggiuntivi rispetto a `RawData`:
    * `filling float [public]`: percentuale di riempimento dell'isola ecologica.

- Classe concreta `TemperatureRawData`: estende `RawData` e rappresenta un dato grezzo generato da un sensore di temperatura. Contiene i seguenti metodi e attributi:
  - Metodi:
    * `accept(visitor: SerializerVisitor) Dict [str,any] [public]`: implementa il metodo della classe padre, invocando il metodo per la serializzazione della classe TemperatureRawData;
    * `topic() str [public]`: ritorna "temperature".
  - Attributi aggiuntivi rispetto a `RawData`:

    * `value float [public]`: valore della temperatura rilevata espressa in °C.

- Classe concreta `TrafficRawData`: estende `RawData` e rappresenta un dato grezzo generato da un sensore di traffico. Contiene i seguenti metodi e attributi:
  - Metodi:
    * `accept(visitor: SerializerVisitor) Dict [str,any] [public]`: implementa il metodo della classe padre, invocando il metodo per la serializzazione della classe TrafficRawData;
    * `topic() str [public]`: ritorna "traffic".
  - Attributi aggiuntivi rispetto a `RawData`:
    * `vehicles int [public]`: numero di veicoli rilevati;
    * `avg_speed float [public]`: velocità media dei veicoli rilevati espressa in km/h.

3.3.5 Progettazione - Panoramica UML

3.4 Redpanda

3.4.1 Kafka topic

I Kafka topic sono categorie o canali di messaggi all'interno di *Redpanda*. Un topic in Kafka è come una cassetta postale virtuale o una categoria di messaggi in cui i dati vengono pubblicati dai produttori e letti dai consumatori.

3.4.2 Formato messaggi

7Last ha scelto di adottare lo standard Avro per i messaggi scambiati tra i produttori e i consumatori. Avro è un sistema di serializzazione dati che offre un modo efficiente per rappresentare dati complessi in un formato binario, rendendoli adatti per il trasporto su rete o per la persistenza su disco. Uno dei principali vantaggi di Avro è la possibilità di definire uno schema per i dati, che viene incluso nei dati stessi. Ecco una panoramica del formato dei messaggi Avro:

- **schema**: definito in formato JSON e descrive la struttura dei dati. Include informazioni come il tipo di ogni campo e la loro posizione all'interno della struttura dei dati;

- **serializzazione binaria**: Avro serializza i dati in un formato binario compatto, che rende efficiente il trasporto e la memorizzazione dei dati. Utilizza un'organizzazione binaria che incorpora lo schema dei dati insieme ai dati stessi. Questo significa che non c'è bisogno di includere esplicitamente il tipo di dato per ogni campo, poiché lo schema fornisce questa informazione;
- **compatibilità**: è progettato per supportare l'evoluzione dei dati nel tempo. Puoi aggiungere nuovi campi, rimuovere campi esistenti o modificare il tipo di dati di un campo mantenendo la compatibilità con le versioni precedenti degli schemi;
- **condivisione dello schema**: supporta la condivisione dello schema tramite un registro dello schema (Schema Registry). Questo consente di registrare gli schemi Avro utilizzati nel sistema in un registro centralizzato, in modo che i produttori e i consumatori possano recuperare gli schemi necessari quando ne hanno bisogno.

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 & 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. In questa sezione, esamineremo le principali componenti di Flink e il loro ruolo nella creazione di un'infrastruttura robusta per l'analisi dei dati.

- **JobManager**: è il componente centrale di Flink responsabile della pianificazione e del coordinamento dei job di elaborazione dei dati. Gestisce il flusso di lavoro complessivo, assegnando i task ai TaskManager per l'esecuzione;

- **TaskManager**: è responsabile dell'esecuzione effettiva delle operazioni di elaborazione dei dati, eseguendo i task assegnati loro dal JobManager e gestendo il caricamento, l'elaborazione e la distribuzione dei dati all'interno del cluster;
- **Processing layer**: è responsabile dell'esecuzione delle operazioni di elaborazione dei dati all'interno del cluster distribuito. Questa layer sfrutta le risorse di calcolo e memorizzazione disponibili nei nodi del cluster per eseguire le operazioni di trasformazione, aggregazione, filtraggio e altro ancora sui dati in ingresso. Utilizzando un modello di esecuzione distribuita, la Processing Layer di Flink è in grado di scalare orizzontalmente per gestire grandi volumi di dati e carichi di lavoro ad alta intensità computazionale.

3.5.3 Processing layer data-flow

1. **Acquisizione dei dati**;
2. **partizione e distribuzione**: i dati vengono divisi in parti più piccole e distribuiti tra i nodi del cluster per massimizzare l'utilizzo delle risorse;
3. **pianificazione dei task**: il JobManager assegna compiti di elaborazione ai TaskManager basandosi sullo stato del cluster e sull'ottimizzazione delle prestazioni;
4. **esecuzione dei task**: i TaskManager eseguono i compiti assegnati in parallelo, elaborando i dati in base alla logica definita nell'applicazione Flink;
5. **scambio e movimento dei dati**: i dati possono essere scambiati e spostati tra i nodi del cluster per supportare operazioni complesse come il join o l'aggregazione;
6. **persistenza e output**: una volta elaborati, i risultati vengono eventualmente salvati o inviati ad altre destinazioni per l'analisi o l'utilizzo successivo.

3.5.4 Modello per il calcolo dei KPI

3.6 Database ClickHouse

Come detto in precedenza, il database adottato è *ClickHouse*. Per ogni sensore è stata creata una tabella *MergeTree*, che permette di memorizzare i dati in modo efficiente e di eseguire query complesse in modo veloce.

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` (Consultato il 2024-06-05)

Utilizzi

- **Aggregazioni pre-calcoltate**: le materialized view possono essere utilizzate per memorizzare i risultati di aggregazioni complesse, come somme, medie, conteggi, ecc., in modo che non debbano essere calcolati ogni volta che viene eseguita una query;
- **rapporti pre-calcolati**: possono essere utilizzate per memorizzare i risultati di query complesse o di rapporti, in modo che i risultati siano immediatamente disponibili senza dover eseguire la query ogni volta;
- **join ottimizzati**: possono essere utilizzate per memorizzare i risultati di join complessi tra più tabelle, in modo che i risultati siano immediatamente disponibili senza dover eseguire il join ogni volta;
- **filtraggio e selezione efficiente**: possono essere utilizzate per filtrare e selezionare dati in base a criteri specifici, migliorando le prestazioni delle query che richiedono l'accesso solo a una parte dei dati.

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`
(Consultato il 2024-06-05)

Utilizzi

- **Analisi dei dati storici**: i dati storici vengono memorizzati in tabelle MergeTree per consentire l'analisi e l'elaborazione dei dati storici;
- **applicazioni di business intelligence**: i dati vengono memorizzati in tabelle MergeTree per consentire l'analisi e la generazione di report per le applicazioni di business intelligence;
- **log e monitoraggio**: i dati di log e di monitoraggio vengono memorizzati in tabelle MergeTree per consentire l'analisi e il monitoraggio delle attività di sistema.

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/`

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/` (Consultato il 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 (es. 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/` (Consultato il 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.

7Last ha deciso di rendere risponibile il server *Discord* configurato a questo scopo e raggiungibile a questo link:

`https://discord.com/channels/1214553333113556992/1241974479345942568`

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` (Consultato il 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 *IoT* 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.

Codice	Importanza	Stato	Descrizione
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.
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 contente 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: *PM10*, *PM2.5*, *NO2*, *CO*, *O3*, *SO2* 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 1: Requisiti funzionali

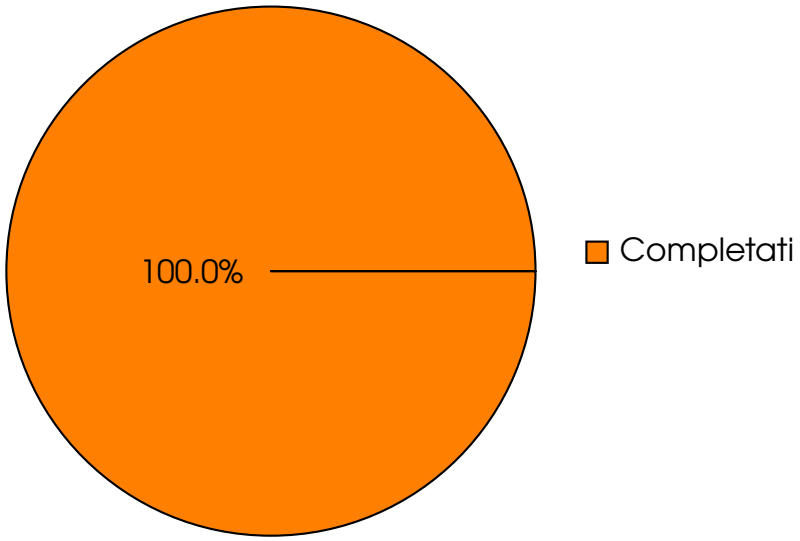

Figura 1: Percentuale di soddisfacimento dei requisiti funzionali

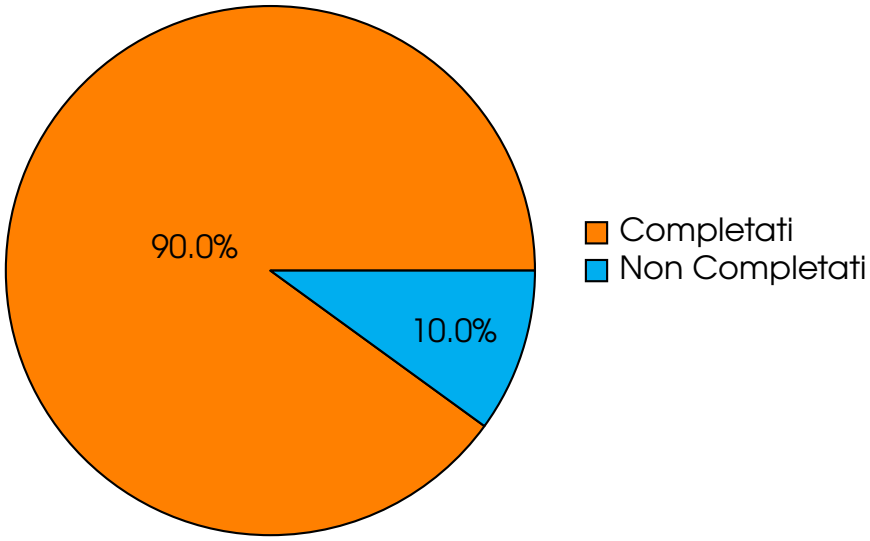

Figura 2: Percentuale di soddisfacimento dei requisiti totale