Web of Thing e Data Storage per applicazioni di Monitoraggio Strutturale

Laurea Informatica Magistrale
Curriculum C: Sistemi e Reti

Relatore: **Marco Di Felice**
Co-Relatore: **Lorenzo Gigli**

Presentata da: **Matteo Sanfelici**

Stato dell'Arte

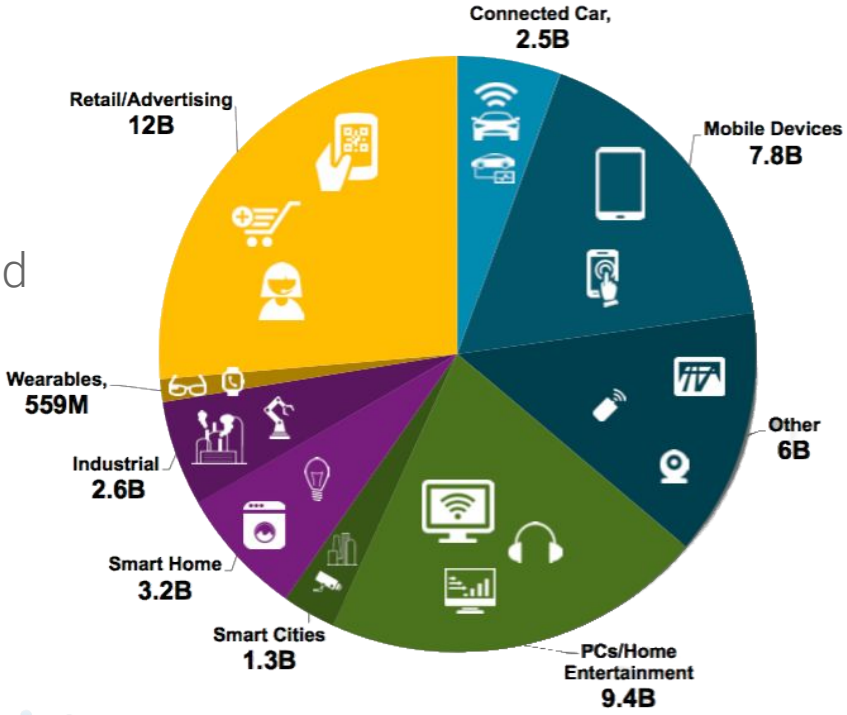
Matteo Sanfelici - Matricola 856403

Internet of Things

Connessione a Internet di un mondo fatto di oggetti concreti e luoghi reali

Partito da oggetti con **tag informativi** (RFID, QR, NFC) fino ad **ambienti più complessi** (Connected Car, Smart Home, Smart Cities)

Problematica principale: **Frammentazione** e difficoltà nell'inter-operabilità tra ecosistemi



Monitoraggio Strutturale (SHM)

Finalità

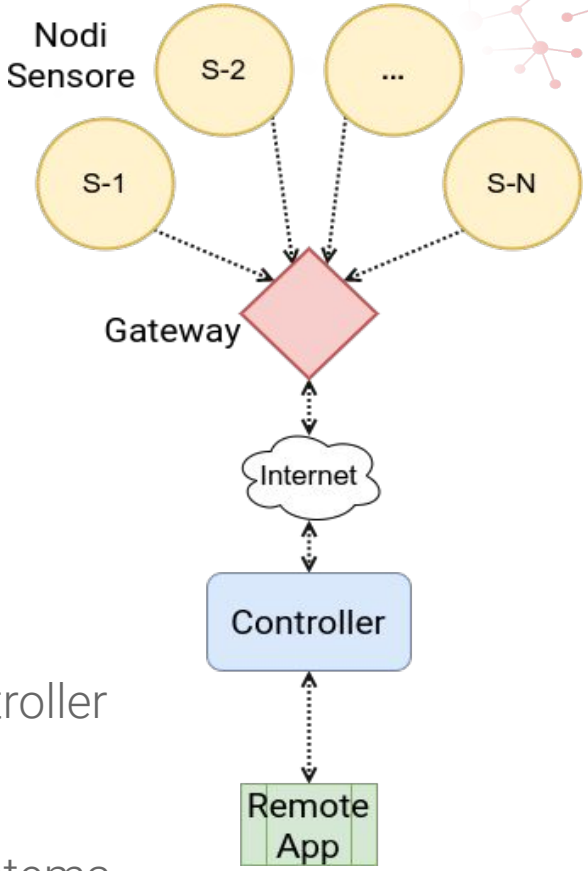
Usare tecniche di prognostica e simulazione per migliorare la sicurezza e ridurre i costi di manutenzione di opere civili ed industriali

Come

Con strutture monitorate da reti di sensori collegati ad un'infrastruttura capace di gestire alti throughput di dati

Architettura Generale

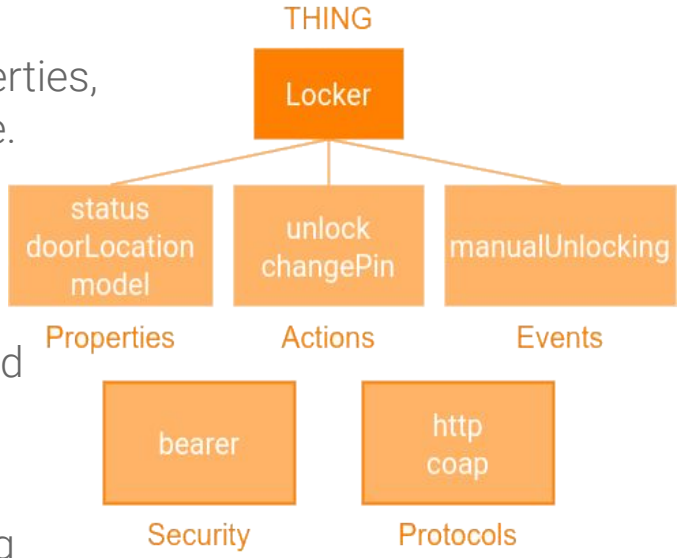
- **Nodi Sensore:** Raccolta dati costante, invio wireless
- **Gateway:** Aggregazione e campionatura, invio al controller
- **Controller:** Persistenza e gestione accesso ai dati, stato dei sensori
- **Remote App:** Interfaccia di controllo ed analisi del sistema



W3C WoT - Web of Things

Per contrastare la frammentazione nell'IoT un W3C Working Group ha proposto lo standard W3C WoT diviso in 3 blocchi:

- **WoT TD Thing Description**
  Descrittore di una Thing serializzata attraverso JSON-LD.
  Descrive metadati relativi a: **modelli di interazione** (properties, action ed event), **sicurezza** e **protocolli** di comunicazione.
  Ogni metadato è *descritto semanticamente* attraverso vocabolari ed ontologie
- **WoT Binding Templates**
  Collezione di Metadati relativi a protocolli implementati ed utilizzabili per la comunicazione
- **WoT Scripting API**
  API per implementare, gestire ed interagire con una Thing



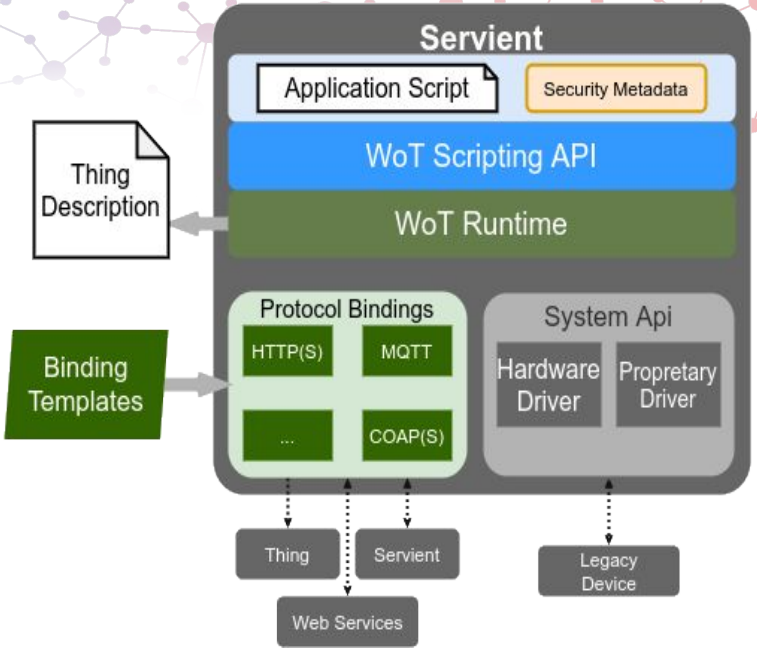
W3C WoT - Servient

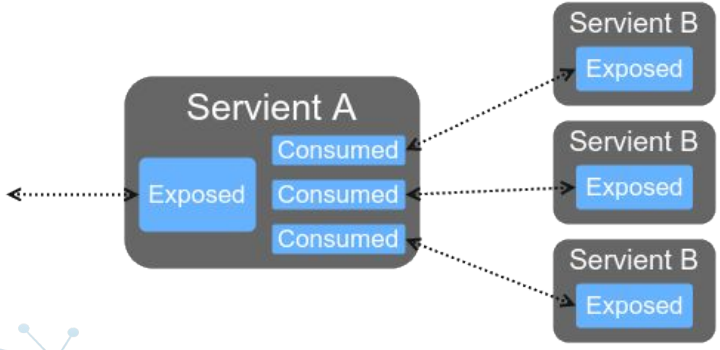
Una servient è lo stack software che **implementa** tutti i blocchi del **W3C WoT**.

Una servient può **esporre** e/o **consumare** una Thing (TD), fungendo sia da client che da server.

È possibile **astrarre funzionalità eterogenee** da una Thing ed **esporle in modo standardizzato** con il W3C WoT.

L'architettura del W3C WoT ha come obiettivo **aumentare l'interoperabilità nell'IoT** astraendo funzionalità attraverso una o più WoT Thing





Mac4Pro - Persistence

Matteo Sanfelici - Matricola 856403

Obiettivi

Obiettivi

Lo scopo del Progetto Mac4Pro è creare un'**infrastruttura W3C WoT** per raccogliere, salvare e consultare dati IoT su cui applicare **tecniche innovative** di analisi per il **Monitoraggio Strutturale**.

All'interno di questo progetto, l'obiettivo del mio elaborato è **progettare** accuratamente ed **implementare** la parte di **raccolta dati**, **storage distribuito** ottimale e **gestione dell'accesso ai dati**

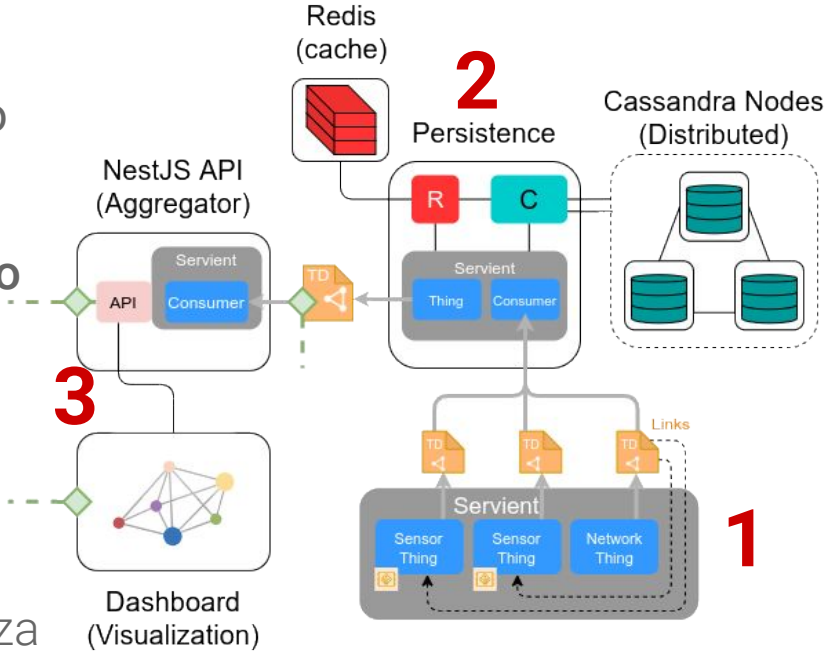
Progettazione

Matteo Sanfelici - Matricola 856403

Infrastruttura progetto Mac4Pro

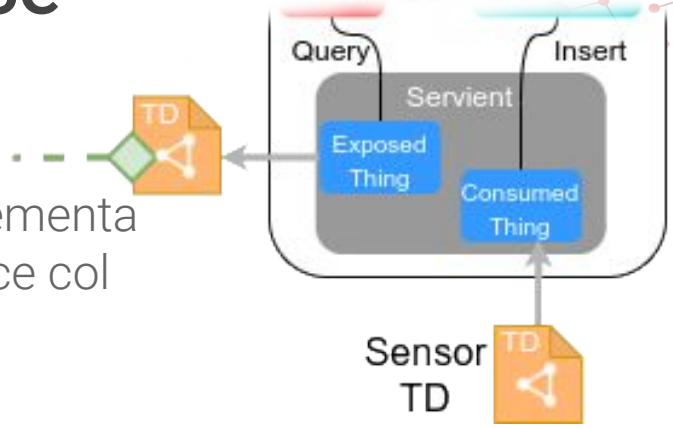
L'architettura è divisa in 3 parti:

1. **Cluster di SensorThing**
   Thing che si **interfacciano** e **rappresentano** sensori fisici collegati ad una struttura
2. **ThingPersistence con Database Distribuito**
   Consuma tutte le SensorThing, **salva** su database distribuito ed **espone** una Thing per interagire col DB
3. **API Aggregator e Visualizer**
   Aggrega i dati IoT dal database e li visualizza in modo smart



Progettazione: Thing Persistence

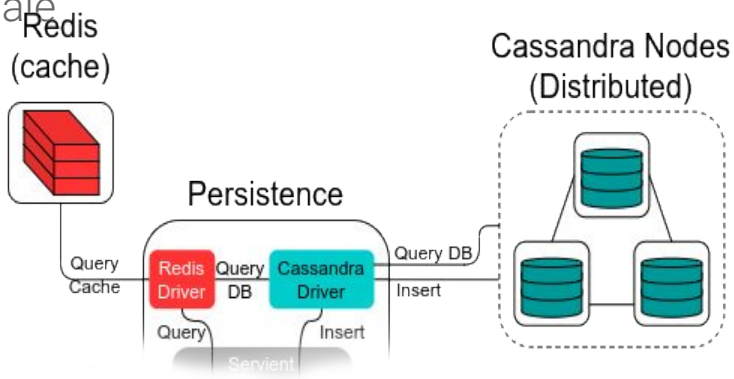
È divisa in tre componenti:



- **Consumer**
  Avvia e stoppa la **routine di consumazione**, implementa il **processing** dei TD delle **SensorThing**. Interagisce col Driver DB per **salvare i dati**
- **Thing Persistence**
  Produce ed espone un **Thing Description** con cui astrae modelli di interazione col nodo Persistence.
  Espone **handler** per **logiche interne** o per **interagire** con Driver **DB** e **Consumer**
- **Driver Database**
  Genera e configura dinamicamente gli schemi sul DB e **inserisce i dati**.
  Ottimizza l'accesso ai dati interrogando prima una Cache Redis (poi il DB se necessario)

Progettazione: Database Distribuito e Cache

Come DB Distribuito viene scelto **Cassandra** perchè è ottimale per **scrittura intensiva** di dati e **scalabilità a runtime**. Inoltre garantisce l'assenza di **single point failure**.



Viene inserita una **cache su Redis** (in-memory DB) **velocizzare l'accesso ai dati** ed alleggerire Cassandra impegnato in una **scrittura intensiva costante.**

Ogni nodo del DB segue una **configurazione specifica**:

- **Gossip**: algoritmo di *`GossipingPropertyFileSnitch`*, per reti semplici proprietarie
- **Partitioner**: il partizionatore usato è *`Murmur3Partitioner`*, per una distribuzione uniforme dei dati sui nodi
- **Replication Factor**: che è il numero di repliche di un record nel cluster, *`settato a 3`*
- **Replica Placement strategy**: *`SimpleStrategy`*, cioè come scegliere i nodi replica

Implementazione

Matteo Sanfelici - Matricola 856403

Implementazione: Database Distribuito

```
cluster_name: 'Mac4Pro_Cluster'
num_tokens: 256
seed_provider:
  - class_name: org.apache.cassandra.locator.SimpleSeedProvider
    parameters:
      - seeds: cas1-mac4pro

listen_address: localhost
rpc_address: 0.0.0.0

endpoint_snitch: GossipingPropertyFileSnitch

commit_failure_policy: stop

disk_failure_policy: stop

commitlog_sync:
  - periodic: 10000

partitioner: org.apache.cassandra.dht.Murmur3Partitioner
```

L'implementazione consiste nella configurazione dei nodi all'interno del Cluster.

Notiamo alcuni parametri citati nella progettazione ed alcuni più implementativi.

Seed provider: come identificare e connettersi al seed del sistema (cas1-mac4pro).

commit e disk failure policy: politica da adottare in casi di failure durante il commit o del disco.

commitlog sync: come sincronizzare il commit_log

Implementazione: Thing Persistence

Thing Esposta

Definisce ed espone il Thing Description ed implementa:

- **Proprietà** per parametrizzare la Thing e le logiche interne
  - **tagTest**: tag relativo ad un test fisico fatto per un dato sensore
  - **updateFrequency**: frequenza di raccolta dati
  - **clusterAddressList**: lista di clusterHead da consumare
- **Action**
  - setTagTest e setUpdateFrequency
  - addClusterHead e removeClusterHead
  - **startConsumingRoutine** e **stopConsumingRoutine**: implementano la logica di avvio/stop della routine di consumazione e salvataggio dei dati
  - **executeQuery**: esegue una query sul DB e restituisce un risultato organizzato in JSON

Implementazione: Thing Persistence

Consumer

Consuma tutte le Thing disponibili secondo parametri impostati

Processa i dati raccolti per inserirli nel DB (codice a lato).

1. Collezione TD
2. Creazione Dinamica Keyspace
3. Parse properties e creazione dinamica Table
4. Creazione ResultObject ed inserimento

```
let json = await fetch(sensorAddress).then(res => res.json());
let thing = await WoT.consume(json);

//Creating new keyspace for a cluster if not exists
let cluster = thing.getThingDescription()["@clusterName"];
let thingName = thing.getThingDescription().title;
await this.cassandraUtils.createKeyspace(cluster);

//Get property samples name
let propertySamplesNames = await this.getSamplesNames(thing);
for (const property of propertySamplesNames) {
  //Creating new table for a measurement if not exists
  await this.getColumnName(thing, property).then(async (columnName :string[] ) => {
    await this.cassandraUtils.createTable(cluster, property, columnName)
  });

  await thing.readProperty(property).then((resultList) => {
    resultList.forEach(async (resultStructure) => {
      await this.getResultObject(thingName, resultStructure).then(async (resultObject :Result ) => {
        await this.cassandraUtils.insertValue(cluster, property, resultObject);
      });
    })
  });
}
```

Implementazione: Driver DB e Deployment

Driver DB

Implementa le funzioni necessarie per interagire col DB:

- createKeyspace
- createTable
- insertValue

} Cassandra

- executeQuery → Redis + Cassandra

Configura dinamicamente per ogni keyspace Replication Strategy (SimpleStrategy) e Replication Factor (3)

Docker

Deployment completamente automatizzato e configurabile con **Docker-Compose**.

Configurazione
Tramite variabile d'ambiente nel file di config

```
node-persistence-thing:
  build:
    context: ../
    dockerfile: docker/persistence.dockerfile
  ports:
    - "8000:8000"
  environment:
    - CLUSTER_HEADS = 137.204.83.10:8080, ...
    - CASSANDRA_ENDPOINTS = cas1-mac4pro, ...
    - UPDATE_FREQUECNY = 10000
```

Implementazione
Variabili d'ambiente usate internamente attraverso `process.env.<var>`

Validazione

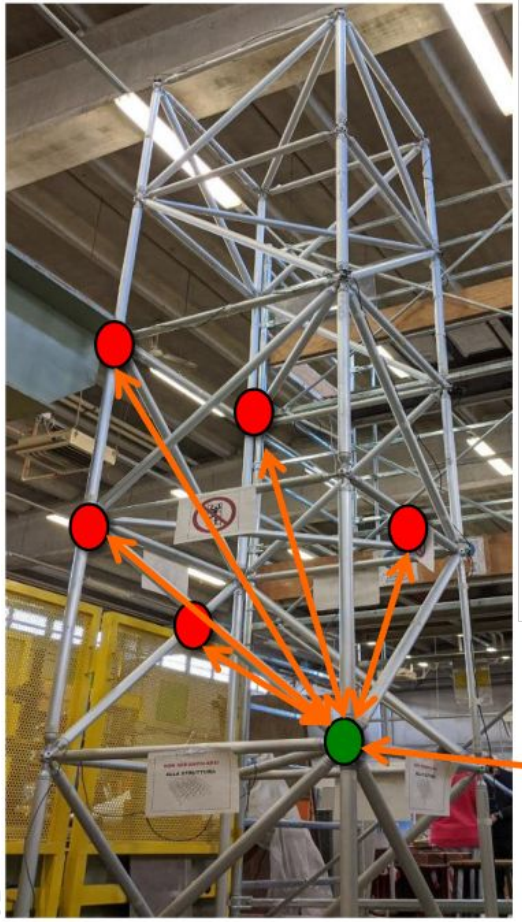
Matteo Sanfelici - Matricola 856403

Validazione: Scenario

Si considera un'infrastruttura metallica posta presso il laboratorio di **Ingegneria Strutturale e Geotecnica** di UNIBO.

Sono presenti strutture differenti con montati sensori collegati al **cluster head** esposto in rete.

Il nodo di **persistenza** si connette coi cluster head e quello di **visualizzazione** col nodo di persistenza



SENSORI

CLUSTER HEAD

Validazione: Risultati

Sono stati eseguiti dei test per confrontare la **reattività** del nodo di persistenza **con e senza Redis**.

Durante i test il nodo è **impegnato** costantemente nella **scrittura di dati** dai sensori.

L'utilizzo della cache Redis ha **riscontri positivi** per la reattività di accesso ai dati

Con la cache, **Cassandra viene stressato meno** e si occupa solo della scrittura

Conclusioni

Conclusioni

Ampio **studio preliminare** di tutte le complesse tecnologie e standard in gioco(IoT, WoT, DB Distribuiti, Deployment Automatizzato)

Progetto svolto in concerto con altri **membri del team** al lavoro su Mac4Pro.

Obiettivo Iniziale Raggiunto

Nodo di persistenza **altamente generalizzato**, **facilmente configurabile**, **scalabile** e in grado di gestire un **alto throughput di dati** da salvare.

Sviluppi Futuri

Implementare attraverso una Virtual Thing conforme con il W3C Wot tutta la **configurazione del sistema**, la possibilità di decidere **come e cosa scalare** o se farlo in modo automatico e la **gestione del deployment.**

Grazie per l’attenzione

Matteo Sanfelici - Matricola 856403