**ALMA MATER STUDIORUM UNIVERSITÁ DI BOLOGNA**

**SCUOLA DI INGEGNERIA E ARCHITETTURA**

DIPARTIMENTO INFORMATICA - SCIENZA E INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

**TESI DI LAUREA**

in

*Mobile Systems M*

# Monitoraggio dei processi produttivi nell'Industra 4.0: un approccio open source

Candidato:

**Gianluca Rosi**

Relatore:

Chiar. mo Prof. Ing. **Paolo Bellavista**

Correlatore:

Dott. Ing. **Luca Foschini**

Anno Accademico 2019-2020

Sessione II

*“Data isn’t information, any more than fifty tons of cement is a skyscraper”*

Clifford Stoll

# Indice

# Abstract

In questo studio si è posto sotto osservazione il comportamento di un prodotto software open source ancora in fase di sviluppo: posizionato a metà tra OT e IT in ambito Indutrial IoT, si prefigge di uniformare le comunicazioni tra le due parti. Si è quindi aggiunta una funzionalità di inoltro, programmabile dinamicamente attraverso una interfaccia utente, che ne agevoli l'utilizzo da parte degli operatori. La conseguente verifica delle prestazioni ha permesso di stilare una prima visione sull'applicabilità sia in termini di vantaggi nell'occupazione di risorse e di conseguenza economici, sia in termini di semplicità di implementazione.

Ci si è focalizzati infatti nell'osservare la quantità di dati che si riescono ad aggregare e inoltrare in proporzione al relativo impegno computazionale, rispetto a varie configurazioni, rispettive a diverse tipologie di implementazione (che siano su dispositivi con capacità limitate, non dedicati e su un'architettura distribuita). Una visione finale d'insieme restituisce un prodotto sì completo ma che lascia diversi punti di approfondimento aperti a nuove personalizzazioni e incrementi in efficienza.

# Introduzione

In un mondo in cui le comunicazioni avvengono più tra dispositivi tecnologici e non sono più caratteristica peculiare degli esseri viventi, solo un radicale cambio architetturale permetterà un utilizzo condiviso dei vettori in maniera sostenibile. Nell'industria, che sia agricola, manifatturiera o dei servizi, è fondamentale un controllo della produzione e dei prodotti stessi generando statistiche e previsioni per ottimizzare scelte produttive e di salvaguardia della componente umana.

Ciò implica uno scambio di dati importante dalle sorgenti verso centri di analisi che li trasformano in informazioni su cui poter basare decisioni automatizzate, quindi attuate dai macchinari stessi, scelte orientate al rendere più salutare il posto di lavoro degli operatori oppure per guidare investimenti in ambito economico.

Un utilizzo non improvvisato delle reti unito ad una redistribuzione delle capacità di calcolo sono i concetti che hanno reso possibile il realizzarsi della rivoluzione industriale in atto: ogni azione viene intrapresa non più solo dallo sfruttamento di una forma di energia, ma a seguito del risultato di un'elaborazione. La generazione di queste risposte alle esigenze produttive richiede l'invio di una ingente quantità di dati che deve essere gestita oculatamente per riservare risorse in caso di necessità, come nel caso di guasti, in cui è necessaria una supervisione concentrata sul componente problematico lasciando lo stretto necessario agli altri.

Problematiche che vengono affrontate tramite personalizzazioni non

sempre uniformi nel mondo dell'acquisizione dati e che richiedono uno sforzo aggiuntivo per l'interpretazione, per questo si adottano sia soluzioni architetturali che nei protocolli condivisi, lasciando così agli utenti la possibilità di investire internamente tempo e denaro.

Il concetto di Industria 4.0, nato nel 2011 alla Fiera di Hannover da Henning Kagermann, Wolf-Dieter Lukas e Wolfgang Wahlster, definisce come l'adozione di più tecnologie interconnesse su più livelli, possono essere aggregate formando una rete di conoscenza multidisciplinare su un sistema che collega OT (Operational Technology) e IT (Information Technology). Questo punto d'incontro è nell'edge computing dove software appositamente creati come quello preso in studio nella presente discussione, EdgeX Foundry, permette di dirigere in maniera *smart* i flussi delle conversazioni, ne uniforma i protocolli, generando un'interfaccia comune che si interpone tra gli attori.

Tutto ciò è stato confermato nelle verifiche di applicazione in diverse configurazioni, nonostante la scarsa maturità del progetto, che prevede ancora una roadmap incentrata sulla stabilità e ottimizzazione dell'architettura interna, lasciando allo sviluppatore l'implementazione di nuove funzionalità secondarie, obiettivo della discussione in oggetto.

Nel **primo capitolo** saranno contestualizzati e poi introdotti i problemi di *offloading* e *zoom-in* che si è cercato di affrontare.

Il **secondo capitolo** descriverà il framework EdgeX Framework, principale strumento sul quale si è sviluppata la soluzione proposta.

L'esplorazione della logica della soluzione sarà descritta nel **terzo capitolo**, evidenziandone le componenti ed il suo comportamento.

Nel **quarto capitolo** si entra nel dettaglio dell'implementazione del sistema sviluppato e se ne valutano le prestazioni.

Infine, al **quinto capitolo** si interpretano i risultati ottenuti con uno sguardo all'applicabilità ed a futuri sviluppi.

Capitolo 1

# Monitoraggio e management dati

## 1.1 Scenario

L'acquisizione affidabile delle informazioni è un passaggio cruciale per la validità e la sostenibilità delle azioni intraprese assieme alla formulazione di nuove. Fino all'introduzione dell'elettronica, i dati, benché utili al progresso e l'efficientamento dei macchinari, assieme alla verifica della rendita della produzione e della logistica, hanno giocato un ruolo marginale perché, per forza di cose, registrati a mano e gestiti in totale manualità dall'uomo.

Manipolare le fonti energetiche è sempre stato il concetto alla base delle rivoluzioni industriali, applicando ricerca e ingegno ha portato l'acqua a diventare vapore e l'elettricità, informazione. La forza motrice della terza rivoluzione industriale nel 1970 è stata infatti l'introduzione dei controllori logici, dei sensori e degli attuatori elettronici, rendendo possibile la verifica istantanea nella catena di produzione, permettendo così di intraprendere decisioni automatizzate, sollevando gli operatori da compiti manuali ripetitivi.

Con l'aumentare della capacità di calcolo, è stato possibile recuperare una quantità e varietà di dati come mai prima: dalle letture logiche ai numeri a virgola mobile, fino agli stream multimediali. Acquisizioni dati effettuate da componenti e protocolli speciali con conseguente frammenta-

zione e difficoltà di analisi degli stessi. Questo ha spinto verso una riforma delle infrastrutture di comunicazione sia nel ramo delle telecomunicazioni con l'introduzione del 5G, che in quello industriale con l'Industria 4.0: qualsiasi meccanismo sarà sempre più attivato dalle informazioni estrapolate ed interpretate in automatico dagli innumerevoli dispositivi, invece che manualmente all'uomo.

I dati presi in considerazione in questo approfondimento descrivono gli aspetti di un sistema in un particolare momento: sono infatti chiamati *metriche*. La raccolta di queste avviene ad intervalli regolari, identificando univocamente ogni lettura per poterne estrarre informazioni e trend nel tempo; ciò permette di capire lo stato della propria infrastruttura, stimarne affidabilità e salubrità.

Da qui la definizione di *sistema ciberfisico* (CPS): un approccio interdisciplinare che va dalla cibernetica alla meccatronica, dedicato al controllo di meccanismi che si interfacciano tra di loro per permettere agli algoritmi che li governano informazioni per agire. Questo ambito è una specializzazione del più alto livello ricoperto nell'IoT (Internet of Things).

### 1.1.1 Industrial IoT

Abbreviato con IIoT è base della rivoluzione industriale in atto: attraverso l'interconnessione di tutto l'apparato sensoristico, si cerca di acquisire quanti più dati utili ad un maggior controllo automatizzato di ogni fase produttiva, sia dei macchinari che degli operatori e nello stilare statistiche. Risultando così anello di connessione con la IT (Information Technology) e la Operational Technology, di cui fa parte in quanto è hardware e software che collabora sia nel rilevare sia nell'attuare cambiamenti operativi.

Tutela della salute e prevenzione di blocchi dell'azienda spronano ogni giorno lo sviluppo di algoritmi dedicati a prevedere sempre più precisamente la gestione del personale e dei macchinari. Il riconoscimento visivo dei movimenti degli operatori permette di stimare condizioni di rischio e lavori usuranti, cercando così di migliorare l'ergonomia e sicurezza del posto di lavoro.

L'avanguardia della ricerca prospetta come le soluzioni che prevedono un addestramento del personale non avverrà più con lo studio sui manuali, bensì attraverso la realtà aumentata: in questo caso dispositivi *wearable* come occhiali a realtà aumentata, forniranno feedback sui movimenti per correggerli [1]. Ove ciò non sia attuabile, le informazioni raccolte possono essere utilizzate per implementare *robot collaborativi*, che rispetto a quelli tradizionali sono adatti a lavorare con l'operatore umano aiutandolo nelle operazioni più intense e meno ergonomiche lasciandogli solo quelle a vero valore aggiunto.

L'automazione nella gestione degli eventi critici ha sfruttato la predominanza dell'interazione M2M (Machine to Machine), diminuendo i tempi di risposta nelle contromisure e avvertimenti scaturiti dagli algoritmi di previsione. Si è resa possibile così la creazione di nuovi metodi che integrano i protocolli già presenti per le interazioni "vicine" ai macchinari con quelli per interazioni da remoto.

Nella Industrial IoT ci si avvale infatti spesso di sistemi di telecontrollo per lasciare sì un punto di interazione con l'uomo, ma soprattutto per effettuare decisioni nella produzione in base alle statistiche generate sui dati elaborati per evitare stop imprevisti. Infatti, nonostante sistemi di controllo elettronici sempre più affidabili, già nel 2005 la catena produttiva dell'industria automobilistica negli U.S.A. stimava una perdita di 22.000$ al minuto [2] in caso di blocco non programmato. Sempre negli U.S.A., indagini più recenti della Food Industry Association hanno appurato che il 25% dello spreco nella filiera alimentare è a causa di una errata gestione dei rifornimenti ed errata programmazione della produzione [3]. Critica è quindi una gestione oculata dei dati per evitare congestioni ed il rischio di perdita di informazioni potenzialmente fondamentali.

## 1.2 Edge Computing e Industria 4.0

Ogni volta che nelle telecomunicazioni si parla di segnali, quindi informazioni che sfruttano un vettore fisico, si deve tenere conto della possibilità di

perdita di dati dalle più disparate cause; questione tanto più evidente quanto più importante è una comunicazione vicina al real-time. La questione si complica nel caso sia necessaria una scelta, quindi un'analisi di dati, ancor più quando diversi dispositivi che condividono le stesse risorse impedendone la corretta allocazione per il *Quality of Service* previsto, che difficilmente è possibile garantire con i già diffusi servizi cloud-centric (CIoT).

- **Larghezza di banda**: l'incremento costante nella dimensione e frequenza dei dati prodotti dai dispositivi IoT mette a dura prova la capacità delle reti di comunicazione;

- **Tempi di latenza e continuità** (dei servizi): impianti industriali, sanitari e finanziari tollerano malamente situazioni in cui la distanza con i servizi di cloud e discontinuità in dispositivi che necessitano di un continuo monitoraggio per il corretto funzionamento.

- **Risorse limitate**: spesso i dispositivi di acquisizione dati sono mobili e alimentati a batteria, quindi sono per ragioni energetiche limitati nei calcoli complessi ed utilizzati meramente per l'inoltro delle acquisizioni demandando calcoli e scelte a servizi esterni.

- **Sicurezza**: tutti i vincoli a cui un dispositivo è sottoposto ne impediscono spesso una completa protezione da attacchi.

Principalmente per questi motivi l'esclusività nell'uso dei servizi cloud è impraticabile e si è reso necessario il cambio architetturale fulcro dell'Industria 4.0. La capacità di calcolo viene infatti decentrata verso aziende stesse, ad un livello di "bordo", al "ciglio", da cui il nome "Edge" in Edge Computing: subito prima del mare della rete Internet e di un potenziale datacenter privato o meno. Idea ispirata dall'introduzione dei Content Delivery Network (CDN) dove si è visto come pochi hop (distanza logica) nell'instradamento garantiscano una maggiore affidabilità e tempi di risposta brevi anche per flussi dati di dimensioni importanti come per lo streaming multimediale.

L'industria 4.0 è quindi mossa dalla cattura di eventi, spingerà per nuove architetture di business che supportino disponibilità continue, scalabilità, ripristino automatico da situazioni critiche ed estensibilità dei sistemi dinamica. Il modello EDA (Event Driven Architecture) agevola le organizzazioni ad essere pronte a rispondere a qualsiasi segnale in quanto ogni momento può generare un'opportunità di investimento, purché la convergenza di eventi sia favorevole [4]. Questo grazie al fatto che sempre più

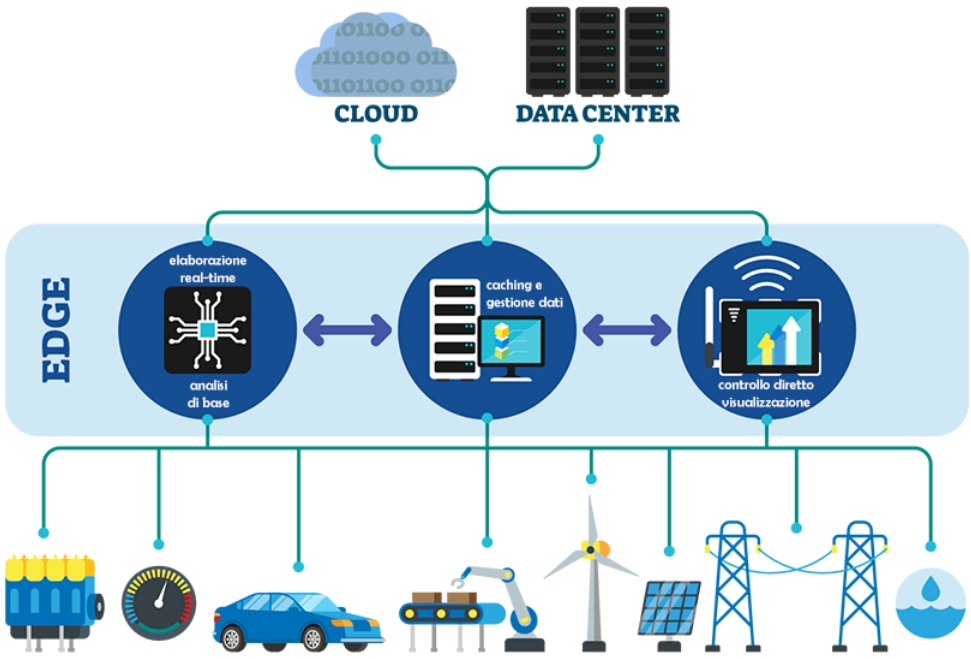



**Figura 1.1:** Schema Edge Computing

scelte sono effettuate in autonomia così da far diventare gli stessi dispositivi IoT parte attiva di una struttura aziendale, lasciando a quest'ultima la possibilità di focalizzare le forze sul business. Edge e Fog computing (FEC) si rivelano così complementari al cloud computing specificatamente ai settori SCANC [5]:

- **Storage** (archiviazione): corrisponde allo stoccaggio temporaneo in memoria e in cache tra i nodi partecipanti al network.

- **Compute** (computazione): fornisce una prima rielaborazione dei dati sotto varie forme, virtualizzando e distribuendo le risorse come servizi (nelle forme I/PaaS - *Infrastucture* e *Platform as a Service*)

- **Acceleration** (accelerazione): avviene nel sia nei collegamenti, virtualizzando i network permettendo di utilizzare più tabelle di routing in parallelo definendo network *software-defined* (SDN), sia dal lato computazionale condividendo e distribuendo il carico tra i nodi. Questa virtualizzazione consente inoltre alle reti di essere *fault tolerant*.

- **Networking** (comunicazione): omogeneizza le interconnessioni sia verticalmente che orizzontalmente adattando i vari protocolli alle necessità energetiche e prestazionali a seconda dei dispositivi.

- **Control** (controllo): si permette una personalizzazione del software in maniera dinamica, si lasciano ai dispositivi più vicini alcune decisioni e si agevola l'interfacciamento tra più sistemi di controllo direttamente nel lato edge, diminuendo così anche i rischi relativi alla sicurezza, delegando localmente (sui nodi) i servizi di autenticazione, identità e protezione dei servizi virtualizzati.

Concetto introdotto nel 2012 dalla Cisco per descrivere le sfide da affrontare nell'ambito IoT in un convenzionale ambiente di cloud computing. Il termine è spesso usato in alternativa quello di edge computing ma formalmente è considerato sottocategoria di quest'ultimo [6], nonostante offra le stesse capacità gestionali dei dati e si piazzi allo stesso livello, non è organizzato in una struttura omogenea, né è controllato da un unico ente.

### 1.2.1 Sides & Fields

Come accennato nella precedente sezione, l'elaborazione a stretto contatto con le "sorgenti di dati" è il nuovo ambiente su cui poter focalizzare ricerca hardware, ma soprattutto software: "south-side" è una locuzione che rappresenta, in senso verticale, la rete di sensori installata sugli apparati, com-

prendendo quindi il livello fisico in cui i dati sono prodotti fino al confine dell'edge computing, dove le acquisizioni subiranno un primo filtraggio.

Uno sguardo invece orizzontale su questo lato permette di comprendere l'eterogeneità dei mezzi e la loro adozione, un esempio su tutti: i macchinari produttivi sono costosi e spesso personalizzati sulle esigenze industriali locali, il che li rende difficilmente sostituibili anche dai nuovi modelli. In questo caso si parla di "brownfield", il parco dei mezzi di vecchia generazione non studiati con in mente l'evoluzione industriale. Viceversa i "greenfield" racchiudono tutte le nuove adozioni di macchinari moderni e predisposti all'aumento dell'interconnessione tra macchine.

Salendo oltre il south-side, si incontrano i dati filtrati magari attraverso un *edge gateway*: dispositivo non necessariamente dedicato, che permette la comunicazione con il cosiddetto "north-side". Con quest'ultimo si intende tutta quella zona in cui i dati vengono archiviati, aggregati ed analizzati, quindi trasformati in informazioni; servizi spesso esterni all'azienda stessa come lo è il cloud computing.

### 1.2.2 Offloading e Data zoom-in

Obiettivo cardine è quindi poter direzionare oculatamente il flusso dati, redirigere le comunicazioni verso il canale più adatto e conseguentemente dedicare le risorse per l'analisi solo dove necessario: è questo il concetto di *offloading*. Dedicare risorse e tempo ad attività che si moltiplicano all'aumentare dei dispositivi IIoT, quindi banda, spazio di memorizzazione e capacità di calcolo, comporta costi economici [8] ed energetici [9] del servizio non marginali; risorse che potrebbero appunto essere impiegate per l'elaborazione di risposte non prioritarie.

Possono verificarsi però delle situazioni in cui i dati acquisiti portino a rilevare una situazione anomala, dove quindi è necessario diagnosticare il problema: è un difetto dei sensori oppure c'è il rischio di un malfunzionamento?

Richiedere in maniera "forzata" letture più frequenti a tutta la struttura confinante al sensore che segnala l'anomalia in maniera automatica è un

bonus auspicabile in un sistema dell'Industria 4.0. In base a quanto accennato precedentemente, prevenire situazioni di stallo della produzione, giustifica il consumo di banda disponibile in queste occasioni definite proprio *Data zoom-in*. Un bilanciamento adeguato dei dati che normalmente viaggiano nella rete non ostacola eventuali richieste eccezionali, consentendo il proseguimento parallelo delle normali routine di controllo.

Capitolo 2

# Il framework EdgeX Foundry™

## 2.1 Personalizzare l'Edge computing

Il problema di poter avere un sistema standardizzato, quindi facile da produrre, ma al tempo stesso flessibile alle modifiche richieste scaturite dalle varie situazioni è noto come *mass customization*. Si rivela una sfida di fondamentale importanza in vista degli oltre 20 miliardi di dispositivi IoT attivi globalmente nel 2020, con un tasso di adozione che aumenta ogni anno [10] e non sembra destinato a rallentare [11].

## 2.2 La piattaforma di sviluppo

Il progetto si propone come integrazione software per fornire capacità di decisione e possibilità di personalizzazione, grazie ad una struttura scalabile offerta dal framework software su cui è basato: EdgeX Foundry. Una piattaforma open source, vendor-neutral e completamente agnostica del sistema operativo che offre un ecosistema di componenti (microservizi) pronti all'uso, rendendola utilizzabile anche su elaboratori non dedicati. L'ampia flessibilità è fondamentale per aumentare il più possibile la fetta di industrie che potranno raggiungere gli standard di efficienza della Industria 4.0.

Sviluppato inizialmente da Dell Inc. come software per i loro gateway IoT, in partnership con The Linux Foundation è stato annunciato alla Fiera di Hannover (Hannover Messe) in contemporanea con San Francisco nell'Aprile 2017 e rilasciato sotto licenza *copyleft* (Apache 2.0). Grazie alla community a supporto degli sviluppatori venutasi a creare, l'intento di omogeneizzare il mercato dell'IoT edge computing si è concretizzato con il rilascio della prima versione stabile evolvendosi fino all'ultima release chiamata "Geneva", seconda revisione della versione stabile (v1.2), sulla quale si basa il progetto. L'obiettivo di mantenere la portabilità del codice è stato reso possibile adottando come linguaggio di programmazione Go assieme alla gestione "a container" dei servizi essenziali con Docker.

### 2.2.1 Architettura

Questo framework è studiato per lavorare a livello edge, ha infatti la possibilità di comunicare direttamente con la rete di sensori attraverso dei microservizi dedicati, uno per protocollo, che una volta compilati vivono come normali programmi su un qualsiasi elaboratore multiprogrammato moderno: questo implica che il sistema non potrà essere veramente real time; per scopi in cui la risposta in tempo reale è critica, è stato appositamente creata una piattaforma complementare, chiamata Edge XRT [12].

Le API RESTful attraverso la quale i microservizi di cui il framework è composto comunicano, assieme alla capacità di poter effettuare scelte grazie al motore logico Kuiper, costituiscono un accoppiamento lasso delle componenti, permettendone l'uso del core anche nel tier del *fog computing*, al quale i servizi di più basso livello possono inviare le informazioni.

La piattaforma è completamente basata su microservizi indipendenti grazie ai container Docker, ma cooperanti attraverso specifici endpoint REST, ognuno in ascolto sulle rispettive porte TCP. Utilizzare i container per i vari servizi, ne garantisce portabilità e leggerezza d'esecuzione: la virtualizzazione del Docker-engine comprende solo l'hardware e non l'intero sistema operativo come nelle macchine virtuali tradizionali, così i programmi nei container possono usufruire direttamente delle routine del kernel

del sistema operativo ospitante (host), indipendentemente da quale sia in realtà, come fossero programmi compilati.

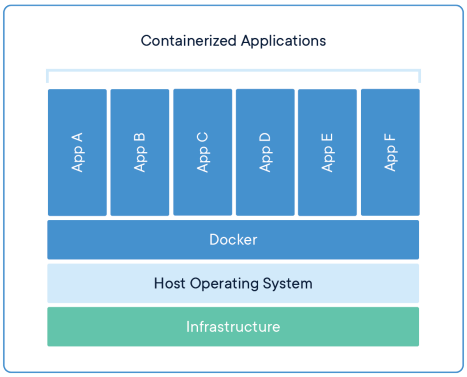



**Figura 2.1:** Container Docker

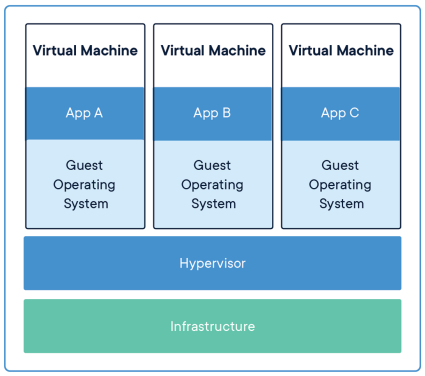



**Figura 2.2:** VM tradizionale

L'installazione viene eseguita automaticamente attraverso lo strumento Docker Compose, in cui un file di configurazione chiamato Dockerfile specifica dove recuperare le immagini dei container, così da effettuarne il download (se percorso remoto) e l'installazione in locale.

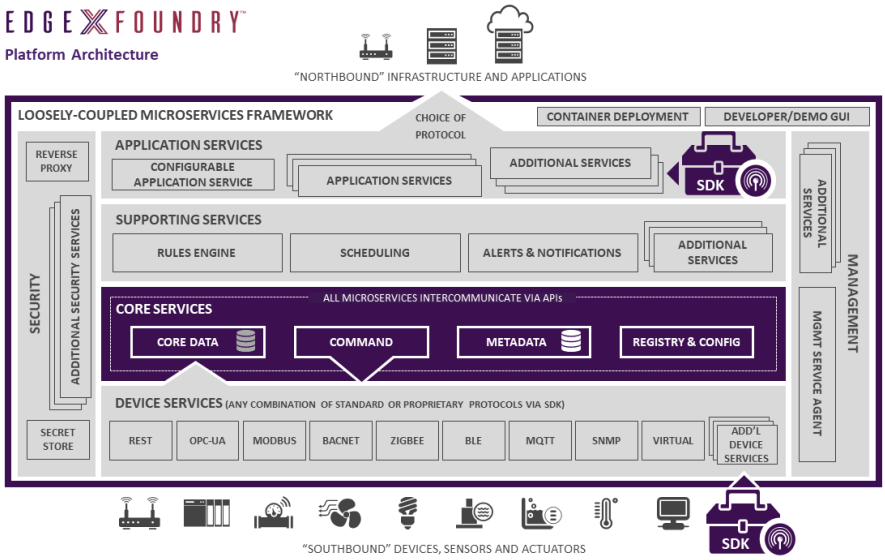



**Figura 2.3:** Architettura EdgeX Foundry

### 2.2.2 Microservizi

Con "microservizio" si intende un paradigma costruttivo di applicativi software con un compito molto specifico che tende alla realizzazione di una singola applicazione; il termine è intercambiabile con "servizi". La tecnica tradizionale ad architettura "monolitica" prevede che sia un solo programma ad occuparsi di tutte le funzionalità, adatta ad applicazioni piccole o comunque poco soggette a cambiamenti. Questo approccio infatti richiede ai programmatori molto tempo anche per piccole modifiche in quanto ognuna comporta test su tutta l'applicazione, quindi la conoscenza dettagliata della stessa in ogni sua parte.

Dividere quindi l'applicativo in base allo stack come nel paradigma multi-tier o alle funzionalità di business in più segmenti cooperanti come nel SOA (Service Oriented Architecture) ha portato ad avere applicazioni scalabili ma nel primo caso con problemi equivalenti ad applicazioni monolitiche o nel secondo caso alla creazione di livelli intermedi per mettere in comunicazione le parti, rendendo troppo costoso l'aggiornamento delle stesse applicazioni.

La soluzione adottata in EdgeX Foundry è quella di decomporre funzionalmente l'applicativo in servizi, assegnando cioè ad ognuno un compito specifico seguendo quindi uno dei principi architetturali adottati in comune dagli stessi sviluppatori:

"*EdgeX Foundry must be extremely flexible. Any part of the platform may be upgraded, replaced or augmented by other micro services or software components.*"

Ci si vuole cioè avvicinare all'architettura ideale a microservizi basata sul principio di design della singola responsabilità (SRP): fare in modo che la piattaforma sia modulare, quindi con servizi a compiti molto precisi, permette che gli stessi possano essere sostituiti o migliorati da altri microservizi in qualsiasi momento e garantisce massima flessibilità e personalizzazione. Il framework offre diverse tipologie di servizi, organizzati su quattro livelli di servizi base che costituiscono la struttura principale, alla quale si aggiungono ulteriori due livelli di ampliamento (si veda Figura 2.3).

#### 2.2.2.1 Core Services

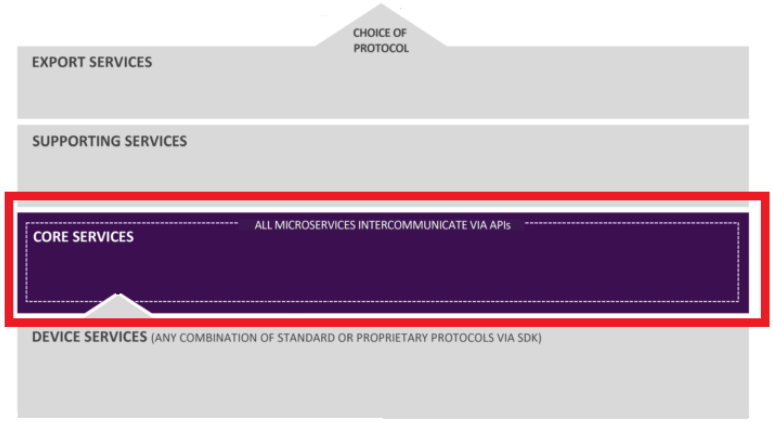



**Figura 2.4:** Posizionamento nella struttura

Questo livello appartenente alla struttura principale permette di coordinare tutte le funzionalità della piattaforma, conoscendo quali dispositivi sono connessi, le relative configurazioni e quali informazioni vengono scambiate. È il livello principalmente responsabile delle performance del sistema, in quanto ogni lettura o comando, proveniente o destinato ai dispositivi IoT (comunicazione verticale) deve passare per le API esposte dai suoi componenti a seconda della richiesta da soddisfare.

- **Core Data**, offre un un supporto di archiviazione dei dati letti dai dispositivi collegati alla piattaforma e le relative letture fino a che non vengono inviate ad altri servizi per essere esportate verso il north-side. Si occupa in particolare di effettuare un marshalling dei dati inviati come letture dai dispositivi nel south-side strutturandoli in oggetti (JSON) che, aggiungendo altre informazioni contestualizzanti come degli identificativi, unità e provenienza, prendono il nome di "eventi" (non hanno infatti un destinatario preciso) e vengono pubblicati nel *message bus*: offerto da ZeroMQ (licenza LGPLv3) e basato sul paradigma *publisher/subscriber* su socket TCP (porta 5556), rendendoli usufruibili dai Core Services. Si può impostare la persistenza

o meno delle letture stesse. Offre un primo livello di protezione e sicurezza sui dati letti dai sensori permettendo letture solo attraverso l'interfaccia API RESTful esposta. Il microservizio si basa su Redis, non un vero e proprio database ma un deposito di strutture dati (coppie chiave-valore) in memoria volatile (in-memory data structure store), estremamente veloce e leggero proprio perché non richiede accesso alla memoria secondaria, molto più lenta rispetto alla RAM; solo sporadicamente viene salvata una copia della struttura in memoria in modo permanente, copia richiamata solo all'avvio.

- **Core Command**, si occupa di esporre i comandi disponibili nei vari dispositivi collegati, esponendo un interfaccia omogenea per essere richiamati da altri microservizi e applicazioni interne al dispositivo su cui è in funzione EdgeX Foundry. Acquisisce le informazioni riguardo i dispositivi dal Metadata Core Service ed esegue i comandi chiedendoli ai rispettivi Device Service, non è infatti in grado di comunicare direttamente coi dispositivi: funge da traduttore di comandi provenienti dal north-side ai dispositivi nel south-side, previene così interazioni erronee o involontarie con i dispositivi e sensori.

- **Core Metadata**, ospita tutto ciò che serve sapere dei dispositivi che si registrano sulla piattaforma: tipologia del dispositivo, dei dati che tratta e i comandi esposti dallo stesso attraverso una configurazione strutturata specificata in file YAML. In particolare, per ogni dispositivo che si vuole aggiungere viene richiesto di esporre determinate proprietà, dalle più generali per l'identificazione (una breve descrizione, il modello e il fabbricante), le risorse che mette a disposizione (tipologia delle letture), gli attributi associati al dispositivo per comunicarvi ma soprattutto i comandi disponibili. La persistenza di tali informazioni è sempre garantita da Redis ma viene creato un livello di astrazione del database per coesistere con il microservizio Core Data.

- **Configuration & Registry**, sono una combinazione di servizi che rac-

chiudono informazioni riguardo gli altri componenti della piattaforma, fornendo appunto anche un servizio di configurazione centralizzato. La funzione più importante del registro è quella di fornire all'avvio dei microservizi le coordinate (porta e nome host) corrette per comunicare con gli altri da cui dipendono, per poi invocarlo in autonomia. Al momento, questa facoltà è erogata attraverso Consul, un servizio in grado di scoprire automaticamente gli altri sotto lo stesso dominio offrendo anche un'interfaccia grafica dove poter controllare lo stato di ogni microservizio.

#### 2.2.2.2 Device Services


**Figura 2.5:** Posizionamento nella struttura

Questa classe di servizi si occupa di connettere un qualsiasi dispositivo alla piattaforma Edgex, trasformando i valori presentati dallo stesso in una struttura dati utilizzabile dai Core Service di Edgex. Inoltre permettono anche di attuare i comandi inviati attraverso l'interfaccia comune del Command Service, traducendo, in modo inverso, la richiesta cosicché sia comprensibile ed attuabile dal dispositivo.

I microservizi creati per questo scopo effettuano quindi un'astrazione software dei dispositivi associati, impacchettando in una API il protocollo

di comunicazione, il driver o firmware e il dispositivo stesso. EdgeX Foundry è al momento fornito con solo alcuni servizi facenti riferimento ai più comuni protocolli di comunicazione (MQTT, SNMP, Modbus), per questo viene rilasciato un SDK per poter scrivere e compilare il proprio servizio per il protocollo desiderato. Questo SDK, paragonabile ad una libreria, permette di focalizzare l'impegno ed il tempo dello sviluppatore sulle specifiche della comunicazione del dispositivo che si vuole collegare ad Edgex, esso è disponibile per i linguaggi C e Go.

In pratica il Software Development Kit fornito aiuta nelle routine che ogni Device Service deve effettuare: registrazione al Core Metadata, leggere la configurazione predefinita, interfacciarsi con il dispositivo stesso (detto *provisioning*, ad esempio instaurare una connessione BLE), monitorare ed informare Edgex riguardo lo stato del dispositivo, inviare le letture e reagire ai comandi REST impartiti.

#### 2.2.2.3 Application Services

Sono applicativi *on-premises*, cioè installate sulla macchina sui cui operano, atte all'esportazione dei dati letti ed elaborati dagli altri livelli sottostanti, in maniera automatizzata, sviluppati anch'essi attraverso l'uso di un SDK (al momento solo per il linguaggio Go) offerto con EdgeX Foundry. La loro architettura è basata su una *pipeline di funzioni* che viene eseguita da un innesco (trigger): gli eventi di lettura provenienti dai Device Service, anticipati prima. La serie di funzioni che il flusso di dati segue subito dopo un trigger servono come filtro in base al nome del dispositivo da cui è originato ed alla tipologia della lettura. Solo dopo viene fatto un marshalling dell'evento così da poterlo indirizzare all'endpoint nel north-side interessato, con una richiesta HTTP o pubblicazione su topic MQTT.

#### 2.2.2.4 Supporting Services

Sono un insieme di servizi a monte di tutto il sistema fin ora descritto che donano alla piattaforma Edgex una capacità analitica e di personalizzazione

- **Alerts & Notifications** è il segmento di applicazione che si occupa appunto di notificare ad un operatore od un qualsiasi altro sistema, una qualsiasi informazione che viene generata da un qualsiasi altro microservizio attraverso l'interfaccia API REST esposta. Al momento Edgex offre a questo scopo la possibilità di inviare email o contattare un endpoint REST. La notifica viene sempre prima resa persistente in memoria e può avere un grado di priorità "normale" o "critico", nel primo caso viene evasa dopo un certo periodo deciso dallo scheduler, nel secondo invece si provvede all'immediata evasione senza attendere di inviare conferma al client che l'ha generata.

- **Rules Engine** è basato su EMQ X Kuiper, un pacchetto software open source con licenza Apache 2.0, dedicato all'analisi di stream di dati, programmabile con regole in SQL. Collegato al message-bus interno di Edgex (via ZeroMQ) ne converte i messaggi in uno stream di dati detto *source*, vi applica quindi le regole programmate e ne inoltra l'elaborazione attraverso il componente *sink*, nello stesso message bus di Edgex oppure ad un qualsiasi endpoint REST o MQTT. Può essere impostato manualmente per i dispositivi più critici con i relativi identificativi dei dispositivi hardcoded nelle regole, oppure può segnalare ad altri servizi (remoti e non, purché in grado di dialogare con la piattaforma Edgex) di prendere provvedimenti in maniera dinamica segnalando il solo nome del dispositivo.

- **Scheduling** funge da "orologio" interno alla piattaforma che può far eseguire una qualunque funzione nei microservizi interni. Ad esempio, dopo un determinato intervallo di tempo, di default viene ripetuta la pulizia della memoria persistente degli eventi correttamente esportati (operazione Scrubbed Pushed) o troppo vecchi (operazione ScrubAged).

#### 2.2.2.5 Servizi addizionali

Sono servizi trasversali all'intera piattaforma EdgeX Foundry, sono richiesti per funzioni esterne alla valutazione dei dati, che completano il quadro delle funzionalità offerte ed intrinseche alla gestione interna a moduli.

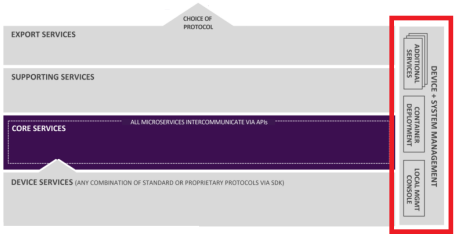



**Figura 2.6:** Servizio SMA

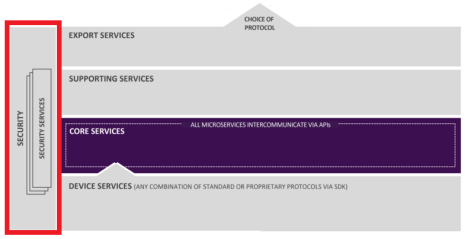



**Figura 2.7:** Servizio di sicurezza

- **System Management Agent** abbreviato con SMA, si occupa di avviare, fermare o riavviare i microservizi, assieme a controllarne lo stato attraverso una propria API che può essere richiamata per effettuare tali operazioni su più microservizi, agendo di fatto come multiplex.

- **Security** è un servizio non essenziale al funzionamento della piattaforma, ma fondamentale in tutti quegli ambienti in cui ogni comunicazione deve essere protetta, crittografandola, in ogni livello in cui avviene. Una volta abilitato permette all'amministratore di autenticarsi e gestire l'accounting degli utenti e crea dei JSON Web Token (JWT) per gestire gli accessi ai microservizi. Inoltre vengono avviati i due componenti:

  - *security store*, che come suggerisce il nome è un sistema sicuro che può generare, depositare e recuperare i segreti come certificati e chiavi di accesso.

  - *API gateway*, il quale è usato come "reverse proxy" per restringere (in base alla definizione di una ACL) l'uso delle risorse REST ed effettuare un controllo generale degli accessi.

## 2.3 Flusso di lavoro

La quantità maggiore di dati scorre dalla parte più a "sud" del sistema al livello superiore: nei Core Services e Application Services, dove vengono rispettivamente analizzati, filtrati ed inoltrati a servizi esterni alla piattaforma, come nel caso dell'applicativo sviluppato.

La piattaforma EdgeX Foundry dirige quindi tutte le componenti fin qui descritte in base alla configurazione che può essere aggiornata anche da sistema avviato, modificando, aggiungendo o rimuovendo i dispositivi attraverso specifici endpoint REST.

### 2.3.1 Dispositivi e profili

Ogni sorgente di dati, quindi ogni sensore, deve essere preventivamente conosciuta, cioè registrata su EdgeX Foundry, dichiarando operatività e risorse offerte per renderle disponibili e riconoscibili ai servizi di tutti i livelli. Tale operazione viene eseguita dopo aver avviato la piattaforma ed essersi assicurati che il servizio facente riferimento al protocollo usato dal dispositivo sia operativo.

La registrazione prevede appunto che un dispositivo, deve essere corredato da un profilo che ne descrive formalmente le capacità dello stesso: infatti ad uno stesso profilo possono far riferimento più di un dispositivo, ad esempio risulta ridondante caricare più profili di macchinari identici ma posizionati su più catene che producono lo stesso bene. Questo documento dichiarativo è racchiuso in un documento YAML, un formato di serializzazione dati che con una sintassi minimale permette una descrizione esaustiva come nel JSON (ne è ufficialmente subset) ma sfruttando solo caratteri di indentazione risulta più leggero rimanendo sempre human-readable. La procedura è sintetizzabile in questi passaggi:

1. viene eseguita una richiesta POST, che effettua il caricamento del suddetto file, al servizio Core Metadata, quindi salvato in modo persistente;

2. in un qualsiasi momento successivo può quindi essere registrato un dispositivo con una seconda richiesta POST al Core Metadata, il cui corpo è una descrizione formale in JSON che indica dove raggiungere il dispositivo stesso (ad esempio indirizzo IP e porta);

3. di conseguenza, in automatico viene interpretato il profilo e personalizzato dal Core Command, creando endpoint univoci (pseudo-random) per richiamare i comandi, che siano di lettura o scrittura, rispettivamente differenziati da tipologia di richiesta GET o PUT e rispettivo URI.

A seguito un estratto del profilo e della dichiarazione del dispositivo utilizzati nel progetto. Nel profilo si può leggere il dettaglio della risorsa che viene esposta attraverso il protocollo Modbus-TCP. Nel JSON usato per dichiarare il macchinario si porta all'attenzione del lettore il riferimento al profilo ed agli eventi di polling "autoEvents" che verranno registrati ed eseguiti dalla piattaforma Edgex Foundry.

```
{   "name" :"HMI Simulator 1",
    "description":"Dispositivo HMI simulator",
    "adminState":"UNLOCKED",
    "operatingState":"ENABLED",
    "protocols":{
       "modbus-tcp":{
          "Address" : "192.168.1.28",
          "Port" : "1502",
          "UnitID" : "1"}
    },
    "labels":[
       "interface",
       "simulator",
       "modbus TCP"],
    "service":{"name":"edgex-device-modbus"},
    "profile":{"name":"HMI_5"},
    "autoEvents":[{
          "frequency":"3s",
          "onChange":false,
          "resource":"DatiCiclo"
     }]
}
```

**Listing 1:** Dichiarazione JSON del dispositivo usato per il macchinario Modbus simulato

```
name: "HMI_5"
manufacturer: "SACMI"
model: "XYZ145"
description: "Dispositivo HMIsimulator"
labels:
  - "modbus"
  - "interface"
  - "simulator"
deviceResources:
    -
        name: "DatiCiclo"
        description: "Dati ciclo"
        attributes:
            { primaryTable: "HOLDING_REGISTERS",
                startingAddress: "1" }
        properties:
          value:
            { type: "UINT16", scale: "1"}
          units:
            { type: "String", readWrite: "R",
                defaultValue: "min"}
deviceCommands:
  -
    name: "DatiCiclo"
    get:
      - { index: "1", operation: "get",
          deviceResource: "DatiCiclo" }
coreCommands:
  -
    name: "DatiCiclo"
    get:
      path: "/api/v1/device/{deviceId}/DatiCiclo"
      responses:
        -
          code: "200"
          description: "Get the DatiCiclo"
          expectedValues: ["DatiCiclo"]
```

**Listing 2:** Estratto del profilo usato per il macchinario Modbus simulato

Capitolo 3

# Struttura del progetto

## 3.1 Introduzione a SCADA

La sigla SCADA (Supervisory Control And Data Acquisition) indica un'infrastruttura informatica incentrata sulla telemetria (misura e controllo remoti) di un intero impianto produttivo. Sviluppato per dare un significato universale alla definizione di controllo remoto, tipico nell'Industria 4.0: dispositivi a controllo remoto (RTU) comunicano con controller (MTU) spesso dotati di un'interfaccia uomo-macchina (HMI).

Caratteristica chiave è la capacità di interconnettere i vari componenti di un impianto produttivo tra i vari livelli:

- **Livello 0**: il livello più basso, dove avviene fisicamente la lettura sui sensori (da semplici contatori alle telecamere per l'interpretazione dei movimenti degli operatori) e dove gli attuatori (come deviatori nei rulli trasportatori) eseguono i rispettivi comandi lungo la catena produttiva.

- **Livello 1**: è il primo livello in cui si introduce una prima componente digitale, che fornisce un'interfaccia di controllo associabile a dispositivi elettronici come PLC e RTU integrati sui macchinari. Qui avviene il principale scambio di informazioni, uniformando protocolli proprietari e aperti per renderli usufruibili ai livelli superiori. Per questo è

spesso ridondante nelle aziende anche se un blocco qui non comporterebbe lo stop del processo produttivo, verrebbe meno la continuità dell'efficienza programmata e tutti i controlli automatizzati superiori.

- **Livello 2**: comprende la parte informatica di supervisione fornendo una interfaccia grafica (HMI) di controllo diretto ed in tempo quasi reale sulla catena produttiva che può essere usata per un controllo umano, prioritario a quello automatizzato, dei livelli sottostanti.

- **Livello 3**: è il livello di controllo del processo produttivo, ne fornisce una visione generale sull'efficienza e permette di coordinare più catene.

- **Livello 4**: astrae questi processi elaborandone i risultati fornendo previsioni così da poter programmare rifornimenti e gestire variazioni nella capacità produttiva.

Ogni livello può essere messo in comunicazione con metodi differenti a seconda delle esigenze organizzative e produttive, variando dal wireless al cablato, ma sempre orientati all'adozione di standard nei protocolli di comunicazione, facilitando

Quest'architettura inoltre permette di gestire eventi imprevisti ad ogni livello, è infatti il motivo principale per cui viene adottata ed è in continua evoluzione: le anomalie possono essere rilevate, ad esempio attraverso i sensori si può osservare malfunzionamento di un macchinario (livello 0), oppure previste, come una penuria di materie prime nei magazzini in caso di aumento della produzione (livello 4).

La tipologia di notifica dipende dal livello in cui scaturisce il problema e dalla gravità dello stesso, si può infatti limitare al segnalare nella grafica delle interfacce HMI un'azione eseguita in automatico in quanto programmata, oppure richiedere un intervento umano per porvi rimedio.

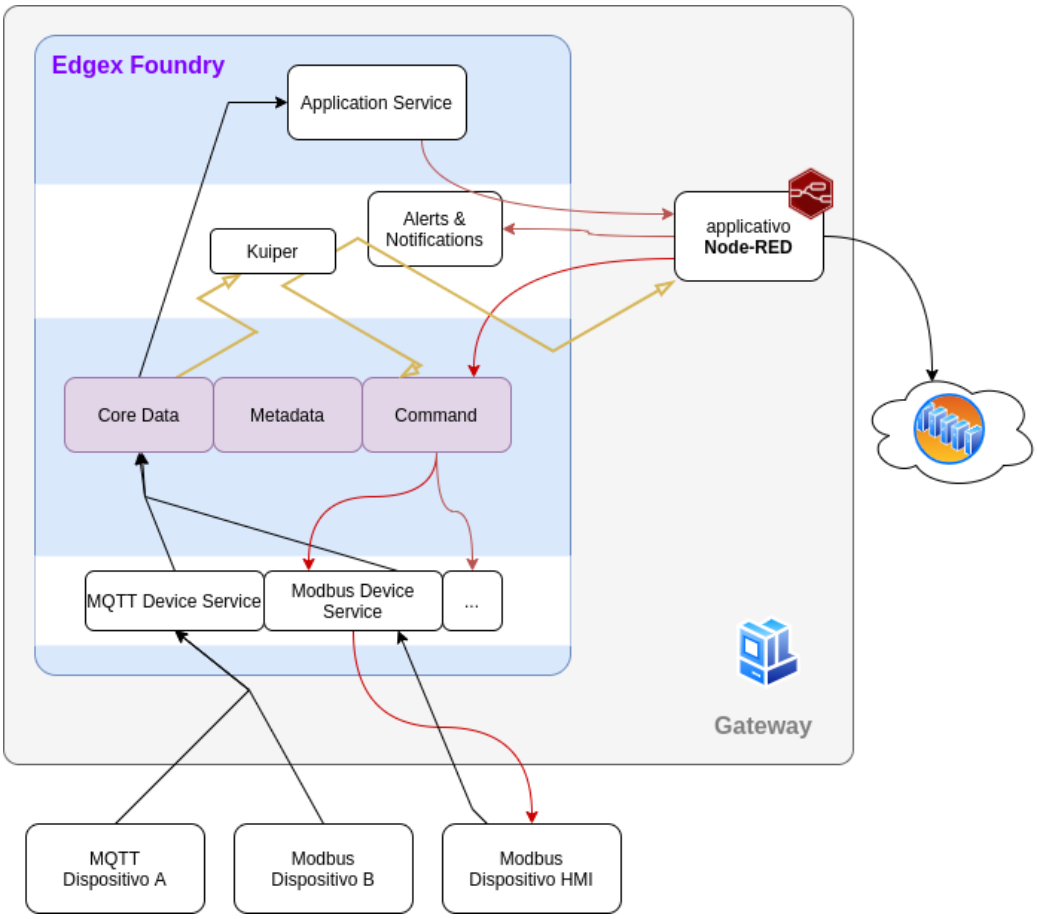



**Figura 3.1:** Gestione del flusso dati

## 3.2 Il progetto

Il progetto si articola su più livelli donando caratteristiche tipiche dell'ambiente SCADA ad EdgeX Foundry, cercando di creare di fatto un sistema per l'edge computing virtuale atto a ricreare situazioni verosimili ma in ambiente controllato. Per questo si è suddiviso il progetto su tre livelli, rappresentanti quanto descritto, che si riferiscono ad altrettanti dispositivi simulati:

- **Livello produzione**, il più basso, ospita i macchinari simulati nel protocollo Modbus su cui si è scelto di indagare.

- **Livello edge**, in cui è ospitato Edgex Foundry, i suoi componenti e le personalizzazioni integrate, sviluppate per gestire offloading e data zoom-in.

- **Livello cloud**, dove si è pubblicato un endpoint a cui far puntare l'esportazione dei dati, quindi la visualizzazione degli stessi attraverso un interfaccia human-friendly.

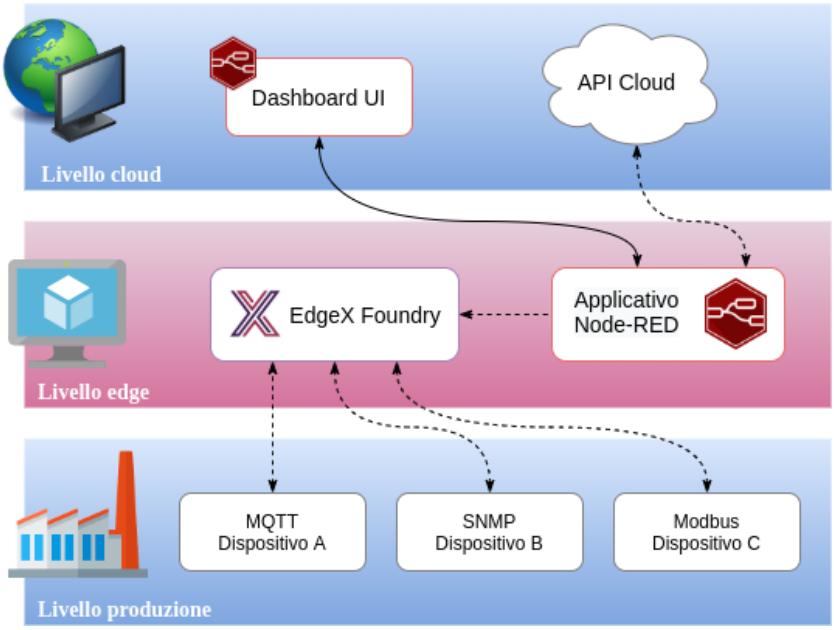



**Figura 3.2:** Struttura Logica

Come anticipato, il sistema è stato sviluppato principalmente al livello intermedio per cercare di migliorare l'operatività offerta da Edgex Foundry: direzionare in maniera personalizzabile il flusso dati aggiungendo un componente plug-and-play sviluppato ad-hoc sfruttando la modularità della piattaforma stessa.

Gli altri livelli sono stati resi il più possibile generici, attuando solo le configurazioni necessarie per la corretta operatività del tutto, eliminando qualsivoglia vincolo di applicabilità che potrebbe sorgere in ambito operativo.

Il sistema è quindi stato ideato per lavorare in due stati ben definiti da una soglia di attenzione, configurata nell'applicativo sviluppato e nel Rules Engine per macchinari che richiedono un intervento automatizzato in caso di guasto:

- **Normal behaviour**, è il "contesto normale" in cui nessuna lettura dei sensori presenta anomalie. Solo una parte degli eventi viene esportata oltre il nodo e inviata all'applicativo per la visualizzazione nell'interfaccia, cioè lo stretto necessario per poter affermare che è tutto corretto nei macchinari monitorati nel lasso di tempo prestabilito (può variare a seconda dei contesti). Comportamento che effettua quindi *offloading* delle informazioni evitando di sovraccaricare il computer ospitante la piattaforma in elaborazioni non necessarie, così da poter dedicare risorse non appena si verifica una situazione di necessità.
- **Critical behaviour**, si presenta al primo evento di un sensore che riporta una lettura fuori norma. Prontamente segnalata e, adottate eventuali contromisure predefinite come regole su Kuiper, il sistema sviluppato provvede a richiedere un aumento del numero di letture sul sensore o sul macchinario intero. Questa operazione fa coesistere il principio di offloading assieme allo zoom-in dinamico sui dati interessanti. Il sistema prevede che, una volta innescato, si possa ignorare questo stato di allarme attraverso un intervento manuale, mantenendo la segnalazione di criticità anche dopo che è cessata magari a seguito della contromisura automatizzata messa in atto. Infatti le letture in export al north-side vengono comunque limitate alle sole critiche.

## 3.3 Livello dei dispositivi

Il progetto qui discusso si è basato sulla ricerca del dott. ing. Lorenzo Patera su un middleware per l'Industria 4.0 che permetta l'uniformazione delle comunicazioni verso il cloud alle industrie e viceversa. Come in questo

```
[
  {
    "id": "0b92fa0c-a3f8-4e35-bb4e-d649e723357b",
    "created": 1599568331567,
    "origin": 1599568331563345773,
    "device": "HMI-dev_20",
    "name": "DatiCiclo",
    "value": "3",
    "valueType": "Uint16"
  }
]
```

**Listing 3:** Esempio di evento Edgex (JSON)

progetto, si è reso necessario simulare i dispositivi per la produzione di dati realistici atti a verificare l'applicabilità della soluzione proposta.

### 3.3.1 Simulatore

Base del livello, è stato sviluppato in Python come programma indipendente atto a simulare fedelmente un macchinario reale nel suo stato, comunicandolo attraverso Modbus-TCP. Mette a disposizione 7 sensori che pubblicano su 190 indirizzi, aggiornandoli ciclicamente in intervalli di tempo configurabili.

Questo programma ricade nella tipologia HMI (Human-Machine Interface) in quanto è esso stesso a fornire dei dati assieme ad esporre metodi utili al controllo del macchinario, in questo caso attraverso il protocollo Modbus-TCP.

Come descritto nel secondo capitolo sarà un sevizio interno alla piattaforma Edgex Foundry a fare le veci del client nel chiedere via Modbus le letture, programmate al momento della registrazione.

## 3.4 Livello edge

Questo è il livello di confine, in cui viene effettuata la scelta per redirigere il traffico dati e si compone di un'interazione tra il livello Supporting Service di Edgex e l'applicativo sviluppato.

### 3.4.1 Node-RED

Tool inizialmente sviluppato da IBM e rilasciato sotto licenza Apache 2.0, Node-RED è un tool programmabile dedicato alla gestione dei flussi di dati. Node.js è il runtime JavaScript su cui è costruito e ben si adatta ad operare con messaggi strutturati in JSON. Il modello event-driven e non-blocking su cui è basato sfrutta la tipologia di comunicazione asincrona, attraverso la quale gli eventi interni ed esterni ricevuti non fermano l'esecuzione del thread principale di esecuzione (ad esempio a causa di interazione con database), bensì vengono interpretati e gestiti; con l'interazione sincrona ogni evento sarebbe gestito "uno ad uno", tendendo a creare un collo di bottiglia.

Il paradigma "low-code" su cui si basa permette di sviluppare velocemente applicazioni senza scrivere codice, è facile da apprendere grazie all'interfaccia web ed offre un vasto supporto dalla community che per questo comprende anche i non programmatori. Il pannello di modifica è infatti raggiungibile come un normale sito web, dove si crea l'applicazione trascinando (drag and drop) i vari nodi predefiniti che, una volta collegati, potranno essere implementati automaticamente con la pressione di un tasto. I nodi sono infatti anche descritti in HTML apparendo quindi con una categoria, un colore, un nome e un'icona. La connessione di uno a un altro o più nodi è detta "flusso"

I flussi programmati sono composti da "flow object" raggruppati in "subflow" (corrispondenti alle schede nell'editor) e sono gestiti da un *flow object globale* che è responsabile per la creazione, l'avvio e lo stop di tutti i nodi che governa. Ad ogni creazione di un subflow viene dato un riferimento al flow globale così da permettere l'acesso alle variabili ed alle configura-

zioni generali. All'avvio viene creata una istanza locale per ogni nodo del subflow. Quando viene fermato un flusso avviene il contrario, fermando ogni nodo istanziato. Questo permette di iniettare nuovi nodi o modificare quelli esistenti senza fermare i flussi. Il deployment ed il mantenimento dei nodi è effettuato dal servizio di runtime di NodeJS, convertendo in JSON tutti i nodi modificati o mai implementati, da oggetti in un array in oggetti strutturati che possono essere utilizzati come flow object.

Un programma Node-RED è sviluppato quindi come un flusso di messaggi che attraversa una sequenza di nodi, ognuno dei quali può implementare un'elaborazione. I messaggi vengono passasti in oggetti Javascript chiamati `msg` con la proprietà `payload` che contiene il corpo del messaggio stesso. Ognuno è inoltre identificato attraverso la proprietà `msgid`. I valori delle proprietà sono quelli supportati normalmente nel JSON:

- Booleano;
- Numero;
- Stringa;
- Array;
- Oggetto;
- Null.

Node-RED non è però una "black-box" che permette l'uso dei soli nodi messi a disposizione: l'approccio *open-by-default* offre infatti la possibilità di personalizzare con funzioni Javascript il comportamento dei vari nodi e di crearne di nuovi direttamente con un doppio click dall'editor web. Ogni nodo rispecchia il paradigma dei microservizi, assegnando ad ognuno un ben preciso scopo.

In questo caso l'immagine del tool Node-RED è stata aggiunta al Dockerfile di EdgeX Foundry. Questo permette di risparmiare in termini di risorse computazionali evitando di creare thread dedicati per la gestione di ogni evento, quindi contenere l'utilizzo della memoria primaria e l'occupazione della CPU.

Oltre al suo scarso impatto sulle risorse dichiarato, la scelta di questo applicativo è stata motivata dalla possibilità di funzionare all'interno di un container Docker come già richiesto per Edgex, quindi direttamente integrabile senza dipendenze da gestire manualmente.

### 3.4.2 Analisi del flusso dati

Le letture dei sensori dei macchinari sono inviate all'applicativo sviluppato su Node-Red attraverso richieste HTTP POST all'endpoint REST esposto. In Figura 3.3 è riportato il flusso di lavoro programmato nell'applicativo sviluppato in cui si può notare come tutte le letture vengono filtrate dallo stesso ma una parte viene sempre aggregata ed esportata verso il cloud (Normal Behaviour). Ogni secondo viene effettuata la media delle letture pervenute da ogni dispositivo e il risultato viene inviato al servizio esterno. Questa procedura può essere personalizzata a seconda delle esigenze funzionali dei servizi di statistica nel cloud.

Nel caso invece si verifichi un evento che presenti una lettura anomala (Critical Behaviour), i filtri vengono superati rendendo automatico lo zoom-in sul flusso dati nei soli dispositivi che pubblicano letture fuori norma mentre Kuiper può intervenire immediatamente con un primo comando (hardcoded), ad esempio riducendo la velocità del macchinario. Una volta sollecitato, l'applicativo sviluppato avvierà una procedura di aggiornamento delle letture che, in questo caso su protocollo Modbus, consiste nell'aumentare la frequenza delle richieste client attraverso la modifica dell'intervallo di polling del Device Service facente riferimento al dispositivo voluto. Questo attiverà una segnalazione grafica al livello successivo come notifica per l'operatore che provvederà a disattivarlo, effettuando uno zoom-out manuale una volta verificato che le contromisure applicate sono state efficaci.

Non essendo obiettivo principale del progetto, per questione di semplicità di programmazione questo innesco avviene attraverso Node-RED stesso, in cui vengono descritti i dispositivi da osservare strettamente con i relativi comandi da chiamare in caso di situazione critica. La richiesta è effettua-

ta con un polling automatico al Core Command relativa alla lettura del sensore che restituisce uno stato anomalo.

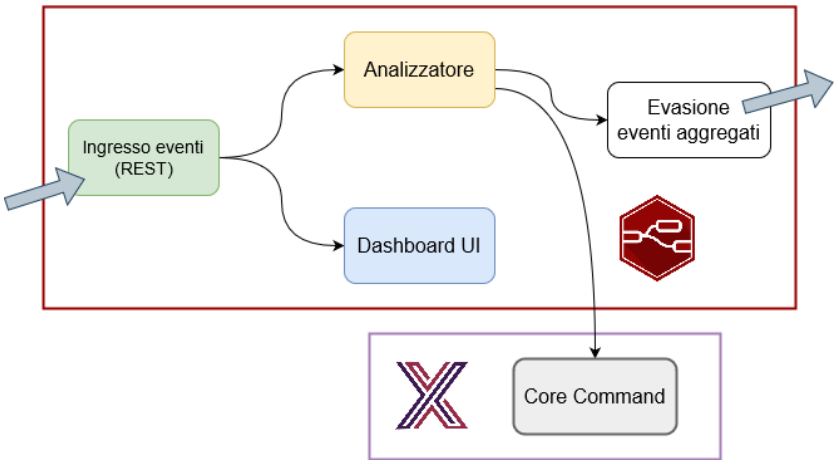



**Figura 3.3:** Gestione del flusso dati nell'applicativo sviluppato ad alto livello

## 3.5 Livello cloud

L'applicativo sviluppato si posiziona a cavallo tra questo ed il livello precedente, in quanto, con la programmazione di un solo flusso di dati da analizzare è possibile generare in automatico un pannello di controllo per la visualizzazione grafica di valori. Grazie a ciò un operatore può riconoscere uno stato di allerta, quindi intervenire apportando le modifiche necessarie per ritornare ad uno stato di normalità.

In questo livello è anche considerato un endpoint REST fittizio a cui il sistema invia le letture, simulando un sistema di calcolo remoto. Per questo motivo nella simulazione il server che lo espone è ospitato su una terza macchina virtuale, esterna al sistema EdgeX Foundry, così da non influenzare il carico sulle risorse. Node-RED permette inoltre la renderizzazione della dashboard sul dispositivo HMI di controllo (Human-Machine Interface) che la richiede, limitandosi all'invio dei soli dati filtrati al livello sottostante.

Questo pannello di controllo prevede la visualizzazione delle letture con grafiche facilmente comprensibili ed utilizzabili dagli operatori, grazie sì alla formalità ma che risulta familiare grazie ad un design semplice tipico delle applicazioni per smartphone, è infatti sufficientemente leggera da essere visualizzabile da remoto anche su dispositivi mobili.

Dato che ogni dispositivo può avere più sensori, quindi offrire più letture contemporaneamente, li si raggruppa per colonne, visualizzando per ognuna lo strumento migliore a seconda della tipologia che rappresentano. Uno switch permette di attivare o disattivare le richieste di lettura extra, per avere uno zoom-in o zoom-out su un determinato dispositivo anche manualmente.

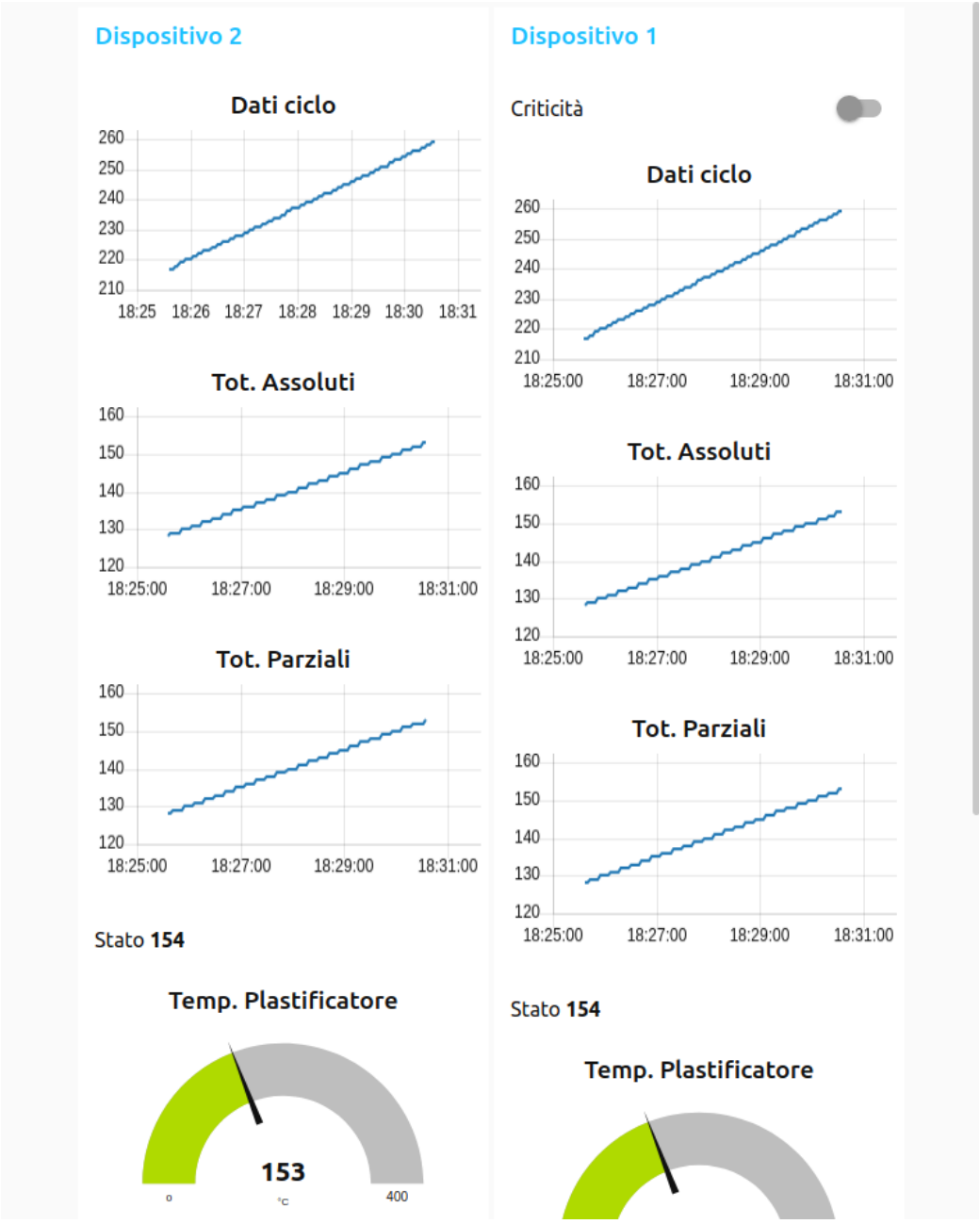



**Figura 3.4:** Schermata dashboard applicativo

Capitolo 4

# Implementazione

## 4.1 Panoramica

L'ambiente è stato simulato attraverso l'uso di macchine virtuali rappresentanti i livelli stessi su cui si articola il progetto: questo ha permesso di misurare con precisione l'impegno richiesto sulle risorse ed i tempi di comunicazione tra gli stessi. Questo è stato possibile in quanto la connessione delle VM è stata impostata come "bridged", cioè in possesso di un indirizzo IP della rete a cui è connesso l'host, comportandosi così come un dispositivo fisico.

Le rilevazioni sono potute così effettuare con le macchine virtuali ospitate in due differenti host sotto la stessa rete: il primo ed il terzo livello nello stesso computer in quanto passivi rispetto al secondo livello, che ospita la VM principale, dove è appunto eseguito EdgeX Foundry assieme all'applicativo in Node-RED.

Le stesse richieste di lettura sul dispositivo sono effettuate direttamente via Modbus-TCP, quindi passanti fisicamente attraverso la stessa rete su cui sono collegate le macchine virtuali, con la conseguente limitazione alla velocità della rete LAN di 100 Mb/s, così da poter osservare eventuali limitazioni dovute al supporto fisico.

Il computer ospitante il secondo livello ha messo a disposizione attraverso Ubuntu 18.04 LTS un processore octa-core a 3,2GHz, 16GB di RAM

DDR4 ed una connessione alla rete LAN, ospitando materialmente la macchina virtuale principale su cui Docker e quindi EdgeX Foundry sono stati installato assieme alla macchina virtuale secondaria che simula il macchinario industriale (south-side). Il servizio RESTful su cui Edgex esporta le letture (simulando il north-side) è stato invece ospitato direttamente nel computer host.

La macchina virtuale principale è stata inizialmente dotata di un processore quad-core e 2GB di RAM cercando di rispecchiare le capacità di un SBC (Single Board Computer) su cui Edgex viene dichiarato lavorare a seguito di crearvi 1GB di RAM virtuale: un Raspberry Pi 3B Plus, infatti grazie a Docker Edgex supporta anche l'architettura ARM. I test di performance sono stati tutti eseguiti su macchine virtuali x86 con sistema operativo Kubuntu 18.04 LTS a 64bit.

## 4.2 Protocolli di comunicazione

Uno dei punti forti di EdgeX Foundry è la flessibilità nelle comunicazioni che ne aumentano la flessibilità di applicazione consentendone la personalizzazione riguardo alle esigenze di chi lo adotta. Questo rimanendo di semplice utilizzo grazie al delegare in maniera distribuita le specifiche dei protocolli sui Device Service, spesso proprietari e di applicazione molto specifica. Vengono qui trasformati in letture standard per tutta la piattaforma che possono essere poi riutilizzate per qualsiasi scopo.

### 4.2.1 Transmission Control Protocol

Tutte le comunicazioni che avvengono all'interno del progetto si basano su pacchetti TCP, protocollo formalizzato nella RFC 793 (Settembre 1983) ed adottato nella maggior parte delle comunicazioni in Internet in quanto garantisce una comunicazione digitale a pacchetti affidabile. Prevede infatti di concordare una connessione tra mittente e destinatario che rimane attiva anche senza scambio di dati e viene esplicitamente chiusa da uno dei due interlocutori.

L'affidabilità nella consegna è invece conseguenza di una combinazione nell'uso del meccanismo di *acknowledgement* (ACK) con la numerazione sequenziale dei pacchetti trattati, permettendo al ricevente di ricostruire nell'ordine corretto la sequenza di byte inviata. Gli stessi ACK sono identificati così da riconoscere le stesse sequenze di bytes correttamente ricevute al livello trasporto ISO/OSI.

Per questi motivi è spesso usato sopra Internet Protocol (IP): congestione, bilanciamento del carico assieme all'intrinseca imprevedibilità della rete stessa sono causa di ritardi nell'instradamento con possibile perdita dei pacchetti generati. Astraendo dal livello applicativo tutto ciò permette una specializzazione dell'hardware per la gestione di queste peculiarità attraverso la creazione di una interfaccia chiamata *network socket*.

Mentre IP si occupa dell'indirizzamento dei pacchetti, TCP tiene traccia dei segmenti (Protocol Data Unit, PDU), cioè le singole parti in cui il messaggio viene diviso, a cui corrisponde un pacchetto IP, nel cui PDU il messaggio incapsulato prende il nome di *datagram*. La dimensione massima di un segmento TCP (Maximum Segment Size, MSS) viene specificata da ogni interlocutore all'instaurazione di una connessione ma è tendenza creare segmenti adatti ad evitare la scomposizione in più pacchetti IP (frammentazione).

Tale dimensione è stabilita in base alla Maximum Transmission Unit (MTU), cioè il più grande PDU che può essere trasmesso in un'unica transazione, derivante dalla capacità dei dispositivi di rete che il pacchetto attraversa. L'algoritmo per il scoprire tale valore è PMTUD (Path MTU Discovery) che prevede di inviare pacchetti con l'opzione "don't fragment" impostando una dimensione sempre minore finché non rientra l'ACK del destinatario che conferma come il pacchetto abbia attraversato tutto il percorso, negli altri casi il mittente riceve un messaggio ICMP "Destination Unreachable (Datagram Too Big)".

Una connessione TCP viene creata da un mittente inviando al destinatario un segmento con il flag SYN, il quale, se disponibile, rispondera con un segmento con flag ACK, alla cui ricezione viene risposto con un segmento di ACK e solo dopo il destinatario inizia l'invio dei segmenti contenenti il

messaggio da recapitare.

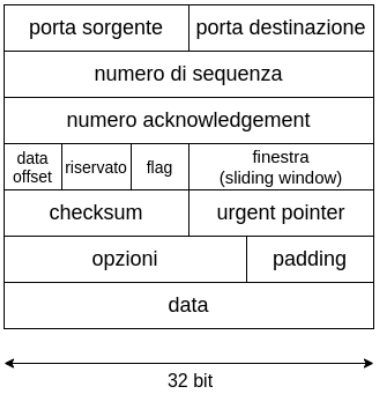



**Figura 4.1:** Segmento TCP

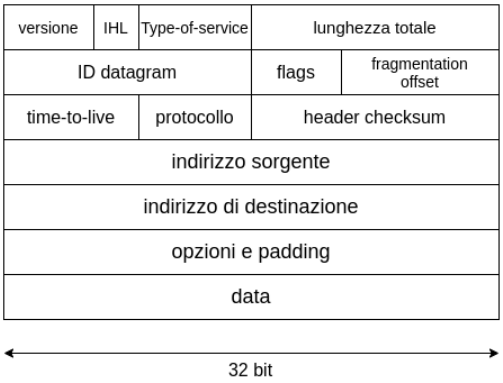



**Figura 4.2:** Pacchetto IP

Un segmento TCP è composto da *header* e *data*, quest'ultima contiene il messaggio vero e proprio, mentre la prima descrive funzione del pacchetto (tramite flags), checksum dello stesso header, porta sorgente e porta destinazione. L'indirizzo dal dispositivo è affidato ad IP, mentre la porta che TCP specifica, identifica una particolare applicazione in esecuzione nel dispositivo indirizzato. Ad esempio i Core Service rispondono alle seguenti porte:

| *Servizio* | *Porta* |
|---|---|
| Core Data | 48080 |
| Core Data (ZeroMQ) | 5563 |
| Core Metadata | 48081 |
| Core Command | 48082 |

L'incapsulamento in IP prevede quindi di arricchire di informazioni il pacchetto che viene generato con le informazioni del segmento TCP, aggiungendovi indirizzo IP di sorgente e destinazione, lunghezza del datagram e numero di ordine della sequenza, utile in caso sia necessario frammentare.

### 4.2.2 RESTful Web services

REST, Reperesentational State Transfer, definisce un insieme di principi architetturali software attraverso i quali è possibile progettare servizi web client-server per favorire l'interoperabilità tra diversi sistemi informatici che espongono risorse web, di computazione o di memorizzazione, locali o meno. Non prevede strettamente una tecnica di comunicazione ma è basato sullo stesse definizioni che lo stesso ideatore, Roy Fielding, stava formalizzando sullo sviluppo del protocollo HTTP/1.1, [13] adottandone i metodi principali: GET, POST, PUT, PATCH, DELETE, HEAD and OPTIONS.

REST ha un così grande impatto sul web che ha quasi totalmente rimpiazzato interfacce basate su SOAP e WSDL per la sua semplicità d'implementazione. Una delle caratteristiche chiave di un servizio web RESTful è l'uso esplicito dei metodi HTTP così come specificato nel documento RFC 2616. Ad esempio, la chiamata HTTP GET è definita come metodo "data-producing", cioè inteso per essere usato dal client per recuperare dati relativi ad una certa risorsa, che sia una pagina web o una query.

Il paradigma REST richiede agli sviluppatori di utilizzare i metodi HTTP in maniera consistente con le definizioni del protocollo. I principi base di design stabiliscono una relazione uno-a-uno tra le operazioni di "create, read, update e delete" (CRUD) e i metodi HTTP:

- POST: per creare una risorsa nel server;
- GET: per recuperare una risorsa;
- PUT: per cambiare lo stato di una risorsa o aggiornarla;
- DELETE: per rimuovere una risorsa.

Definendo questi vincoli alla realizzazione di applicazioni web, si è iniziato ad uniformare le interfacce per richiedere servizi web rendendoli come risorse remote identificate univocamente attraverso URI (Universal Resource Identifier). Questo identificatore contiene pertanto parametri o una loro serie che definiscono i criteri di ricerca per recuperare un insieme di risorse corrispondenti: errore comune è utilizzare questi elementi

per creare una nuova risorsa. Scelta infelice in quanto comporta un cambio di stato operativo, costituendo in effetti collaterali nel server, rendendo inconsistente la richiesta.

```
$ curl \
http://localhost:48081/api/v1/deviceprofile/uploadfile \
  -F "file=@HMI.profile.yaml"
```

**Listing 4:** Esempio caricamento profilo attraverso interfaccia REST

Il metodo POST evidenziato nell'esempio è riferito al caricamento di un profilo nella piattaforma EdgeX Foundry ed è consistente: infatti il payload è contenuto nel corpo della richiesta. Il ricevente può essere così processata in base al contenuto del corpo subordinatamente alla risorsa indicata nell'URI. Il client così associa la riuscita dell'esecuzione della richiesta con una relazione generatrice di un nuovo URI, infatti Edgex restituisce un identificativo del profilo caricato.

Queste risorse sono disaccoppiate dalla loro rappresentazione consentendo così che il contenuto possa essere di qualsiasi formato rendendolo riconoscibile attraverso dei metadati collegati, rendendo le risorse autodescriventi. Per questo le interazioni con le risorse possono essere definite "stateless", ognuna indipendente dalle precedenti e successive, quindi né il client né il server conservano o hanno informazioni sullo stato reciproco. Ciò rende più semplice l'implementazione da entrambi i lati, riducendo la complessità dei componenti che interagiscono, la semantica dei connettori e permettendo un aggiustamento fine nell'ottimizzazione delle performance, rendendo il sistema facilmente scalabile senza l'uso di intermediari. Per processare la richiesta quindi non si richiede al server di recuperare alcun tipo di contesto che non sia contenuto nella richiesta stessa per generare la risposta.

Cluster di server che autobilanciano il carico, con capacità di recupero in caso di *fail* sono organizzati in una specifica topologia con gateway e server proxy che permette alle richieste di essere inoltrate da un server all'altro, distribuendole in caso di necessità con l'obiettivo di diminuire i tempi di risposta delle varie chiamate.

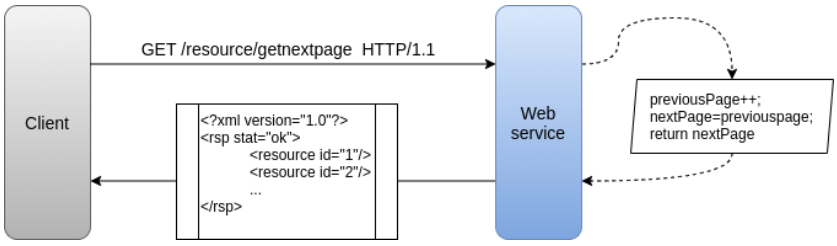



**Figura 4.3:** Esempio servizio *stateful*

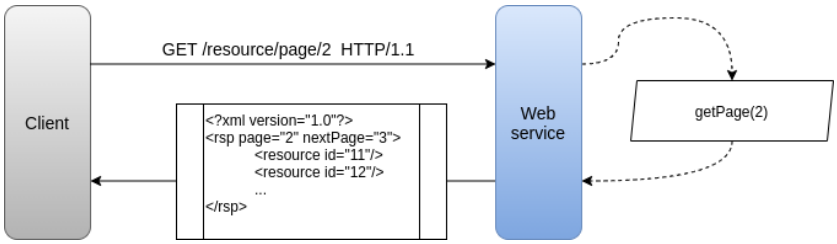



**Figura 4.4:** Esempio servizio *stateless*

Questo sistema accomoda il concetto di "cache", permette cioè di fornire risposte pre-calcolate senza richiedere ulteriori passaggi computazionali o di ricerca. Queste risorse devono però essere preventivamente dichiarate e seppur questa tecnica vede un largo utilizzo nel World Wide Web per il risparmio di banda, nel progetto è disabilitata in quanto il poter notare le differenze tra i vari dati è una peculiarità fondamentale alla corretta esecuzione del tutto.

Differentemente, i servizi "stateful" possono diventare complicati: la sincronizzazione richiesta richiede molte considerazioni per essere implementata e mantenuta in maniera efficiente, soprattutto in ambienti distribuiti. I componenti "state-less" sono più semplici da gestire dal lato server in quanto è responsabilità del client indicare la risorsa cercata in maniera formale. Questa tipologia di interazione può essere suddivisa in due insiemi di responsabilità ad alto livello di separazione, chiarificano come questi servizi possano essere mantenuti.

- **Server**
  - Genera risposte che includono link ad altre risorse, permetten-

do alle applicazioni si navigare tra risorse collegate. Analogamente, se la richiesta è riferita ad una risorsa genitore, una tipica risposta RESTful può includere un collegamento alla risorsa subordinata, così da rimanere logicamente vicina.

– Genera risposte che indicano se possono essere inserite in cache per ridurre il numero di richieste e la duplicazione di risorse. Viene implementato includendo nell'header della risposta i valori `Cache-Control` e `Last-Modified`.

- **Client application**

  – Legge nella riposta se viene specificato `Cache-Control` per indicare se copiare in memoria o meno il risultato chiesto. Il client inoltre utilizza `Last-Modified` per richiedere di poter restituire una risorsa se è stata modificata successivamente alla data in archivio (conditional GET), al quale viene risposto con un codice 304 (non modificato) o con la risorsa aggiornata.

  – Invia una richiesta completa come descritto precedentemente, senza porre troppe assunzioni sulle richieste precedenti.

Questa collaborazione tra l'applicazione client e il servizio è essenziale in un web service RESTful: ne migliora le performance salvaguardando la banda e minimizza gli stati lato server.

Dal punto di vista del client, la logica con cui un URI indirizza una risorsa, determina quanto sia intuitivo l'uso di un servizio, anticipa inoltre se tale servizio sarà usato come lo sviluppatore intende. Gli URI devono pertanto essere facili da indovinare cioè prevedibile anche per i programmatori lato client, risparmiando tempo e grattacapi, quindi aumentandone l'appetibilità.

Un modo per raggiungere questo scopo è definire una struttura gerarchica "a cartelle", indirizzabile quindi granularmente con un singolo percorso ad albero. Per questo è prassi seguire queste linee guida:

- nascondere la tecnologia di scripting lato server dalle estensioni dei file (`.jsp`, `.php`, `.asp`, ecc...), garantendo così anche la portabilità del sistema;
- indirizzi URI scritti in minuscolo;
- sostituire gli spazi con caratteri "URL friendly" come underscore o trattini;
- evitare stringhe di query;
- fornire una pagina di default o una risorsa generica in caso la richiesta sia incompleta o errata, quindi non generare un codice 404.

Viene invece richiesto che gli URI siano statici, in modo che se la risorsa stessa o l'implementazione del servizio cambiano, il percorso rimane lo stesso. Deve però essere mantenuta l'indipendenza da come la risorsa è salvata nel server da come appare nell'URI.

La rappresentazione di una risorsa riflette lo stato corrente della stessa con i suoi attributi, al momento in cui è richiesta; come una fotografia. Questa rappresentazione può essere semplice come un record in un database che consiste in un collegamento tra nomi di colonne e tag XML contenenti i valori delle righe. Oppure se il sistema ha un *data model*, in accordo alla definizione di rappresentazione di risorsa, è una copia degli attributi di una variabile nel data model. Ed è questo che un servizio REST deve restituire.

### 4.2.3 Modbus®

Protocollo di comunicazione dati, è nato nel 1979 per il monitoraggio digitale dei macchinari industriali con i controller logici programmabili (PLC) della stessa casa produttrice: Modicon, oggi Schneider Electric. Posizionato al livello applicazione del modello OSI, nella versione seriale è passato dalla comunicazione EIA/TIA RS-232 alla RS-485 per usufruire della maggiore distanza di comunicazione e affidabilità in ambienti elettricamente rumorosi, ampliando così l'adozione anche alle industrie manifatturiere. È stato

rilasciato nel 2004 alla Modbus Organization, associazione no profit di produttori ed utenti di dispositivi che lo implementano, ha l'incarico di pubblicare gratuitamente le specifiche, promuovere l'adozione del protocollo e mantenere lo sviluppo attivo [16].

La comunicazione è di tipo request/reply è orientata al controllo e la programmazione remota di apparecchiature compatibili. Nello specifico si basa sulla lettura e scrittura di registri che prendono il nome a seconda della funzione offerta e sono indirizzati con un numero 16bit (65.535 posizioni):

- *Holding Registers*, posizioni di memoria da 16bit che possono essere scritte e lette;
- *Input Registers*, posizioni di memoria da 16bit che possono essere solo lette;
- *Coils*, posizioni di memoria booleana (1bit) che possono essere scritte e lette;
- *Discrete Inputs*, posizioni di memoria booleana (1bit) che possono essere solo lette.

Le operazioni sono espresse con un numero nell'intervallo 1-127 ed oltre alla lettura e scrittura descrivono appositi registri per diagnosi e preferenze, lasciando alcuni intervalli liberi per funzioni definite dai produttori. In una sola richiesta di lettura è possibile leggere più registri, in particolare è possibile impostare un dispositivo per usare una coppia di indirizzi contigui da 16bit per utilizzarli assieme ed avere numerazioni con precisione a 32bit.

La velocità di trasmissione e affidabilità delle moderne reti anche su lunghe distanze (le specifiche per RS-485 raccomandano distanze inferiori ai 12m), ha portato ad ideare una variante che permettesse l'incapsulamento su TCP, replicando in toto il comportamento fin qui descritto. Questo permette l'instradamento nelle moderne reti Ethernet (IEEE 802.3) ed indirizzare i vari dispositivi con IP. Viene quindi definito ed implementato un

pacchetto (ADU) che comprende un header specifico (MBAP), le funzioni, il payload stesso nonché un elemento per il controllo degli errori. Si vengono così a creare due interlocutori definiti client e server che ricalcando quelli nel protocollo classico, si prevede che le richieste operative siano effettuate dal primo, quindi evase dal secondo. Lo scambio è effettuato in quattro fasi:

1. **Request**, viene prima effettuata la richiesta dal client che inizia quindi la transazione (identidicata da un numero su 2 byte);

2. **Indication**, è lo stato del server quando interpreta la richiesta appena ricevuta;

3. **Response**, si invia poi il messaggio stesso che prende questo nome;

4. **Confirmation**, il client quindi risponde con una conferma di ricezione e chiude la transazione.

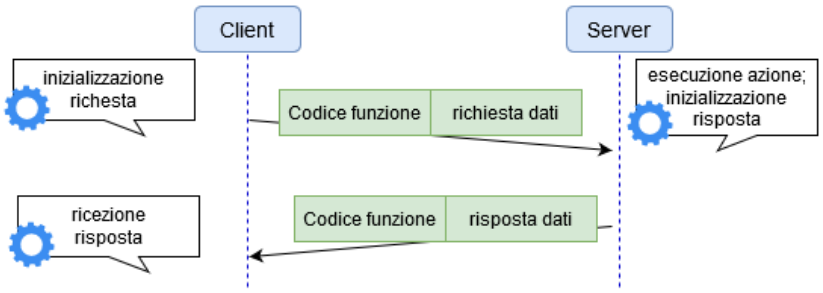



**Figura 4.5:** Schema conversazione su Modbus

Request e Response sono corredate di un codice specifico per la funzione richiesta, che nella restituzione può cambiare a seconda dell'esito. Questi codici comprendono funzioni di accesso ai registri (in lettura o scrittura) e di diagnostica.

#### 4.2.3.1 Conversione dati

Nei casi in cui la risorsa di un dispositivo usa un tipo di dato intero ma in un registro a scala "float" si rischia di perdere in precisione nel processo di trasformazione del dato.

Ad esempio un dispositivo Modbus che detiene valori di temperatura ed umidità in un `INT16` con una scala centesimale, se la temperatura è di 26,53, il valore letto è 2653, che trasformato diventa 26. Per evitare questo la risorsa del dispositivo deve specificare l'attributo opzionale `rawType` nel profilo, definendo così una lettura binaria sul dispositivo e un `value type` che indichi il dato atteso.

Quando viene specificato questo attributo il device service esegue il comando di lettura trattandola come dati binari, che vengono quindi letti come `INT16` e convertiti (cast) in un valore `FLOAT32`. In caso di scrittura, avviene il processo inverso, in cui un valore `FLOAT32` viene prima convertito in intero, poi in base 2 (binario) e solo dopo eseguita la scrittura.

La conversione può quindi avvenire da un `rawType` di tipo `INT16/UNIT16` a un `value Type` di tipo `FLOAT32/FLOAT64`

## 4.3 Rilevamento delle prestazioni

La soluzione sviluppata è stata messa alla prova per verificare sperimentalmente le prestazioni sia in condizioni normali che in *critical behaviour*. Condizioni calmierate su quelle presentate nella tesi del dott. ing. L. Patera in quanto dimostrate realistiche sia nella qualità dei valori prodotti che nella quantità. In particolare sono stati effettuati test a carico moderato da 600 letture al minuto ed a carico doppio, cioè 1200 letture al minuto sul macchinario simulato.

Ci si è inoltre spinti a moltiplicare il numero di richieste nel south-side aumentando i dispositivi registrati su EdgeX Foundry a step di 10 fino a 30, con 4 richieste al secondo, ottenendo 2400, 4800 e 7200 richieste al minuto, osservandone il comportamento verso la saturazione delle risorse. Ogni test ha previsto oltre alle richieste di lettura programmate anche richieste

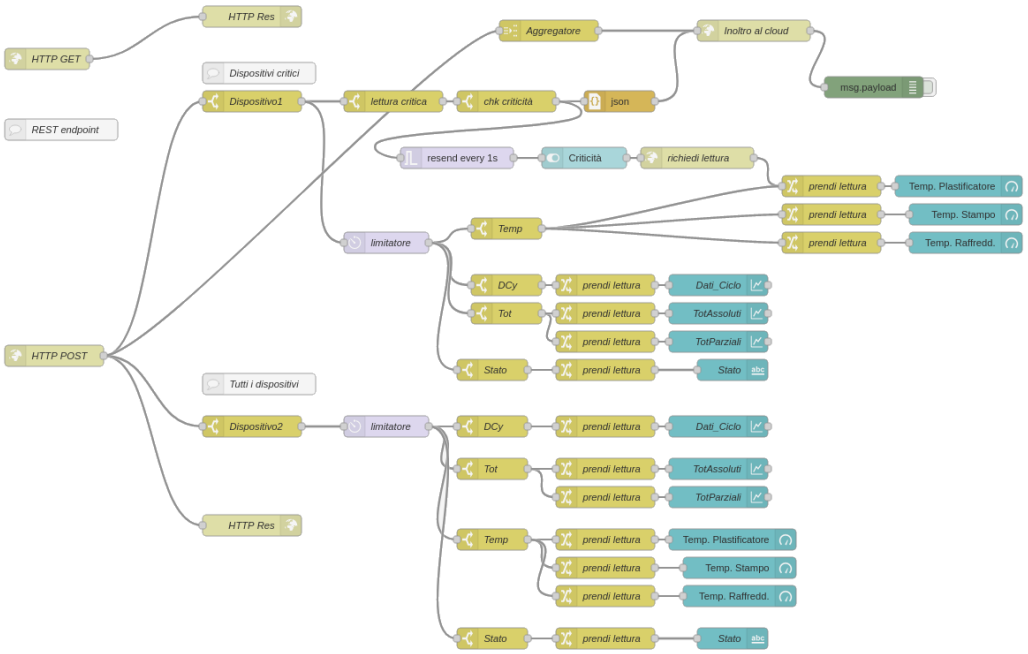



**Figura 4.6:** Dettaglio gestione del flusso dati in Node-RED

estemporanee su alcuni dei dispositivi, effettuate attraverso i comandi pubblicati dal servizio Core Command e richiamati dall'applicativo sviluppato su Node-RED, il cui dettaglio è visibile nello schema in Figura 4.6.

Si sono quindi controllati i tempi delle conversazioni Modbus-TCP e quelle verso l'endpoint remoto nel north-side effettuate tramite HTTP con metodo POST e corpo JSON.

A titolo di confronto è stato monitorato anche l'uso delle risorse prima di avviare la piattaforma EdgeX Foundry con l'applicativo in Node-RED e dopo avviato, ma comunque prima di far iniziare le richieste di lettura sui dispositivi Modbus.

Infine è stato effettuato un monitoraggio granulare per osservare i picchi di utilizzo delle risorse dei vari componenti: in ambiente Linux, il kernel può rilevare lo stato di occupazione della memoria centrale che se non è valutato più sufficiente invia un'eccezione OOME (Out Of Memory Exception) per poi iniziare ad arrestare i programmi in esecuzione non prioritari. A questo comportamento sono soggetti anche i container di Edgex, fino ad

arrivare ad abbattere Docker stesso.

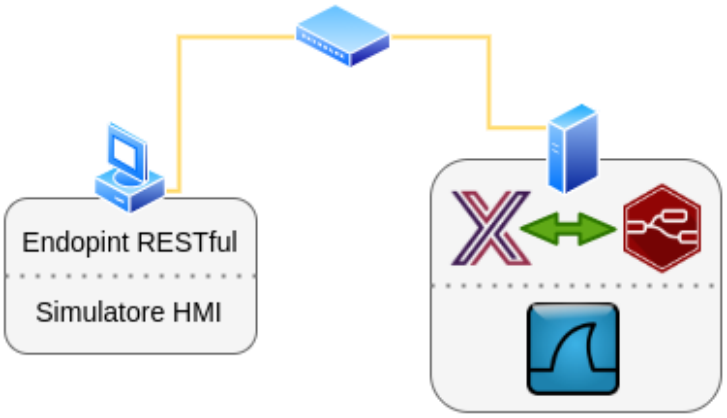



**Figura 4.7:** Schema alto livello dell'ambiente di test

### 4.3.1 Virtualizzazione

Con questo termine si intende genericamente la separazione di una risorsa o richiesta per un servizio dalla sottostante esecuzione fisica di tale servizio. Ad esempio, con la memoria virtuale, un computer acquisisce più RAM di quella fisicamente installata, ma tenendo in background un servizio che in maniera trasparente ne sposta i record con quella reale. In modo analogo le tecniche di virtualizzazione forniscono un livello di astrazione tra hardware computazionale, archivio e networking, e le applicazioni eseguite.

Questo permette una oculata gestione delle risorse a livello amministrativo, ripartendole virtualmente tra i vari sistemi che possono essere ospitati. Inoltre agevola lo sviluppo ed il testing standardizzato di applicazioni permettendo il riutilizzo di macchine virtuali preconfigurate. La continuità aziendale è assicurata grazie ad un "disaster recovery" agevolato dal poter contenere tutto il sistema su cui si lavora in un unico file che può essere replicato e ripristinato in qualsiasi momento.

Pioniere della virtualizzazione a livello hardware fu IBM negli anni '70 sui mainframe proprietari e solo successivamente applicata ai sistemi Unix/RISC, ma ben presto diventò software-based con l'aumentare delle ca-

pacità computazionali e la difficoltà di sviluppare hardware compatibile. Quest'ultima metodologia permette due differenti approcci:

- **Hosted**, permette di eseguire le macchine virtuali come applicazioni, sopra sistemi operativi standard, supportando la maggior parte delle configurazioni hardware;
- **Hypervisor**, si interpone tra l'hardware per garantire l'accesso diretto alle risorse, è quindi più efficiente e più scalabile ma più difficili da implementare.

Gerald J. Popek and Robert P. Goldberg nel 1974 hanno sintetizzato come siano 3 le caratteristiche principali di una macchina virtualizzata [14] :

- Principio d'equivalenza, un programma eseguito all'interno di una VM deve esibire un comportamento essenzialmente identico a quello dimostrato su una macchina non virtualizzata.
- Principio di controllo delle risorse, il Virual Machine Monitor deve essere in completo controllo delle risorse virtualizzate.
- Principio d'efficienza, una frazione statisticamente predominante di istruzioni macchina deve essere eseguita senza l'intervento del VMM.

Le macchine virtuali citate sono eseguite attraverso VMware Workstation Player, applicazione per virtualizzazione hosted che sfrutta la virtualizzazione offerta dai processori Intel (VT-x) e AMD (AMD-V) per l'accesso privilegiato alle risorse computazionali, di memoria e I/O dedicate alla virtualizzazione x86. Permette inoltre l'integrazione grafica e la condivisione di file, appunti e dispositivi di input come tastiera e mouse, rendendo efficiente e veloce lo sviluppo di applicazioni.

#### 4.3.1.1 Memoria swap

Nei sistemi GNU/Linux, come nel caso di studio, la RAM viene divisa in più parti chiamate "pagine" e viene utilizzata una parte della memoria di

massa per memorizzare dati temporanei che non riescono ad essere contenuti nella memoria centrale, detta di *swap* (dall'inglesismo che sta per "scambiare"). L'insieme formato dall'area di swap e dalla memoria centrale costituisce la memoria virtuale, meccanismo che permette di poter considerare come se la memoria primaria fosse illimitata.

Quando il kernel ha bisogno di memoria centrale da dedicare ad un'attività, parte del contenuto di quest'iltima viene "scaricato" nello swap (*swap out*). Quest'azione è effettuata dal *virtual memory manager* del kernel, che decide ad esempio di scaricare quella meno frequentemente acceduta. Quando invece le informazioni alle quali un'attività fa riferimento sono nello swap, queste devono essere ricaricate in memoria centrale (*swap in*). Questo meccanismo può aiutare a migliorare le prestazioni di computer con una quantità di memoria primaria limitata, anche se la velocità di accesso è di circa $10^6$ volte più lenta.

Lo swap può essere di tipo statico nel caso la dimensione sia fissata alla creazione, oppure dinamico nel caso in cui possa incrementare o diminuire la dimensione in modo dipendente dal suo utilizzo da parte del sistema; nel progetto è sempre statica, assegnata automaticamente dal software di virtualizzazione.

### 4.3.2 Strumenti di misura

Come nel sangue umano, analizzare i vari globuli che lo compongono ci permettono di stimare la salute di un individuo, analizzare i pacchetti che percorrono una rete ci permette di verificare la congestione della stessa, così da risalirne alle cause e poter intervenire con precisione. Analogamente come un elettrocardiogramma sotto sforzo ci permette di valutare la prestanza fisica, con il monitoraggio sotto stress delle risorse usate da un applicativo si riesce a verificarne l'efficienza.

#### 4.3.2.1 Wireshark

Espressamente ideato come analizzatore di protocolli di rete, anche questo strumento è un software open source sviluppato sin dal 1998 è ora anno-

verato tra i più famosi e completi disponibili [15]. Nei sistemi basati su Unix-like permette nativamente (con *libpcap*) la cattura di tutti i dati provenienti da qualsiasi sorgente collegata (modalità promiscua) alla stessa rete del computer che ospita Wireshark, indipendentemente dall'indirizzo fisico.

Riesce a catturare così i pacchetti in maniera trasparente senza quindi influire sul traffico stesso (packet sniffing), rivelandosi un ottimo analizzatore imparziale. La configurazione del filtro di cattura permette di evitare la registrazione di pacchetti che non interessano ai fini dell'analisi che si vuole effettuare, per questo è stato configurato un filtro che lasciasse acquisire solo i dati proveniente ed uscenti dalla macchina virtuale principale.

Il tool è stato installato ed avviato sul computer ospitante la VM principale lasciandolo in ascolto sulla rete impostando un filtro di cattura aperto ai soli pacchetti TCP di rilevo: quelli scambiati tra macchina virtuale principale e gli altri livelli. La cattura permette quindi di discernere le conversazioni in base agli IP e la direzione dei pacchetti, in quanto Wireshark è in grado di comprenderne la struttura (incapsulamento), catturando così lo scambio di pacchetti che avviene all'interno di esse, acquisendone tempi e dimensioni.

#### 4.3.2.2 htop 3 e risorse

Nato come miglioramento allo strumento di sistema di monitoraggio e gestione dei processi *top*, permette il controllo interattivo dei processi attivi nel sistema e dell'impegno richiesto dalle risorse, sia a livello generale che discreto visualizzando il dettaglio del solo processo. Tutto questo direttamente nel terminal di sistema così da avere un impatto trascurabile nell'impegno delle risorse dello stesso sistema.

Il tool è stato eseguito all'interno della macchina virtuale stessa in quanto VMware Player, nell'eseguire la macchina virtuale pre-alloca l'occupazione di memoria, impedendo una rilevazione granulare, inoltre l'occupazione della CPU generale sarebbe falsificata dagli altri programmi nella macchina ospitante (Wireshark).

Dall'architettura di EdgeX Foundry si evince quanto si faccia affidamento sulla memoria RAM per limitare la latenza nelle risposte e solo marginale sia l'uso della memoria di massa; mimando il comportamento di una cache. Per questo ne è stata osservata l'impronta di utilizzo assieme all'uso di quella virtuale in condizioni più stringenti (2GB rispetto ai 4GB target). **Requisiti minimi** (architettura x86 e ARM64):

- Memoria RAM: 1GB (dedicati);
- Spazio: 3GB, raccomandati 32GB totali, che possono variare a seconda della persistenza voluta delle letture;
- Sistema operativo:
  - Microsoft Windows: versione 7 e più recenti;
  - Ubuntu Desktop/Server: da versione 14 a 20;
  - Ubuntu Core: da versione 16 a 18;
  - Apple MacOS X: versione 10 e più recenti.

Come accennato precedentemente, è stata eseguita anche una valutazione focalizzata sui container eseguiti su Docker attraverso l'uso del comando CLI `docker stats` che visualizza uno stream in tempo reale del relativo utilizzo di risorse. I risultati riportati nei grafici 4.21 e 4.20 mostrano le percentuali raggiunte durante l'esecuzione dei vari test, in riferimento alle risorse disponibili sulla macchina virtuale per i vari container ospitanti i microservizi.

### 4.3.3 Throughput

Si sono misurati quanti bit vengono scambiati nel caso di massimo impegno e quindi quanto occupano la banda di trasmissione disponibile di 100Mbit/s dal secondo livello nelle direzioni verso:

- **north-side**: conversazione dalla VM all'endpoint REST: 10 pacchetti TCP per un totale trasmesso e ricevuto per ogni conversazione di

2214B con una velocità media di 0,54Mb/s, che senza funzione di aggregazione ammontano ad una media di 1,22Mb/s;

- **south-side**: velocità media letture su Modbus-TCP di 1,51Mb/s con una media di 23 pacchetti TCP per richiesta scambiati.

L'occupazione massima dei pacchetti sulla banda è circa il 2% sulla linea di testing da 100Mb/s, sia nel south-side che in export verso il north-side. La maggiore velocità delle letture sul dispositivo Modbus-TCP si giustifica sapendo che sono divisi in più conversazioni con una minore dimensione del payload pro-capite, il tempo di conversazione medio diminuisce. In dettaglio le letture dei sensori nel south-side, nel caso di studio comprendono:

- la richiesta per le temperature occupa 1264B in 20 pacchetti, comprende la lettura dei valori corrispondenti a:
  - Temperatura Plastificatore;
  - Temperatura Stampo;
  - Temperatura Raffreddamenti;
- per avere i valori dei totalizzatori si occupano 1012B in 16 pacchetti e comprendono:
  - Totalizzatore Assoluto;
  - Totalizzatore Parziale;
- infine le letture più "leggere" sono quelle che riportano un solo valore occupando 760B in 12 pacchetti come:
  - Dati Ciclo;
  - Stato Macchina.

La velocità ed il numero di byte passanti in totale rimangono ben al disotto della capacità della rete interna, bensì il trasporto verso l'esterno del north-side risulta impegnativo occupando la stessa percentuale nella banda

disponibile delle letture Modbus ma con un ordine di pacchetti 100 volte inferiore (decine di migliaia di letture Modbus contro alcune centinaia in export verso il north-side).

### 4.3.4 Tempi di risposta e latenza

La valutazione dei tempi rilevati conferma come l'utilizzo di un sistema multicore permetta di ospitare più thread che rispondono ai vari micro-servizi contemporaneamente in esecuzione, mantenendo basso il tempo di comunicazione interno ad EdgeX Foundry senza gravare sui tempi di risposta nell'inoltro dei dati. Come è possibile verificare dai grafici, sia nel south-side (Figura 4.8) che nell'esportazione verso north-side (Figura 4.11) non mostrano tendenze di andamento importanti.

Infatti la media di risposta per effettuare dalla richiesta alla risposta con la lettura dai dispositivi Modbus-TCP è di 3,5ms e come mostrano i grafici 4.10 e 4.9 non sono correlati alla dimensione degli stessi. Infatti come anticipato, un dispositivo può fornire più letture in risposta ad un'unica richiesta, come nel caso in studio.

I grafici che offrono una visione panoramica considerano tutti i pacchetti TCP scambiati tra le parti, comprendendo quindi sincronizzazione e finalizzazione tipici del protocollo. Nelle visioni dettagliate invece sono riportate le conversazioni che hanno effettivamente trasportato le letture inoltrate dal south-side.

L'interpretazione dei risultati non evidenzia particolari rallentamenti nei casi di studio da 600 o 1200 richieste di letture al minuto, con uno o due dispositivi critici che inviano letture all'endpoint remoto.

Si può notare invece dai grafici nelle figure 4.15 e 4.16 come sia la tipologia del servizio verso cui viene richiesta un'interazione ad essere causa dei diversi tempi di risposta: Modbus si attesta molto adatto a letture piccole e veloci, le richieste con metodi HTTP si prestano invece all'invio di dati di dimensioni maggiori. I test da 2400, 4800 e 7200 richieste al minuto sul canale Modbus mostrano una tendenza alla saturazione delle risorse per la gestione del south-side (Figura 4.17). Mentre l'elaborazione richiesta

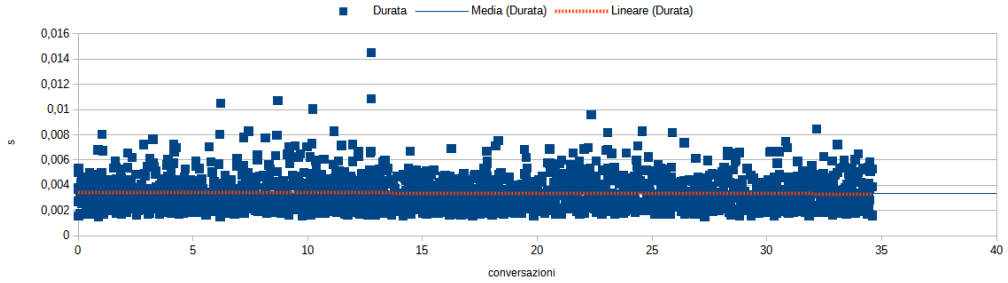



**Figura 4.8:** Panoramica tempo di lettura Modbus-TCP con 1200 richieste al minuto

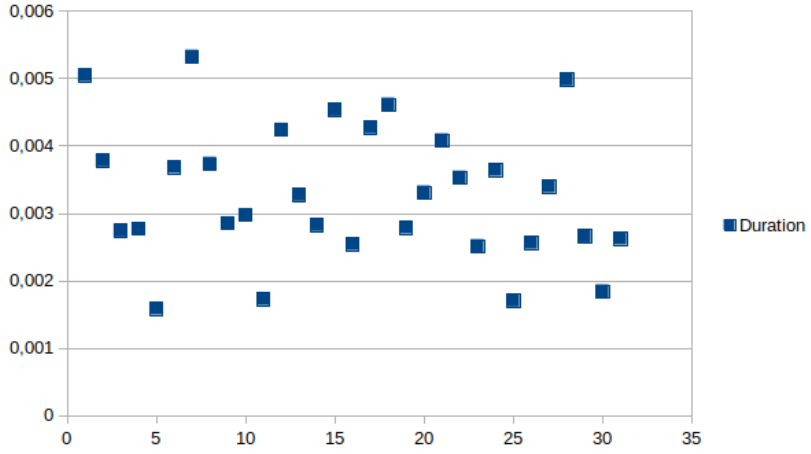



**Figura 4.9:** Dettaglio tempi invio pacchetti Modbus

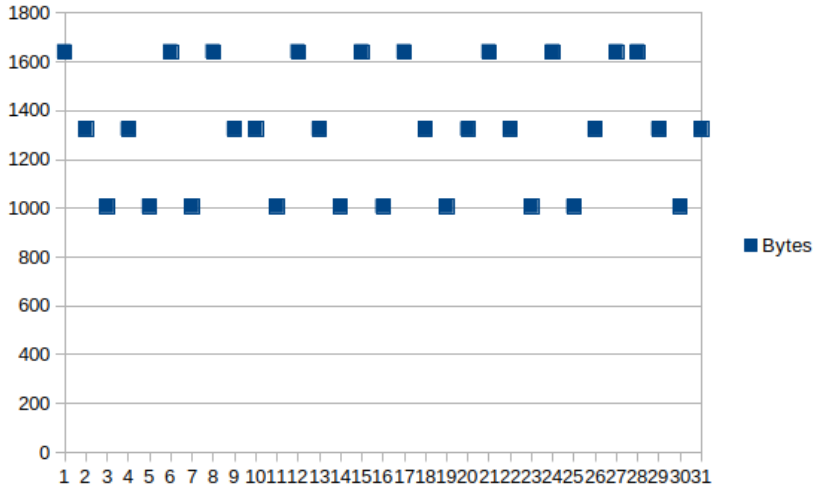



**Figura 4.10:** Dettaglio dimensioni pacchetti Modbus

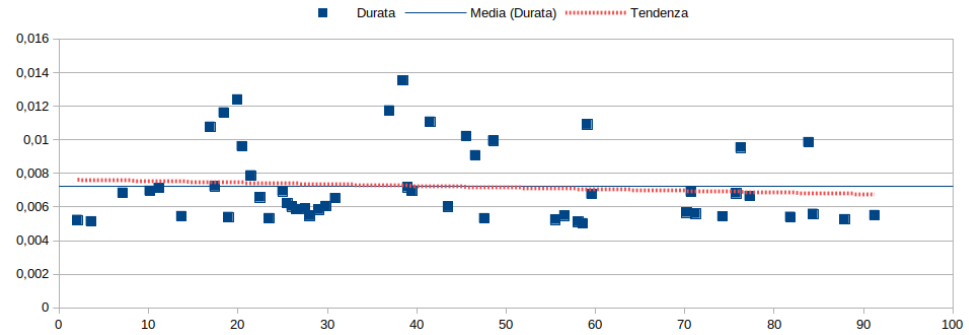



**Figura 4.11:** Dettaglio tempi invio pacchetti a servizio REST (600 richieste/minuto)

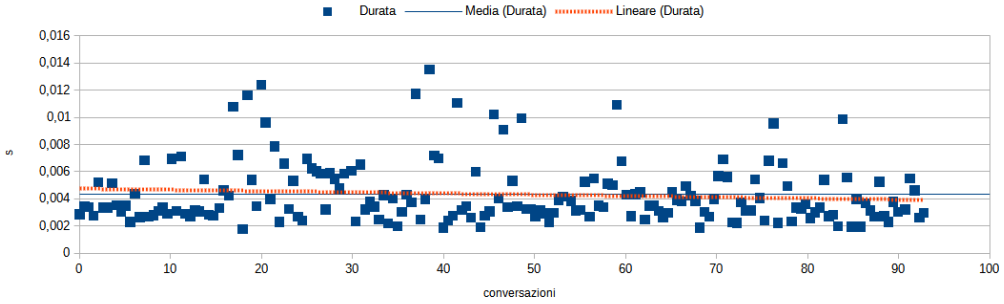



**Figura 4.12:** Panoramica scambio pacchetti servizio REST (600 richieste/minuto)

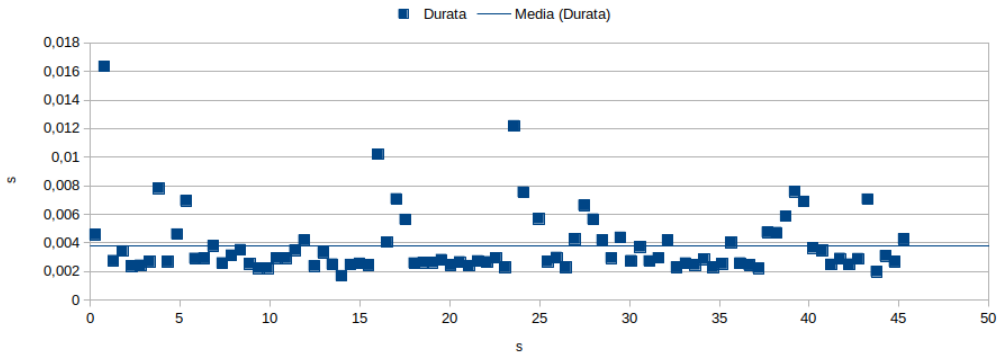



**Figura 4.13:** Panoramica conversazioni con endpoint REST, una sola lettura esportata

per i dati esportati verso il north-side, come visibile nel riassunto in Figura 4.18, sono del tutto paragonabili tra loro senza significativi cambiamenti nei tempi di invio.

Questo conferma come sia fondamentale gestire adeguatamente le conversazioni sia a causa della dimensione media sia del tempo impiegato per l'elaborazione, evitando di occupare eccessivamente il canale. In particolare le informazioni uscenti verso il north-side sono quelle più sensibili perché appunto è più difficile garantire reti efficienti e la banda di trasmissione, risorsa limitata, è condivisa.

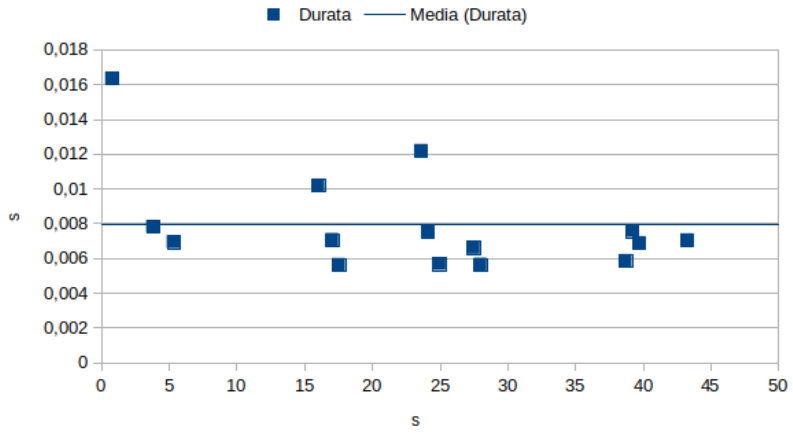



**Figura 4.14:** Dettaglio conversazioni con endpoint REST, una sola lettura esportata

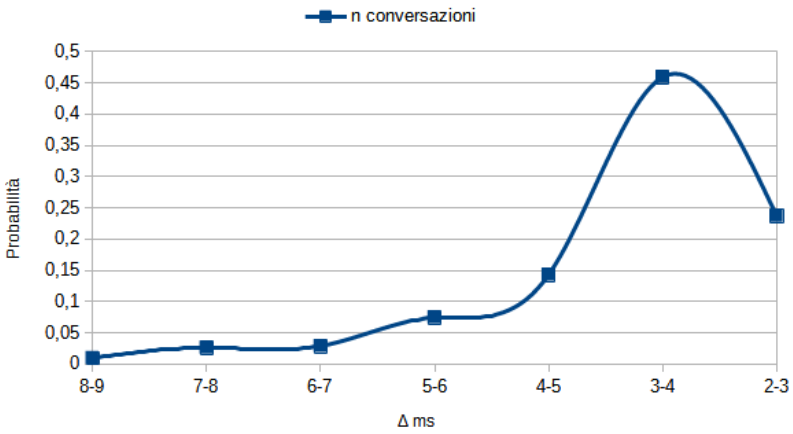



**Figura 4.15:** Probabilità durata conversazione Modbus

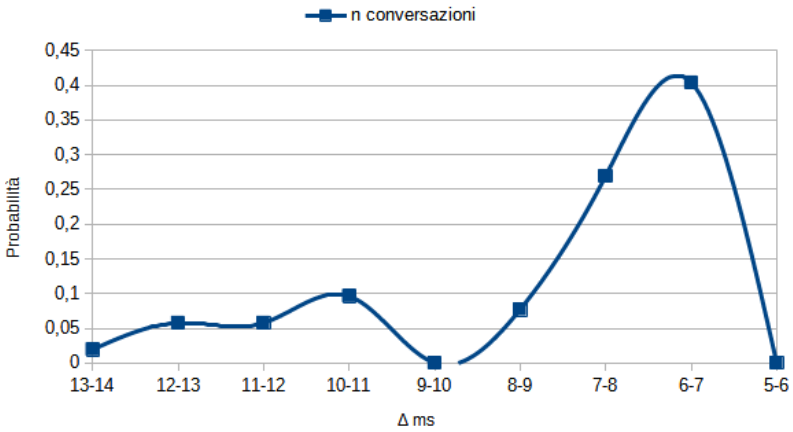



**Figura 4.16:** Probabilità durata conversazione con endpoint REST

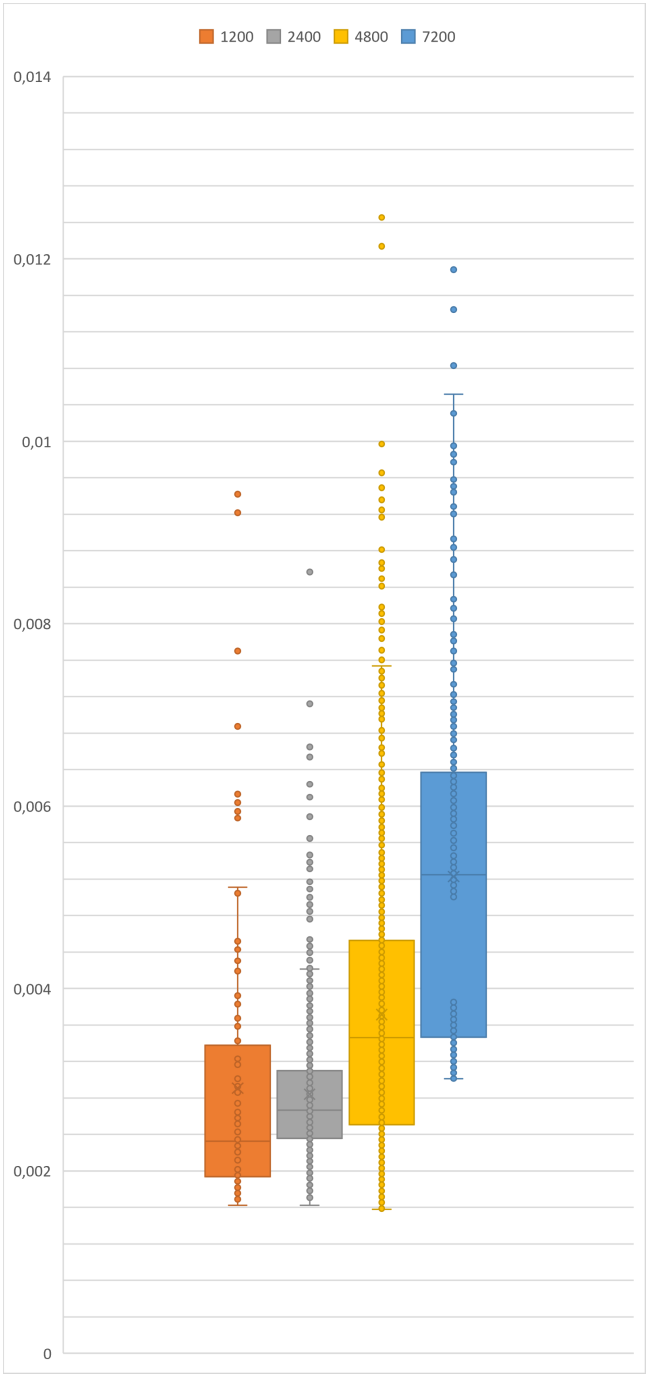



**Figura 4.17:** Riassunto andamento letture Modbus al minuto

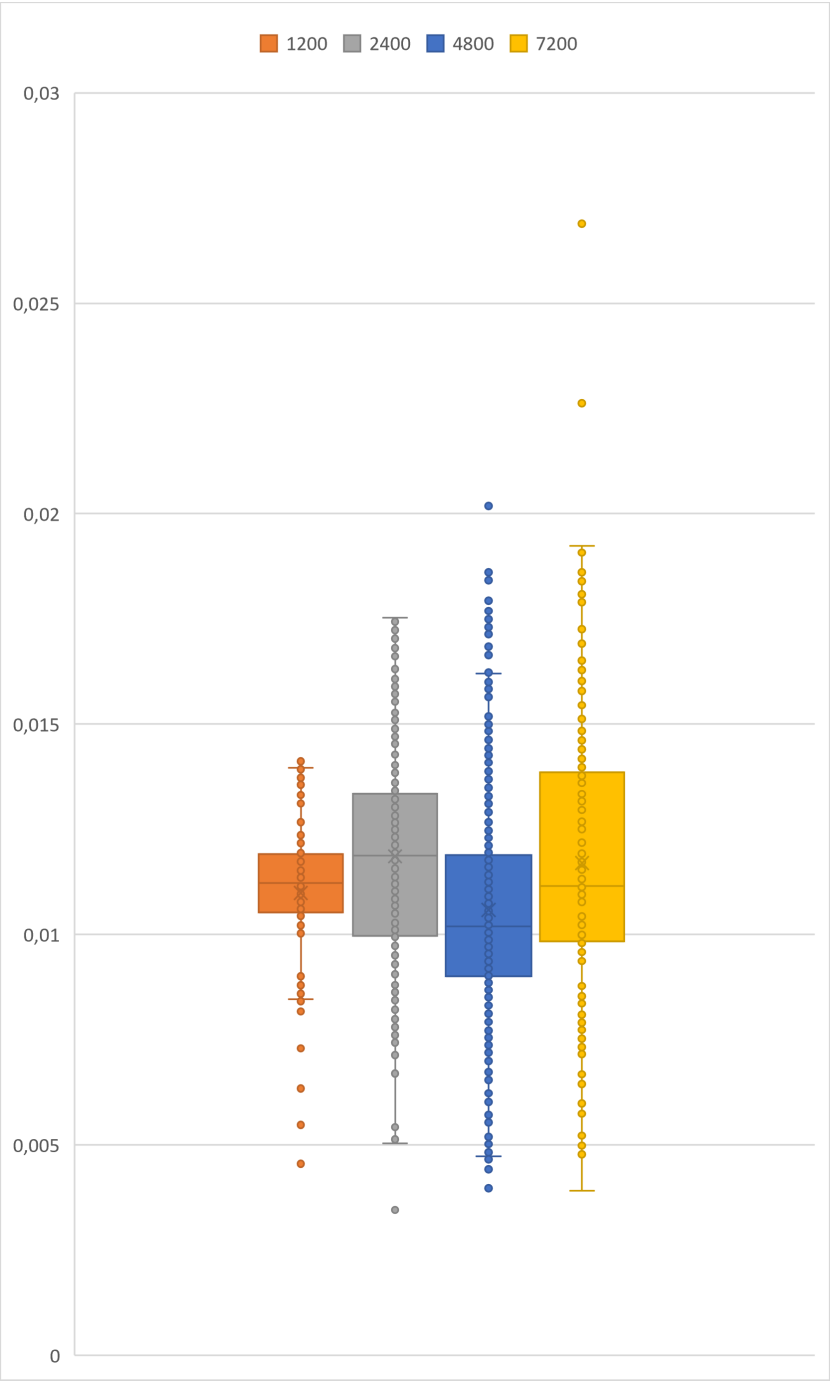



**Figura 4.18:** Riassunto andamento export REST

### 4.3.5 Occupazione risorse

La tabella 4.1 mostra i campionamenti effettuati allo stabilizzarsi dei valori sul quinto minuto, valore del calcolo automatico della media dell'occupazione della CPU da parte di *htop*, rispetto all'utilizzo delle risorse da parte di tutto il sistema. Il valore tCPU è invece da considerarsi in rapporto al numero di core disponibili: in questo caso un tempo di occupazione dei processi pari a 4,0 equivale ad un utilizzo del 100% della CPU. In considerazione di ciò, il valore "CPU Edgex" nel grafico 4.19 è riportato nella relativa percentuale per una migliore leggibilità.

In caso di sovraccarico e congestione della CPU questo valore può superare il valore equivalente al numero dei core indicando che i thread non riescono ad acquisire la risorsa processore, dovendo aspettare più del dovuto (rischio starvation). I valori in percentuale sono riferiti all'utilizzo della risorsa rispetto al totale delle risorse disponibili da parte del processo padre Docker che ospita i servizi della piattaforma EdgeX Foundry quando specificato in tabella. All'inizializzazione della macchina virtuale principale, VMware Workstation Player assegna in automatico 947MB di memoria swap.

Interessante è stato osservare come influisce l'aumento della RAM sul sistema, i risultati sono mostrati nei grafici 4.20 e 4.21, omettendo i valori della memoria virtuale in quanto non rilevante (uso complessivo inferiore ai 20MB in tutti i casi). Si può notare come l'uso della memoria primaria sia aumentato sin dall'avvio rispetto alla situazione con 2GB, eliminando di fatto l'uso della memoria swap ma senza aumenti sostanziali al seguito. In proporzione l'utilizzo della RAM è risultato marginale rispetto a quella necessaria al sistema operativo stesso solo fino a che le letture vengono correttamente consumate e quindi la pulizia automatica di EdgeX Foundry può liberare la memoria occupata da Redis.

Il monitoraggio granulare delle risorse dei servizi del *core* di Edgex permettono di paragonarne l'utilizzo con l'applicativo sviluppato in Node-RED, il quale si rivela essere il container Docker più esigente. Nonostante l'aumento della RAM, il carico computazionale è rimasto pressoché invariato,

| | *RAM (MB)* | *tCPU* | *occ. CPU (%)* | *Tasks* | *Threads* | *memoria swap (MB)* | *occ. memoria swap (%)* |
|---|---|---|---|---|---|---|---|
| idle, no Edgex | 609 | 0,09 | 0,1 | 93 | 163 | 0,268 | 0,03 |
| idle, con Edgex | 932 | 0,14 | 0,7 | 140 | 600 | 188 | 19,85 |
| 600 r/min | 944 | 0,5 | 5 | 140 | 606 | 226 | 23,86 |
| 1200 r/min | 983 | 0,83 | 8,1 | 140 | 609 | 243 | 25,66 |
| 2400 r/min | 1005 | 2,7 | 25 | 141 | 603 | 277 | 29,25 |
| 4800 r/min | 1030 | 3,4 | 30 | 141 | 604 | 304 | 32,10 |

**Tabella 4.1:** Utilizzo risorse in base al carico

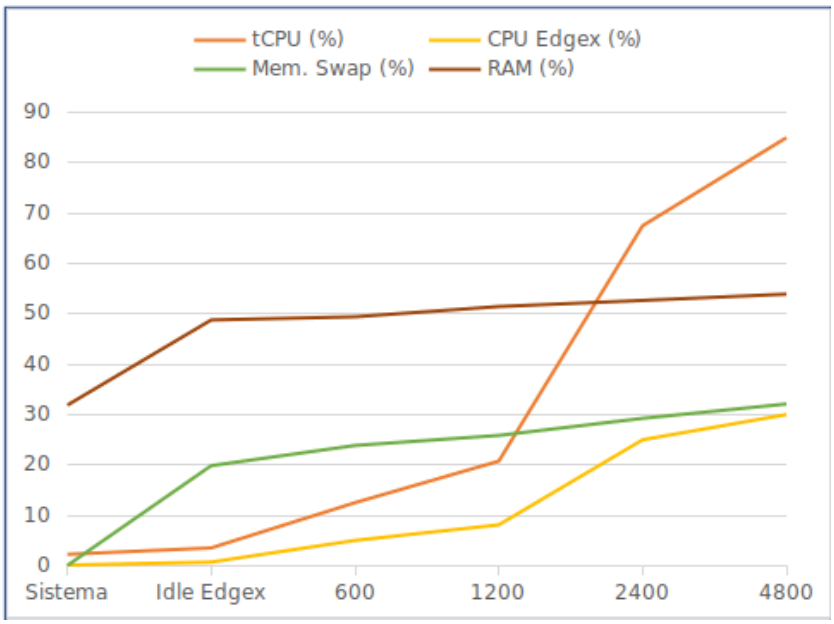
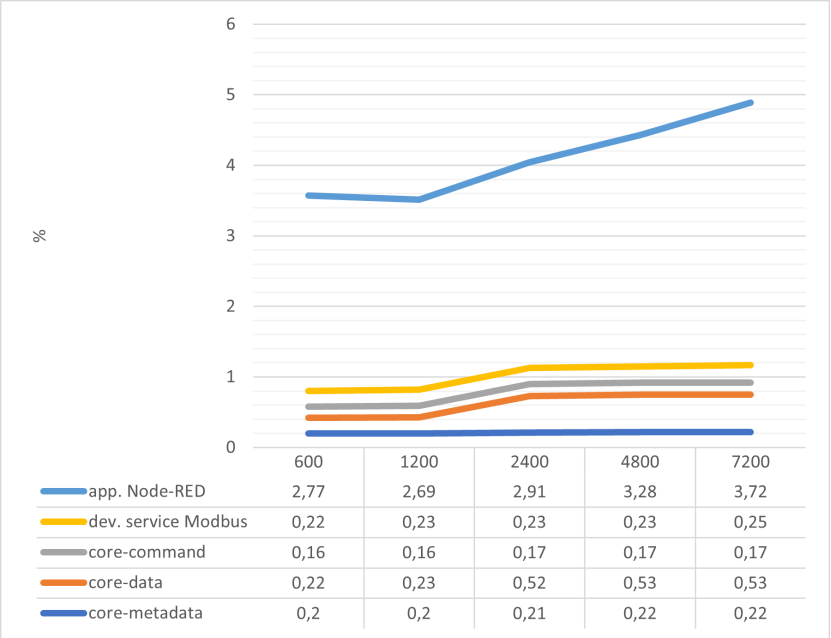


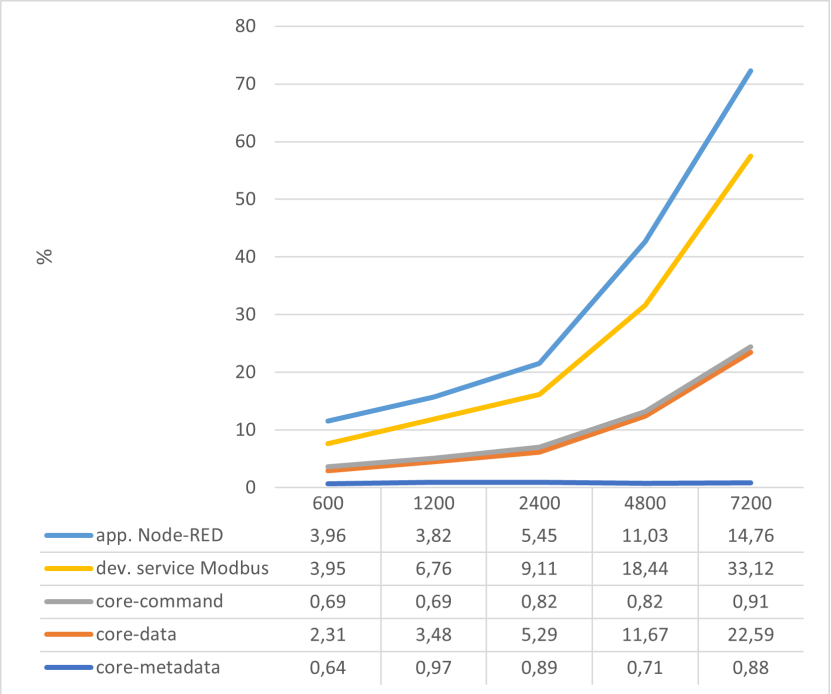

**Figura 4.19:** Riassunto grafico della Tabella 4.1

tendente ad essere più alleggerito, risultato verosimilmente correlato dal fatto che non vi è più necessità di recuperare dati dalla memoria virtuale e i thread non risentono quindi della lentezza della memoria di massa (I/O bound). In questo caso non sono stati monitorati i tempi di risposta nelle comunicazioni ma vista l'occupazione della CPU pressoché identica non ci si aspetta cambiamenti significativi.




| | 600 | 1200 | 2400 | 4800 | 7200 |
|---|---|---|---|---|---|
| app. Node-RED | 2,77 | 2,69 | 2,91 | 3,28 | 3,72 |
| dev. service Modbus | 0,22 | 0,23 | 0,23 | 0,23 | 0,25 |
| core-command | 0,16 | 0,16 | 0,17 | 0,17 | 0,17 |
| core-data | 0,22 | 0,23 | 0,52 | 0,53 | 0,53 |
| core-metadata | 0,2 | 0,2 | 0,21 | 0,22 | 0,22 |

**Figura 4.20:** Utilizzo RAM in base al carico (4GB RAM)




| | 600 | 1200 | 2400 | 4800 | 7200 |
|---|---|---|---|---|---|
| app. Node-RED | 3,96 | 3,82 | 5,45 | 11,03 | 14,76 |
| dev. service Modbus | 3,95 | 6,76 | 9,11 | 18,44 | 33,12 |
| core-command | 0,69 | 0,69 | 0,82 | 0,82 | 0,91 |
| core-data | 2,31 | 3,48 | 5,29 | 11,67 | 22,59 |
| core-metadata | 0,64 | 0,97 | 0,89 | 0,71 | 0,88 |

**Figura 4.21:** Utilizzo CPU in base al carico (4GB RAM)

## 4.4 Scalabilità

Con la versione Geneva di EdgeX Foundry i microservizi sono vincolati di default ad alla sola comunicazione interna (localhost), delegando la risoluzione degli host a Docker. Per distribuire la piattaforma su più macchine si è scelto di non incidere l'architettura di EdgeX Foundry distribuendo manualmente i vari microservizi, bensì si è delegato lo strumento di orchestrazione dei container integrato in Docker stesso, chiamato Swarm. Distribuire un sistema è un'operazione delicata che deve tener conto di alcune sfide comuni, quali dati ridondanti e sincronizzazione degli stessi tra le parti in gioco, che potrebbero portare ad inconsistenze. I principali problemi che possono incorrere sono relativi a:

- **caduta o pausa dei processi** a causa di difetti hardware o software, manutenzione non programmata o carichi eccessivi che ne bloccano le risposte temporaneamente;
- **ritardi nella rete**, ad esempio lo stack di protocolli TCP/IP non prevede un massimo di ritardo nelle comunicazioni che varia a seconda del carico nel network;
- **errata sincronizzazione degli orologi e ordine degli eventi** implica l'impossibilità dell'uso di timestamp per verificare l'età dei messaggi scambiati, fondamentale in caso contengano comandi.

### 4.4.1 Docker Swarm

A differenza di Compose che deve essere installato manualmente, Swarm è uno dei tool compresi nativamente in Docker, con il compito di agevolare lo sviluppo e la gestione di applicativi multi-servizio. Swarm è il relativo motore di clustering dei servizi definiti nello stesso file che usa Compose per gestirli. Il tool è in grado di replicare i servizi che vengono indicati grazie a SwarmKit, un progetto separato ma integrato in Docker dalla versione 1.12, e distribuirli in un insieme di nodi Docker che eseguiranno che li eseguiranno prendendo il nome di *task*.

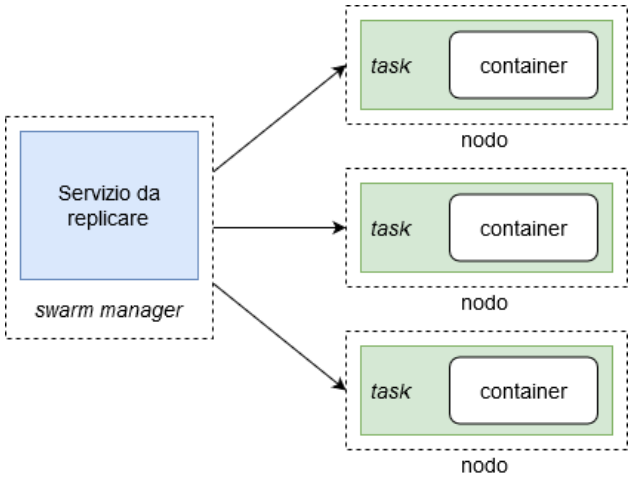



**Figura 4.22:** Schema Docker Swarm

Per "nodo" si intende una istanza Docker che partecipa al cluster mettendo a disposizione le proprie risorse, è quindi un host su cui è in esecuzione Docker in "swarm mode" e può essere essere di due tipologie:

- **Master**: obbligatorio per creare una rete swarm, gestisce il deploy dell'applicazione delegando gli altri nodi per l'esecuzione del servizio e monitora lo stato dei nodi del cluster. Più nodi possono ricoprire tale ruolo ma solo uno è il coordinatore, detto "swarm manager" (di default il primo istanziato) che ha la responsabilità di mantenere il cluster prossimo allo stato ottimale definito all'avvio (tramite CLI) in cui è specificato tra gli altri il numero di repliche, le risorse di archiviazione e network.
- **Worker**: esecutore passivo del servizio assegnato, viene preferito in quanto ha il solo compito di eseguire il task e la responsabilità di informare il nodo master coordinatore sul suo stato.

Un task è quindi un'istanza di un container a partire da un'immagine che prende il nome di "stack" quando sono molteplici e ad ognuna è assegnato ad un servizio. In questo modo lo swarm manager bilancia il carico in base alle richieste esterne in entrata e quelle interne tra i servizi, attraverso le entry DNS assegnate corrispondenti ai nomi dei servizi distribuiti e

tra le porte TCP interne automaticamente assegnate al deployment. Que-

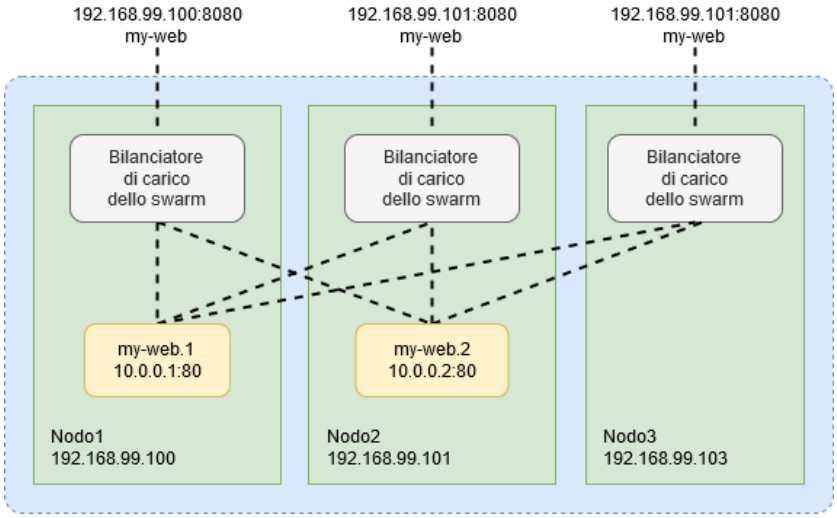



**Figura 4.23:** Schema Mesh Swarm e bilanciamento del carico

sto permette di instaurare una "routing mesh" che collega attraverso un *overlay network* (se non specificato ne viene creato uno di default chiamato `ingress` riportato in Figura 4.23 in azzurro) ogni nodo ad una porta, punto d'ingresso controllato per le richieste provenienti dall'esterno alle risorse pubblicate. Richieste che i bilanciatori di carico redirigono ai container attivi: questa comunicazione interna tra i vari container avviene invece attraverso un secondo network che prende il nome di `docker_gwbridge`.

### 4.4.2 Risultati

Per il test sono stati messi a disposizione dello swarm 3 nodi totali, il cui coordinatore risiede nella VM principale in cui EdgeX Foundry è stato sempre ospitato. La creazione di questa rete di nodi è avvenuta attraverso l'uso del file riassuntivo per Docker compose messo a disposizione dagli sviluppatori, risultato della modifica del file originale per permettere all'intera piattaforma di lavorare attraverso gli overlay. Il device service Edgex Foundry per Modbus sin dall'avvio è stato spostato in automatico su uno dei nodi esterni al coordinatore, liberando dal carico della generazione delle

richieste la VM principale. Il test ha poi evidenziato come la distribuzione del carico di lavoro tende a mantenere le risorse con un impegno moderato: risultando in un occupazione dimezzata sui carichi maggiori della CPU garantendo così alla memoria centrale la possibilità di poter effettuare più spesso le routine di pulizia e scrittura in memoria di massa dei dati inoltrati.

Questo alleggerimento dei carichi evidenzia come anche i tempi delle conversazioni su Modbus-TCP siano ridotti, addirittura migliori a 7200 richieste al minuto (si veda il grafico 4.24) rispetto ad un carico modesto, di 1200 richieste al minuto, ma non distribuito. Un secondo test di performance che mostra una panoramica dello swarm, effettuato con campionature distanti circa un minuto l'una dall'altra, è riportato nel grafico 4.28: mostra come l'applicativo sviluppato in Node-RED inneschi una redistribuzione più sensibile e, nonostante varie oscillazioni, sia confermato un carico analogo al precedente focalizzato sul nodo principale (Grafico 4.27).

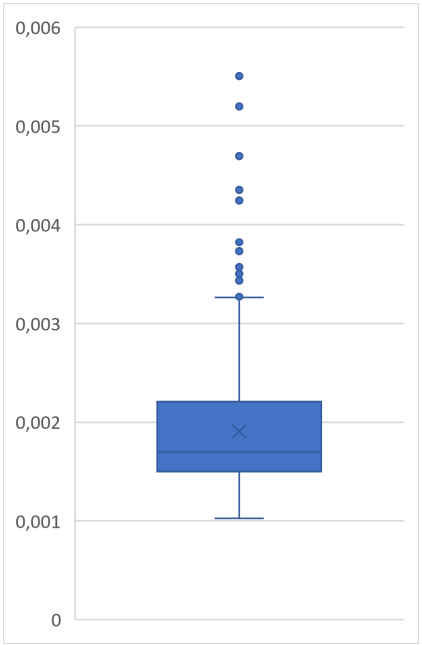



**Figura 4.24:** Tempi conversazioni Modbus-TCP con sistema distribuito

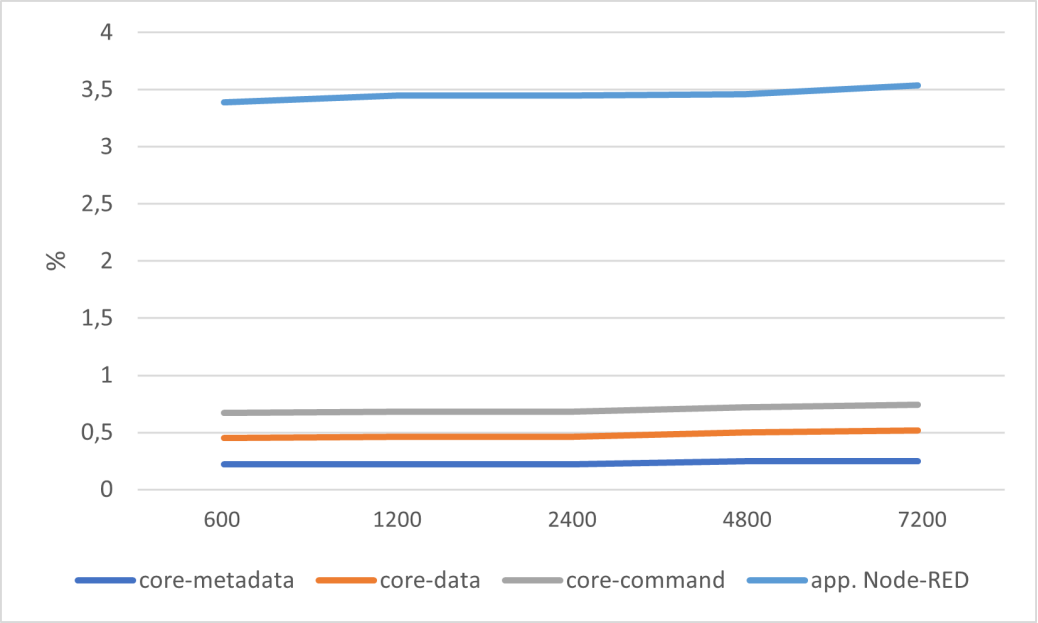



**Figura 4.25:** Andamento uso RAM nodo principale

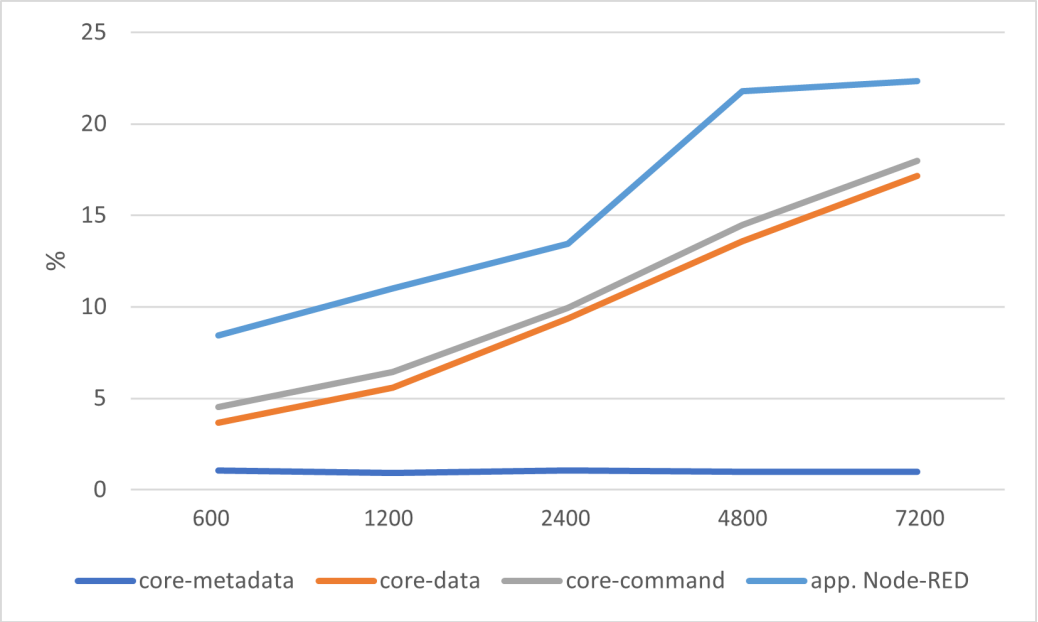



**Figura 4.26:** Andamento uso CPU nodo principale

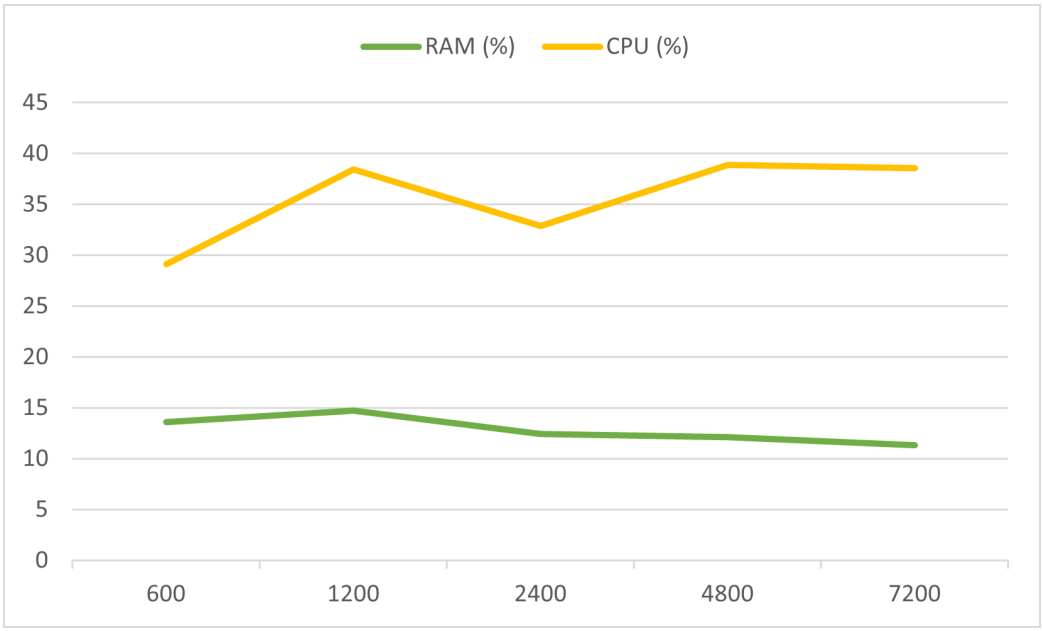



**Figura 4.27:** Andamento utilizzo generale risorse VM principale

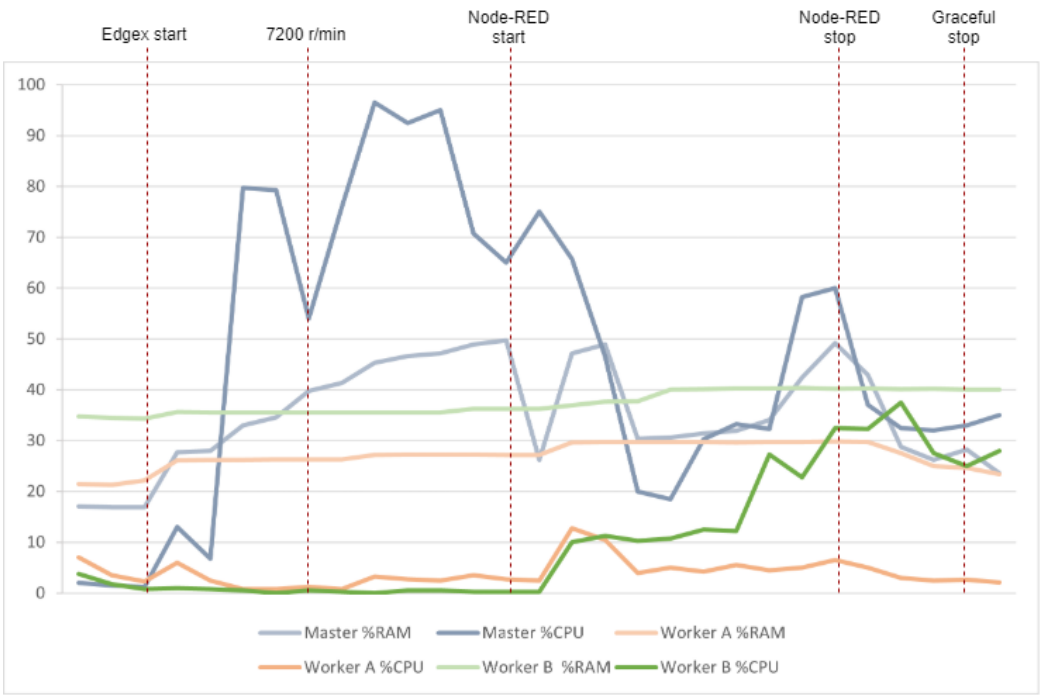



**Figura 4.28:** Andamento dello *swarm* rispetto agli eventi

Capitolo 5

# Conclusioni e sviluppi futuri

Lo studio effettuato è iniziato come "proof of concept" per avere una prima valutazione dell'utilizzabilità del framework Edgex Foundry e delle relative capacità che sono state ampliate grazie alla versatilità dell'architettura, accogliendo un applicativo secondario dedicato alla gestione dinamica dell'orchestrazione del flusso dati. Facoltà che ha reso concorrenziale in termini di caratteristiche il progetto sviluppato, nonostante lo sviluppo della piattaforma EdgeX Foundry non sia ancora maturo.

Infatti, nel corso dell'approfondimento sono state riscontrate alcune inconsistenze nella gestione della memoria permanente da parte del Core Metadata che si presentano al riavvio del sistema rispetto allo stato precedente, che non ne compromettono il ciclo vita e la stabilità una volta avviato.

Le aspettative sono alte e ben motivate dai risultati che dimostrano come sia facilmente adottabile soprattutto per piccole e medie imprese con un costo estremamente contenuto. Oltre alla flessibilità di applicazione che evita l'acquisto di nuovo materiale dedicato, non essere soggetti ad alcun costo di licenza è tanto vantaggioso quanto avere la possibilità di personalizzare le funzionalità della piattaforma da parte di personale già assunto presso il cliente che la adotta.

Non creando vincoli sull'adozione di macchinari di ultima generazione, chiunque abbia a disposizione un apparato sensoristico digitalizzato, è ve-

rosimile poterlo sempre collegare alla piattaforma EdgeX Foundry grazie alla possibilità di compilare un proprio servizio, sia appunto nel south-side che nel north-side attraverso gli SDK a disposizione.

Per questo, un primo passo sarà verificare sul campo i risultati studiati nel virtuale, osservarne il comportamento con protocolli e dispositivi differenti, verificandone l'utilità del sistema sviluppato rispetto all'attuale in uso presso l'azienda. Attraverso un accurato *tuning* dei parametri nella modellazione distribuita del framework per una integrazione, erogazione e distribuzione continua del software, finalizzata alla diminuzione del *time to market* del prodotto completo e personalizzato nelle funzionalità richieste.

L'impatto sulle risorse dell'applicativo sviluppato in Node-RED non è trascurabile, la cui visione complessiva rende il sistema inadatto a lavorare su dispositivi dalle risorse potenzialmente limitate. In questo caso sarà pertanto opportuno considerare di distribuire il framework su più apparecchiature assieme al prendere in considerazione le capacità di inoltro native implementate nel Rules Engine della piattaforma, quindi sulle query Kuiper. Si delega così Node-RED al solo compito di visualizzazione grafica delle letture e aggiornamento, tramite la stessa interfaccia utente, dei parametri delle regole, attraverso la creazione di richieste REST.

In un mondo dotato di risorse limitate è sempre più necessario salvaguardarsi evitando sprechi ed ottimizzando qualsiasi processo necessario allo sviluppo tecnologico e dell'umanità: costruire una base su cui chiunque potrà costruirvi è lasciare un segno tangibile alle future generazioni.

# Elenco delle figure

# Elenco delle tabelle

# Bibliografia


[1] Camera di Commercio, Industria, Artigianato e Agricoltura di Bolzano, "Realtà virtuale nella formazione del personale", 2018, visitato a settembre 2020 https://www.handelskammer.bz.it/it/realt%C3%A0-virtuale-nella-formazione-del-personale

[2] E. Vadala, C. Graham, "Downtime Costs Auto Industry $22k/-Minute - Survey", 2005, Thomasnet, visitato a settembre 2020, https://news.thomasnet.com/companystory/downtime-costs-auto-industry-22k-minute-survey-481017

[3] J.C. Buzby, J.T. Bentley, B. Padera, C. Ammon, J. Campuzano, "Estimated Fresh Produce Shrink and Food Loss in U.S. Supermarkets", MDPI Agriculture, 2015, doi:10.3390/agriculture5030626

[4] A. Forni, R. van der Meulen, Gartner, 2017, visitato a settembre 2020, https://www.gartner.com/en/newsroom/press-relea...

[5] R. Buyya, S. N. Srirama, "Fog and Edge Computing: Principles and Paradigms", John Wiley & Sons Inc, 2019

[6] R. Mahmud, K. Ramamohanarao, and R. Buyya. "Fog computing: A taxonomy, survey and future directions." "Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives". B. Di

Martino, L. Yang, Kuan-Ching Li, A. Esposito, doi.org/10.1007/978-981-10-5861-5, Springer, Singapore, Oct. 2017.

[7] A. Lerner, "The cost of downtime", 2014, Gartner, visitato a settembre 2020, https://blogs.gartner.com/andrew-lerner/2014/07/16/the-cost-of-downtime

[8] Martens B., Walterbusch M., Teuteberg F., "Costing of Cloud Computing Services: A Total Cost of Ownership Approach", 2012, Proceedings of the Annual Hawaii International Conference on System Sciences. 1563-1572. 10.1109/HICSS.2012.186.

[9] Andrae Anders, Corcoran Peter M., "Emerging trends in electricity consumption for consumer ICT", 2013, visitato a settembre 2020, http://hdl.handle.net/10379/3563

[10] Rob van der Meulen, "Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016", 2017, Gartner , visitato a settembre 2020, https://www.gartner.com/en/newsroom/press-relea...

[11] M. Shirer, C. MacGillivray, "The Growth in Connected IoT Devices Is Expected to Generate 79.4ZB of Data in 2025, According to a New IDC Forecast", 2019, IDC Corp., visitato a settembre 2020, https://www.idc.com/getdoc.jsp?containerId=prUS45213219

[12] Edge XRT, IOTech, "The Time-Critical Edge Software Platform", visitato a settembre 2020, http://www.iotechsys.com/what-we-do/products/edge-xrt/

[13] Roy T. Fielding, "Working Group Last Call on the HTTPbis document set", 2013, Adobe, visitato a settembre 2020, https://lists.w3.org/Archives/Public/www-tag/2013May/0005.html

[14] Gerald J. Popek, Robert P. Goldberg "Formal Requirements for Virtualizable Third Generation Architectures", Communications of

the ACM, 1974, DOI 10.1145/361011.361073, http://doi.acm.org/10.1145/361011.361073

[15] Gordon Lyon, "SecTools.Org: Top 125 Network Security Tools", visitato a settembre 2020, https://sectools.org/tag/sniffers/

[16] Modbus Organization, “Modbus FAQ”, visitato a settembre 2020, https://www.modbus.org/faq.php

# Ringraziamenti

*A chiunque mi abbia supportato e sopportato*

Scritto interamente in

LaTeX