

Technological Infrastructures

Part I - Prof Ciavotta

1 Componenti di NIST

NIST sviluppa standard di riferimento per il pubblico. Software Architecture è un'organizzazione globale di sistemi software, e consiste in:

- Divisione della componenti software in sottosistemi;
- Definizione delle politiche con cui questi sistemi interagiscono;
- Definizione delle interfacce tra le varie componenti.

Un'architettura di riferimento è essenzialmente un template (scatola vuota con elementi prefissati), fornisce solo il vocabolario usato comunemente per discutere le implementazioni di un dato software.

Un'architettura di riferimento per il software non è altro che architettura software dove le strutture e i vari elementi e relazioni sono forniti dai template.

L'architettura di riferimento, fornita da NIST, per i Big Data:

- Fornisce un linguaggio comune per i stakeholders;
- Incoraggia aderenza ai standard comuni;
- Permette di implementare le architetture con una certa consistenza;
- Illustra e migliora la comprensione delle componenti, processi e sistemi di Big Data;

1.1 I 5 ruoli principali per Big Data

L'architettura concettuale dei Big Data è un'architettura a croce con due assi: Information value (IV) e Information Technology (IT) (Fig 1).

I 5 ruoli principali sui 2 assi dei Big Data sono:

1.1.1 System Orchestrator

Il system orchestrator coinvolge spesso anche Information Value chain, poiché si preoccupa di implementare e monitorare i processi business a livelli enterprise e le varie politiche sui dati: Redere i dati accessibili per un tempo limitato oppure fornire i dati a velocità diversa (passando il dato in memoria al disco). Può assegnare/fornire componenti framework fisici o virtuale al sistema, questa assegnazione può essere molto spesso elastica ed indipendente. Può fornire supporto GUI e collegare i varie applicazioni a un livello alto, e attraverso il management fabric monitorare i carichi e il sistema per garantire/specificare la qualità del servizio necessaria per i vari carichi. E' molto spesso centralizzato. Es. Ambari/Cloudera

1.1.2 Data Provider

Può essere sia un software (ad es. in una pipeline più grande) che una persona. Se è una persona metterà i suoi dati e userà gli strumenti di collection e curation/preparation per caricare i dati sul sistemi e migliorarne la qualità, Se un

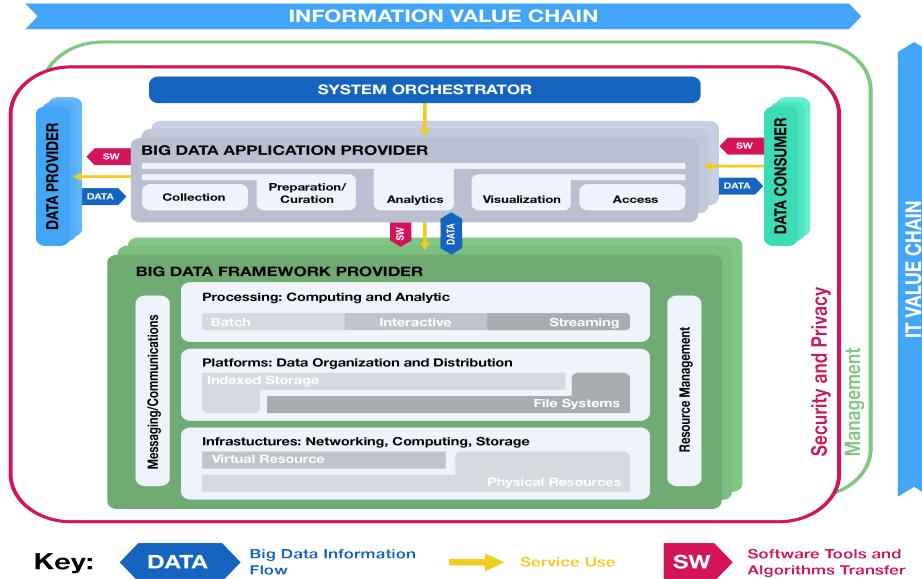


Figura 1: NBDRA Conceptual Model

software metterà i suoi dati a disposizione attraverso delle interfacce come Apache Scoop. Il data provider può essere interno o esterno alla piattaforma, deve fornire i diritti di accesso ai dati, ed è obbligato a seguire le policy di privacy e security fabbric. I dati possono essere inseriti in pull o push. es. Flume per caricare i dati da MySQL a HDFS.

1.1.3 Data Consumer

Riceve i output dei sistemi BigData, può anche lui fare pull e push dei dati, può usare le informazioni per data reporting, retrieval/search e visualization. Ci deve essere l'autenticazione ed autorizzazione da parte della privacy and security fabbric per la comunicazione tra l'architettura e il Data Consumer.

1.1.4 Big Data Application Provider

Corrisponde alle attività tipiche di un Data Scientist, che esistono anche nei sistemi tradizionali ma hanno delle trasformazioni nella implementazioni con i Big Data. Queste attività sono:

- Collection: Si occupa di gestire l'interfaccia fornita dal Data Provider, salva/gestisce questi dati in una certa zona affinché questi non vengano persititi, inoltre implementa funzionalità di estrazione dati dal data provider;
- Preparation: Effettua Data Validation, rimozione outlier, standardizzazione, formattazione e arricchimento. Cerca di promuovere dati di alta qualità;
- Analytics: Estrazione conoscenza dai dati, sfruttando il software sottostante del Big Data Framework Provider;

- Visualization: Presentazione dati in maniera visuale;
- Access: E' l'opposto della collection, si preoccupa di esporre i dati verso l'esterno.

1.1.5 Big Data Framework Provider

Fornisce le infrastrutture per supportare i Big Data Application Provider. Si occupa in particolare di:

- Processing dei dati - ha una dualità: da una parte è un framework che mette a disposizione delle interfacce per la programmazione per fare certe cose (es. MapReduce) e dall'altra parte definisce come l'implementazione viene effettuata. I framework possono variare tra un processamento batch e in streaming.
- Platforms per l'organizzazione e immagazzinamento dei dati - può contenere i meta-dati insieme alle descrizioni semantiche dei dati. Può sia essere relazionale distribuito che non relazionale.
- Infrastructures per l'esecuzione fisica del nostro software, è l'insieme delle risorse computazionali fisiche o virtuali sulle quali il nostro sistema Big Data gira, può essere costituito da server di grandi o piccole dimensioni. Queste componenti forniscono:
 - Networking - Possono essere definiti attraverso software e possono essere reti fisiche che può essere a sua volta partitionato in reti virtuali. Possono essere reti puramente virtualizzate cioè tutto quanto (firewall, router, load balancing) è realizzato in maniera virtuale (es. VM dentro la nostra macchina);
 - Computing - Hardware, Software, OS, memoria per il computing;

- Storage - dischi per storare (in locale), RAID, in rete ecc.;
- e altri servizi come il Raffreddamento, l'apparato elettrico e la sicurezza

Può essere deploys su ambienti fisici che virtualizzati (nativi, hostati o contenerizzati).

Inoltre ha 2 ruoli diffusi nelle 3 componenti sopraindicate:

- Comunicazione e messaggistica tra le componenti;
- Gestione delle risorse per l'integrazione delle componenti.

1.2 I 2 ruoli diffusi per Big Data

Questi 2 ruoli prendono il nome di Fabric, il termine Fabric(tessuto) viene usato perché questi due ruoli sono cross-cutting, cioè sono presenti un po ovunque nell'architettura.

1.2.1 Management fabric

Le due attività principali associate sono:

- Gestione del sistema per provvedere le risorse, gestione dei software e pacchetti e infine la gestione delle configurazioni e performance delle varie pipeline;
- Ciclo di vita dei Big Data, BDLM (Big Data Life Cycle Management), contiene l'enforcing delle Policy (es. encoding/decoding), gestione dei meta data (data governance), accessibilità dei dati, data recovery e la loro preservazione.

1.2.2 Security and Privacy Fabric

Si occupa delle tre caratteristiche tipiche delle security:

- Autenticazione: Indica tutte le attività che validano l'utente;

- Autorizzazione: Una volta autenticato l'utente verifica i suoi permessi, es. alcuni dati potrebbero non essere accessibili a certi utenti;
- Auditing: riguarda la registrazione degli eventi che accadono nel sistema, può far partire un allarme in caso di evento anomalo o a posteriori analizzare la sequenza di eventi (con i file log).

2 Virtualizzazione

Per i computer sono stati definiti con le 5 componenti classiche:

1. Input Devices: Tastiera ecc.
2. Output Devices: Display ecc.
3. Storage Devices: Volatile(RAM), Permanente(HD, SSD)
4. Processore:
 - Datapath
 - Control
5. Network

La virtualizzazione permette l'esecuzione di più sistemi operativi simultaneamente sulla macchina in maniera totalmente isolata. Può essere visto come una emulazione di un software o hardware su cui altri software possono eseguirsi, questo ambiente emulato è detto virtual machine. Il concetto di VM è stato sviluppato negli anni 60 da IBM sui mainframes. Viene abbandonato con la nascita di PC moderni e ripreso con la crescita recente di cloud. La virtual machine viene ottenuto attraverso un Virtual Machine Monitor (VMM) detto anche ipervisore.

L'**Hypervisor**(Ipervisore) è un software che giace sotto gli OS virtualizzati per offrire le funzionalità di condivisione delle risorse disponibili in modo tale che il programma o OS in esecuzione veda queste risorse come se fosse a lui dedicate.

Le risorse sono CPU, memoria, storage e la rete. Vi sono diversi benefici della virtualizzazione:

- Visione unificata delle risorse es. vedo tanti dischi come un unico disco;
- Consolidazione delle risorse virtualizzate, in modo da avere utilizzo ottimale delle risorse;
- Facilità nell'implementazione della ridondanza per copiare gli ambienti virtualizzati;
- Facilità la migrazioni di sistema su un altro, inoltre se non cambio ipervisore la macchina(virtuale) funzionerà identicamente a prima;
- Gestione centralizzata del hardware e software.

Altri benefici/proprietà della virtualizzazione sono:

Workload Isolation: attraverso la virtualizzazione è possibile isolare completamente i programmi, che ha miglioramenti anche nella sicurezza, inoltre aumenta affidabilità poiché il fallimento di un programma non comporta fallimento dei programmi poiché sono isolati, inoltre si risolvono anche i problemi riguardanti i conflitti di librerie in questo modo. Infine si ottiene un controllo sulle performance poiché l'esecuzione di una VM non affligge il performance dell'altra;

Workload Migration: ciò aiuta in:

- Mantenimento di Hardware;
- Load Balancing;
- Fault Tolerance;
- Disaster Recovery.

Poiché possiamo spostare tutto l'ambiente virtualizzato su una nuova macchina in maniera abbastanza trasparente, per fare ciò la macchina dovrebbe essere sospesa, totalmente serializzata per essere inviata nella rete, migrata su una nuova macchina e fatta ripartire immediatamente o solo salvata senza esecuzione.

Consolidazione: Sfruttando il Workload Migration è possibile consolidare macchine separate

su un'unica piattaforma riducendo i costi (usata molto spesso nei Datacenter in orari non di picchi).

I diversi tipi di Hypervisor sono (Fig 2):

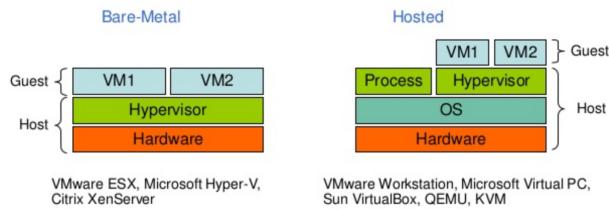


Figura 2: I tipi di Hypervisor

- **Hosted:** in questo caso l'ipervisore è un processo che gira al di sopra del sistema operativo e permette l'esecuzione di più macchine Guest. In questo caso quindi bisogna installare prima un OS su cui verrà installata VMM (ipervisore) e a questo punto l'host potrà eseguire le applicazioni all'interno della sua finestra. Il vantaggio qua è la facilità di installazione e configurazione, inoltre la HostOS e GuestOS rimangono non modificati e non dipendono dal particolare hardware, ma gli svantaggi sono la degradazione delle performance e la mancanza di supporto real time OS poiché vi sono varie entità/software in mezzo;
- **Bare-Metal:** in questo caso l'ipervisore funziona direttamente al di sopra del Hardware. In questo caso l'ipervisore è un OS molto leggero e comunica direttamente con hardware al posto di dipendere su un altro OS. I vantaggi sono miglioramento in I/O e supporto real time, mentre gli svantaggi sono la difficoltà di installazione e configurazione e la dipendenza dal tipo di hardware specifico.

Ci sono principalmente 2 tecniche di virtualizzazione: **Software Virtualization** (di cui abbiamo parlato fino ad adesso) e **Hardware Assisted Virtualization**.

Nella **Virtualizzazione Totale** VMM si preoccupa di emulare in maniera completa tutto l'hardware, quindi avremo un processore, memoria, disco e network virtuale. In questo caso OS ospite non è consapevole dell'esistenza dell'ambiente virtuale e ogni macchina è del tutto indipendente. A livello CPU, avviene la traduzione binaria: Questo avviene in diversi anelli di sicurezza, in particolare gli anelli sono 4, dove l'anello 0 è quello più privilegiato e permette l'esecuzione del codice direttamente sul hardware e qui viene eseguito il kernel dell'OS. Le applicazioni dell'utente vengono eseguite sull'ultimo anello (anello 3).

L'ipervisore gira sull'anello 0 mentre le GuestOS gira sull'anello 1, quindi hanno più permessi delle normali applicazioni. VMM ha accesso sull'anello 0 per avere accesso diretto sulla CPU piuttosto che virtualizzarla.

La **Para-Virtualizzazione** ha un approccio diverso rispetto alla virtualizzazione totale (è una via di mezzo tra total virtualization e bare-metal), in questo caso il Guest sono consapevoli della VMM e usa chiamate speciali in alcuni casi per essere eseguite direttamente sul hardware e ciò comporta un miglioramento nella performance ma ciò lo rende meno flessibile.

La **OS-level Virtualization (Containerizzazione)** non usa la VMM, la virtualizzazione è fornita direttamente dal HostOS che esegue tutte le funzioni di un ipervisore totalmente virtualizzato, quindi ha una partizione puramente virtuale delle risorse, e ciò comporta un'assegnazione flessibile delle risorse alle varie applicazioni. Es. Docker.

Ci sono 3 modelli di servizio:

- 1. Server Virtualization:** supponiamo di avere diversi server su macchine diverse, in caso di un problema (crash di un nodo) o vi è un bisogno di upgrade, in caso di macchine fisiche dovrò prendere lo stesso modello o lo stesso vendore, la soluzione è quella di sfruttare una medesima macchina con un virtualizzatore e mettere i diversi server insieme. In questo modo ho una consolidazione, risorse condivise, una gestione centralizzata, facilità di migrazione, maggiore ROI e meno spazio occupato. La Disaster Recovery e scalabilità viene facilitata, diversi modelli (hardware) a scelta (basta che sia uguale l'ipervisore) e ho maggiore disponibilità.
- 2. Desktop Virtualization:** è la tecnologia che separa l'ambiente desktop e le applicazioni software dal cliente fisico che lo usa. Il nostro desktop diventa un thin client che usa PC privo della memoria RAM e potenza calcolo necessaria per far funzionare una macchina reale, la macchina gira su un pool di VM su un server su un data center. I benefici sono molteplici: Upgrade di software e OS facilitato, alta disponibilità, fault tolerance, accessibile da LAN, WAN, Internet, inoltre vi è la possibilità di far eseguire una piccola parte della computazione dal dispositivo locale.
- 3. Application Virtualization:** Molto simile alla Desktop Virtualization, solo che al posto di tutto il desktop, sono le applicazioni ad essere in virtualizzate, quindi un'applicazione che gira sul desktop, fa la computazione sul cloud, una piccola parte della computazione può essere effettuata anche in locale. I vantaggi sono molto simili a quelli del Desktop Virtualization, un vantaggio aggiuntivo può essere che pago solo quello che uso, riducendo i costi delle licenze.

3 Cloud

Definizione “vera” del cloud da ricordare: è un tipo di servizio distribuito di sistemi interconnessi e computer virtualizzati dinamicamente provvigionati e presentati come un'unica risorsa computazionale basato su service-level agreement. Sostanzialmente il cloud è la possibilità di avere accesso alle risorse computazionali attraverso la rete on-demand ed è composto da 5 punti chiave, 3 modelli di servizio e 4 modelli di deployment.

3.1 The 5 Properties of cloud

Le 5 proprietà aggirano attorno ad una idea centrale del cloud: Utility Computing, SOA (Service Oriented Architecture) + SLA (Service Level Agreement).

Utility Computing nel senso che il service provider fornisce le risorse computazionali e le infrastrutture che servono al cliente e li fa pagare in base all'utilizzo piuttosto che con un costo fisso, come i servizi on-demand, per massimizzare l'uso efficiente, minimizzando i costi nella maniera meno trasparente possibile (senza mostrare al cliente tutto ciò che si fa).

SOA (Service Oriented Architecture): è un insieme di servizi che comunicano tra di loro poiché non basta di solito un solo servizio per implementare un servizio web completo per utente.

SLA (Service Level Agreement): è un contratto tra il service provider e il cliente che specifica il livello di servizio che il service provider deve fornire in termini di QoS. Il **QoS** (Quality of Service) è un set di tecnologie per gestire il traffico della rete in modo da migliorare la user experience, ormai è associato a un significato più generico riguardante le valutazioni di tecnologie non funzionali, cioè non mi interessa solo la funzionalità (il risultato della mia richiesta) ma an-

che la sua efficienza.

Le metriche più comuni delle SLA sono up-time e down-time, Response time. Se le metriche garantite non vengono rispettate vi sono delle penalità sui service provider.

La tecnologia grazie al quale cloud funziona sono:

- Hardware Virtualization;
- Computing distribuito e parallelo;
- Service-oriented Computing;
- Autonomic Computing (reazione a cambiamenti del sistema).

Le 5 proprietà/caratteristiche che identificano il cloud sono:

1. Scalabilità, Elasticità:

- Scalabilità è una proprietà del sistema di crescere in maniera graduale col crescere di richieste, può essere orizzontale (Scale In & Scale Out) o verticale (Scale Up & Scale Down).
- Elasticità è l'abilità di adattarsi automaticamente per inizializzare la scalabilità.

Ciò si può ottenere attraverso provisioning dinamico, che fa riferimento è un ambiente complesso che permette la allocazione e deallocazione on-demand delle istanze/risorse attraverso applicazione o un console amministrativo. Di solito vengono messe delle regole per automatizzare il provisioning, ottenendo così un abbassamento dei costi e performance migliorata.

2. Disponibilità, Affidabilità:

- Availability è il rapporto tra il up-time e tempo totale di esecuzione;
- Affidabilità è la capacità di funzionare in situazioni particolari (rottura di un nodo, attacco di hacker) per un certo tempo.

Questi punti possono essere ottenuti attraverso sistemi di:

- Fault Tolerance è la capacità del sistema

di continuare ad funzionare anche dopo un fallimento di uno dei suoi componenti, spesso il servizio viene degradato proporzionalmente alla gravità del fallimento ma il servizio continua ad funzionare.

Le caratteristiche principali sono che non ha un singolo punto di fallimento, la componente fallita viene individuata ed isolata per prevenire la propagazione del fallimento;

- Resilience è l'abilità di offrire e mantenere un livello di servizio accettabile anche dopo un fallimento, essenzialmente ritornare allo stato di funzionamento dopo un caso di fallimento del sistema (ad es. viene a mancare l'elettricità). Quindi i sistemi devono avere le policy e procedure per il recupero, ad es. Backup (dei dati off-site oppure di tutto il sistema) oppure preparazione contro fault come un una corrente non-interrompibile (UPS) oppure sbalzi di corrente/tensione.
- Sicurezza, che riguarda l'impiego di politiche e tecnologie e sistemi di controllo per proteggere applicazioni e infrastrutture da accessi malevoli, quindi abbiamo il suo impiego in:
 - la protezione i dati, mantenendo l'accesso ai dati riservato solo in base ai privilegi;
 - Gestione dell'identità per garantire l'accesso alle risorse in base ai loro privilegi (protezione dei dati);
 - Sicurezza dell'applicazione riguarda la possibilità di blindare le nostre applicazioni per accessi malintenzionati;
 - Privacy per oscurare i dati in base alle leggi privacy e gestiti solo da utenti competenti.
- Gestibilità, Interoperabilità: La Gestibilità riguarda la gestione di questi sistemi

attraverso un singolo punto di accesso mentre la interoperabilità è una proprietà di un prodotto o sistema per lavorare con altri prodotti/sistemi. Lo si ottiene attraverso **System control automation** e **System State Monitoring**

che è un processo che monitora lo stato dei hardware, metriche dei performance, i log dei sistemi, il pattern dei accessi alla rete ecc. E' anche collegato al sistema di Billing, poiché gli utenti pagano ciò che usano e il cloud provider deve registrare le risorse/servizio usate da ciascun utente.

4. Accessibilità, Portabilità

5. Ottimizzazione, Performance Il Load Balancing

è una tecnica per la distribuzione del workload su un sistema di più computer, CPU, HD ecc. per ottenere l'utilizzo ottimale, migliorando così: l'utilizzo del sistema, la performance e l'efficienza energetica riducendo overload.

Il **Job Scheduler** è un applicazione che ha il compito di eseguire i processi in background (chiamati anche processi batch). Nel cloud i task intensivi computazionalmente o task che crescono dinamicamente vanno pianificati con Job Scheduler.

Da un punto di vista dell'utente, lui non vuole sapere come e cosa viene fatto né chi gestisce il servizio, ma vuole solo un servizio funzionale.

3.2 The 3 Service Model of cloud

Iniziamo con i tre modelli del servizio del cloud sono:

1. **IaaS (Infrastructure as a Service)**: es. le macchine virtuali, in questo caso il provider/vendor gestisce la virtualizzazione, il cliente ottiene le risorse virtuali, di solito da un catalogo che contiene le specifiche hardware (virtualizza-

to) pre-selezionate dal provider. Nella maggior parte dei casi gli OS sono anche prefissate per questioni di prestazioni/stabilità e compatibilità con hypervisor da loro usato. Gestito di solito da un system administrator.

2. **PaaS (Platform as a Service)**: ci viene fornito una scatoletta dove scrivere ed eseguire/testare le applicazioni, il provider sarà il garante del fatto che il sistema sia abbastanza responsive e abbia un runtime sufficiente. E' meno flessibile rispetto allo IaaS ma allo stesso momento lo sviluppatore perde meno tempo nella configurazione delle macchine. Gestito di solito da sviluppatori.

3. **SaaS (Software as a Service)**: es. GMail quindi utente non può fare niente tranne usare il software. Usato da utenti o business analyst.

In realtà il cloud vero è un po' più offuscato di quanto detto sopra, spesso non è facile collocare il modello di cloud di un'azienda esattamente in uno dei 3 modelli.

Un aspetto molto importante da considerare è quello economico, cioè grazie al modello pay-as-you-go cloud il costo capitale (CAPEX) si trasforma in costo operazionale (OPEX), inoltre il rischio di errore nella scegliere le macchine sparisce, se ho bisogno di più potenza richiedo una macchina più forte e se ho bisogno di meno potenza abbasso la potenza e risparmio. Allo stesso momento i cloud service provider hanno diversi benefici: profitto sfruttando l'economia di scala (comprare un servizio costa X, ma comprare N server non costa N*X), possono capitalizzare sui loro investimenti (amazon che vende la potenza residua) o possono sfruttarlo per promuovere un loro prodotto (es. per far usare .NET ai sviluppatori).

3.2.1 IaaS

IaaS fornisce automaticamente il processing, storage, network e altre risorse fondamentale per il computing, e l'utente può installare i software che sono per lui necessari.

Lo IaaS supporta queste 3 caratteristiche del cloud:

- Gestibilità e Interoperabilità: Attraverso un interfaccia può gestire tutte le VM che vuole, le può pagare come vuole;
- Disponibilità ed Affidabilità: gestita dal cloud provider attraverso le zone di disponibilità;
- Scalabilità ed elasticità: Proprietà del sistema ma deve essere implementato dall'utente.

L'architettura è composta dall'hardware seguita da un layer di virtualizzazione e l'utente ha accesso a due interfacce:

- Interfaccia per gestione delle risorse, queste risorse sono:
 - VM: creazione, cancellazione, gestione, sospensione delle VM
 - Virtual Storage: Allocare spazio, ridurre/aumentare del DB, scegliere servizi con velocità di lettura/scrittura diversi (per prezzo);
 - Virtual Network: gestione delle IP associate alle VM, registrazioni al dominio delle IP, scegliere la larghezza di banda.
- Interfaccia del il monitoring del sistema: viene messa a disposizione da VIM: L'orchestrazione delle macchine/risorse, in maniera rapida e dinamica, su un server viene gestita dalla Virtual Infrastructure Manager(VIM), diverse metriche vengono usate per il monitoraggio es:
 - VM: CPU usage, memory usage;

- Virtual Storage: utilizzo del disco, livello di duplicazione dei dati e velocità di accesso al disco;
- Virtual Network: L'uso della banda, lo stato di connettività e il bilanciamento sulla rete.

3.2.2 PaaS

PaaS mette a disposizione dell'utente i linguaggi di programmazione e i tools (come IDE) supportati dal provider, quindi il controllo sul deployment dell'applicazione è nelle mani del consumer ma le infrastrutture sottostanti sono gestite dal cloud provider.

Alla base vi è una archittettura come quella di IaaS, vengono aggiunti i Runtime Environment e sopra questi ci sono i Programming IDE e le API/tools supportati dall'ambiente, spesso hanno in comune le funzioni come: computation e lo storage.

Il Runtime Environment si riferisce a una collezione di software, implementati con una collezione di librerie, le proprietà fornite dal Runtime Environment sono:

- Gestibilità ed Interoperabilità
- Performance ed Ottimizzazione
- Disponibilità ed Affidabilità
- Scalabilità ed Elasticità

L'interfaccia messa a disposizione per il controllo del sistema mette a disposizione un set di azioni in base a:

- Policy based control: le azioni seguono delle regole per prendere delle decisioni quindi azioni seguite con if <> then <>
- Controllo del workflow: descrizione del flusso di installazione e configurazione delle risorse oppure dei daemon.

3.2.3 SaaS

SaaS: vengono fornite al cliente applicazione del provider in cloud, l'utente non può gestire o sviluppare l'applicazione o gestire l'architettura nel cloud ma può interagire con esso attraverso interfacce ad es. portali/applicazioni web, che con introduzione del Web 2.0 ha potenziato l'iterabilità con l'applicazione in cloud, es: Facebook(Messenger), Office, Skype, DropBox, sistema CRM, sistema medico nazionale, sistema di trasporto pubblico.

Il punto fondamentale di questo tipo di cloud è l'Accessibilità e la portabilità.

Oss: La differenza tra il portale web e una pagina web è che il portale permette l'integrazione di diverse pagine attraverso dei login.

3.3 The 4 Cloud Deployment Models

I 4 modelli primari che differenziano un cloud, basati sulle possibilità di accesso, la dimensione e la proprietà dei servizi, sono:

- **Public Cloud** (o multi-tenant o external cloud): L'infrastruttura viene messa a disposizione del pubblico generale o almeno alle organizzazioni grandi attraverso pagamento per il suo uso, le caratteristiche sono:

- Infrastruttura omogenea per garantire le medesime prestazioni a due utenti che pur trovandosi in posti diversi usano lo stesso servizio;
- Policy comuni;
- Risorse condivise;
- Infrastrutture viene affittata;
- Economia di Scala.

- **Private Cloud:** (o on-premise cloud o internal cloud) Viene operato da un'unica organizzazione e può essere gestita dalla stessa

o un'altra organizzazione, cioè la categoria di utenti è limitata. A differenza del public cloud le caratteristiche sono:

- Infrastruttura eterogenea per via del costo e dal fatto che una azienda privata non butta via un hardware solo per averlo omogeneità;
 - Policy customizzate ad-hoc per gli utenti;
 - Risorse dedicate;
 - Infrastrutture gestita da house;
 - End-to-end control sui processi.
- **Community Cloud:** è una struttura gestita da diverse entità, in pratica più entità mettono insieme i loro cloud privati per generare un super cloud, ad esempio alcune università americane possono unire i loro cloud per qualche ricerca specifica ecc.
 - **Hybrid Cloud:** è la composizioni di più tipi di cloud, ad esempio un'azienda privata crea il suo cloud privato per la gestione di informazioni sensibili e usa il cloud pubblico per il resto, oppure lo usa per la scalabilità in caso di traffico aumentato.

4 Amazon ECOSYSTEM-IaaS

 sono creati per lavorare indipendentemente ma possono interagire tra di loro, condividendo tra di loro le stesse convenzioni di nomi e autenticazione e minimizzando le connessioni interne.

Amazon mette a disposizione il concetto di regione e di zona di disponibilità:

- Ciascuna **Availability Zone** è un data center indipendente con la propria griglia di potenza e connessione della rete, le zone all'interno di una regione sono collegate tra di loro attraverso connessione a latenza bassa. Il nome deriva dal

fatto che sono zone ad alta disponibilità, quindi se una zona fallisce il suo effetto non viene notato dalle altre zone creando così una stabilità ed alta fault tolerance.

- **Regione** è un cluster di Availability Zone localizzati in una area geografica. Quando si crea una istanza Amazon ci dà la possibilità di scegliere una availability zone (o di default lo si lascia far scegliere ad Amazon) e all'interno della stessa regione è possibile distribuire la propria istanza su più availability zone, quindi se una zona fallisce l'altra (molto probabilmente) continua a funzionare. Amazon non fa pagare i trasferimenti di dati all'interno della stessa regione, ma tra regioni le comunicazioni vengono fatte pagare.

Esiste AWS GovCloud che è una regione AWS isolata e permette di gestire dati estremamente sensibili e può essere acceduta solo da cittadini Americani verificate dal governo Americano e sul suolo Americano.

Uno dei servizi principali di AWS è **Simple Storage Service (S3)**, un storage object che è un sistema di archiviazione piatto cioè non è possibile creare cartelle al suo interno, i dati vengono salvati in formato binario e vengono distribuite automaticamente. L'accesso può essere effettuato da chiamate web (REST, Soap, BitTorrent) o query SQL e gli oggetti possono anche essere molto grandi (fino a 5TB ciascuno).

S3 sta alla base di tutta l'infrastruttura che sta alla base di Amazon e-commerce. Oltre a S3 Amazon fornisce anche un **EBS (Elastic Block Store)**, questo disco può essere formato, montato e usato come hard disk locale, è un volume che persiste indipendentemente dal resto. Mentre le VM possono morire, i volumi rimangono sempre, e la garanzia di durabilità viene fornita da Amazon.

EBS è categorizzato in SSD e HDD, a seconda

del tipo di macchine/dischi si paga in maniera diversa.

I **Security Group** definiscono l'insieme delle connessioni possibili per una certa istanza, e questi insiemi possono essere protocolli, porte, range delle IP.

4.1 EC2 - Elastic Cloud Computing

EC2(Elastic Cloud Computing) è uno dei servizi offerti per la creazione/gestione delle macchine virtuali on-demand. EC2 si basa sul concetto di data center programmabile, la possibilità di creare una propria infrastruttura (macchine in più availability zone o regioni) tramite linguaggi di programmazione. I concetti chiave che costituiscono EC2:

- **Amazon Machine Image (AMI)**: è un file contenente una descrizione di una VM, cioè eventuali software e le configurazioni, possono essere pre-built, create, modificate e vendute. Ciascun AMI ha un ID unico;
- **Istanza**: rappresenta una copia di AMI in esecuzione, multiple copie di un'unica AMI può essere messa in esecuzione;
- **Elastic IP address**: allocazione di un IP statico e collegarla alla propria istanza, ciascuna istanza può avere al più un IP statico.

I core messi a disposizione dal server (di cui non sappiamo praticamente nulla) vengono virtualizzati, dividendo prima in core poi dividendo ulteriormente il core-time. L'unità di misura è Elastic Compute units che corrisponde a Intel Xeon (o AMD Opteron) da 1.0-1.2 GHz del 2007.

Le risorse possono essere:

- **Persistenti**: anche in caso di fallimento del hardware Amazon ci garantisce la sua persistenza attraverso ridondanza, recupero automatico.

tico e failover automatizzato. Le componenti persistenti sono:

- Elastic IP Address
- EBS
- Elastic Load Balancer
- Security Groups
- AMI salvate in S3 o EBS

- Effimere: l'utente deve garantire la sua ridondanza e mantenimento attraverso altri servizi di EC2, in caso di fallimento i dati salvati vengono in generale persi. Le istanze sono effimere.

I modelli del prezzo sono:

- **On demand:** Il pagamento avviene per le capacità (tempo) usate con nessun obbligo a lungo termine.

- **Reservati:** Si fa una sorta di abbonamento con un pagamento di una somma prima e poi le macchine si possono usare per la durata dell'abbonamento a un prezzo scontato, è disponibile in 3 tipi Light, Medium e Heavy e può essere per 1 o 3 anni e può far risparmiare fino al 71% rispetto a On-Demand. Attenzione non ci riserva delle risorse per la durata dell'abbonamento ma solo il prezzo.

- **Spot:** Si fa asta per la potenza computazionale non usata da Amazon EC2, sono quelle che costano di meno, perché non mi viene garantita la durata della macchina nel tempo, nel caso il prezzo offerto è maggiore del costo necessario per eseguire quella macchina Amazon darà la sua macchina, ma siccome il prezzo di elettricità è variabile e se dovesse superare il prezzo offerto Amazon può spegnere tale macchina senza avvisare.

Un altro indice di costo è **Data Transfer**, il trasferimento dei dati dentro o fuori dalla istanza EC2 vengono pagati (in base alla quantità) se non il trasferimento avviene al di fuori della

regione. Il trasferimento dentro la regione è completamente gratuita.

4.2 AutoScaling

Uno servizio molto importante di Amazon è **AutoScaling** (Fig 3). Per far funzionare AutoScaling è importante anche il **Elastic Load Balancer**, grazie al quale l'utente finale vedrà solo un collegamento e Elastic Load Balancer pensa a distribuire le richieste alle nostre macchine. Le istanze EC2 possono essere categorizzate in auto scaling groups, di solito con un range tra un minimo e un massimo, se una nuova macchina entra in esecuzione oppure viene de-allocata il Load Balancer viene avvertito della modifica in modo che possa funzionare bene. Gli Auto Scaling group sono istanze EC2 che condividono caratteristiche simili e devono essere fatte per scalare, e aggiunge automaticamente un nuovo nodo in base a policy definite dall'utente, programmazione oppure health checks e workload e

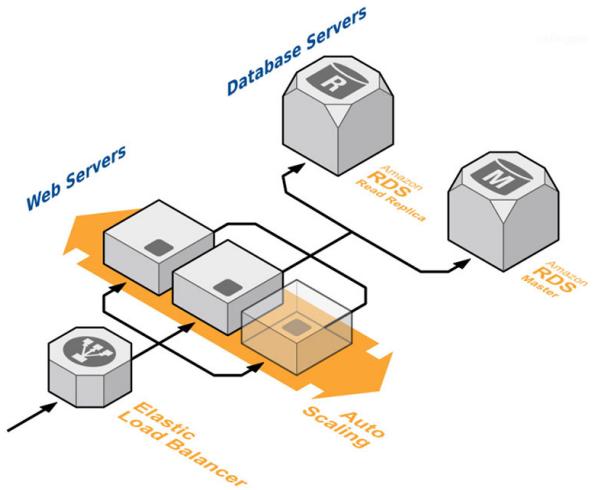


Figura 3: Esempio di Auto Scaling for AWS

la nuova istanza può venire anche da availability zone diverse della stessa regione.

Auto Scaling group contiene solo un launch configuration che descrive le istanze che devono far partire (contiene i parametri delle AMI), se si aggiorna il launch configuration questo affligerà solo le nuove istanze, le vecchie istanze rimangono tali ma esse vengono terminate prima durante un scale in.

Attenzione una macchina può essere o spenta o terminata, nel caso viene terminata viene eliminata completamente e AutoScaling group termina una macchina.

L'ecosistema di Auto Scaling è composta da:

- Cloud Watch: Per monitorare le istanze EC2, le metriche più usate per il CloudWatch sono:
 - CPU usage
 - Latenza
 - Numero di Richieste
 - Il numero di Host(macchine) in buona salute
 - Il numero di macchine non in buona salute
- Elastic Load Balancer: Per distribuire il lavoro su un numero qualunque di istanze EC2 ed effettuare controlli periodici sulla salute del nodo (basandosi sul tempo di risposta) e in caso non siano in buona salute il load balancer smette di inviare a lui le richieste.
- Auto Scaling: Utilizza i dati collezionati dal CloudWatch per costruire sistemi che possono scalare (dentro o fuori) all'intero del range

Il **Trigger** è un meccanismo per l'attivazione di una policy, in questo caso di aumentare o diminuire il numero dei nodi. Il trigger può essere attivato da un allarme cloud watch (configurato per guardare una metrica di cloud watch) oppu-

re un auto scaling policy che descrive cosa fare in caso di un allarme. Quando si attiva il trigger lancia un processo chiamato Scaling activity che esegue la auto scaling policy. Osservazione Auto Scaling supporta ma non necessita Elastic Load Balancer. In generale almeno due trigger sono necessari, uno per Scale up e uno per Scale down, per mantenere un equilibrio desiderato. Un allarme è una metrica e verifica se questa metrica supera un limite stabilito per un certo tempo. Un'allarme è un'entità in grado di osservare con continuità una certa metrica e ci indica se tale metrica ha superato un certo valore prefissato per un certo tempo, per creare un allarme è necessario specificare:

- Metrica da osservare
 - Il threshold della metrica
 - Il numero di periodo di valutazione
- Gli stati in cui l'allarme si può trovare sono:
- Ok, tutto bene! Metrica è sotto il threshold.
 - Alarm: Metrica è sopra il threshold, è necessaria un'azione.
 - Dati Insufficiente, metrica non disponibile o non ci sono abbastanza dati.

Se l'allarme cambia lo stato e vi rimane per un determinato periodo di valutazione, un'azione viene invocata in base alla policy.

I scenari automatici dell'auto scaling sono:

- Fleet Management: assicura il performance ottimale della macchina, controllando la salute dell'istanza con Health Check: se un'istanza dovesse terminare questa viene individuata e si fa partire un'altra macchina.
- Scheduled scaling: Azioni vengono eseguite in maniera programmata, quindi è una sorta di evento temporale per eventi ricorsivi o scheduled.
- Scaling Dinamico: In base all'allarme su una metrica c'è una politica che risponde

all'allarme. L'azione può dipendere step-wise da gravità dell'allarme oppure facendo tracking di una certa metrica e aggiusto il sistema in modo da calmare l'allarme.

La terminazione della macchina può essere fatta anche per un ribilanciamento delle macchine nelle availability zone, la precedenza dello spegnimento viene data in modo da bilanciare tra le zone seguito dal fatto di voler preservare le macchine con il launch configuration più recente seguito dalla macchina prossima allo scatto dell'ora (per il prezzo che si paga ogni ora). Auto Scaling inizializza una nuova istanza prima di spegnerne una per non compromettere la disponibilità e le performance dell'applicazione.

Dopo che è iniziata la fase di Auto Scaling esiste un periodo di cooldown, in questo periodo nessun'altra attività di scaling può iniziare.

I candidati per autoscaling sono:

- Web Tier - sistemi che forniscono servizi web
- Application Tier
- Load Balancing Tier - sistemi che gestiscono il carico
- Stateless Tier - sistemi sono prive di stato es. funzioni pure, sono prive di memoria e quindi da un punto di visto funzionale non cambia se faccio autoscaling o no.

I candidati che non dovrebbero fare autoscaling:

- Database Relazionali
- Database non-relazionali
- Sistema di caching distribuito
- Elastic Search
- Sistemi con lo stato - es. Kafka, non avrebbe senso fare autoscaling automatico.

5 PaaS with Azure

Essenzialmente il PaaS mette a disposizione un'interfaccia di programmazione, un linguaggio di programmazione (runtime) e tutta una sorta di servizi necessari. Secondo l'approccio tradizionale, si sviluppa un'applicazione in locale e poi si deploya sul cloud, ma vi possono essere dei problemi come "it works on my machine": uno dovrebbe replicare esattamente le proprie condizioni di lavoro sul cloud. Mentre con il PaaS l'ambiente di sviluppo diventa l'ambiente di deployment e tale gestione dipende direttamente dal provider.

I 5 servizi Paas principali di  Azure sono:

- Web App - Servizi PaaS per lo sviluppo di applicazioni Web;
- Mobile App - Servizi per lo sviluppo di applicazioni Mobile (gestione contenuti, utenti);
- Function App - Azioni microscopiche (piccole funzione) che possono essere deployate sul cloud;
- API App - Sviluppare API o usare API di terzi;
- Logic App - Serve per scrivere logica di integrazioni per applicazioni.

L'applicazione Web può avere 2 ruoli:

- Web Role: L'esecuzione di una web app, ha il compito della presentazione.
- Worker Role: Supporta l'esecuzione backend.

Osservazione: Un nuovo role non è una VM è simile ad una sandbox (l'applicazione vede se stessa come l'unica app in corso), quindi si possono avere più ruoli sulla stessa macchina, inoltre nel caso di una WebApp non sappiamo né la memoria né informazioni sul CPU, si pagano solo le richieste servite (o cicli del processore usati),

inoltre viene garantito autoscaling e auto load balancing. La comunicazione tra Web Role e Worker Role avviene attraverso un sistema di code di messaggi, e un load balancer distribuisce le richieste tra i vari Web Role.

Per la persistenza dei dati Azure mette a disposizione:

- **BLOB (Binary Large Object) Storage:** è un object store molto simili al modello S3 di Amazon e può salvare qualunque tipo di file anche di grande dimensioni, è un filesystem distribuito ma la differenza rispetto all'S3 è che BLOB storage è a due strati, il primo livello prende il nome di Container e il secondo livello è BLOB. L'interfaccia con cui si interagisce con BLOBS è REST API, quindi PUT, GET, (UN)DELETE, COPY, SNAPSHOT, LEASE.

- **Azure Cloud SQL:** Server SQL senza overhead amministrativi, si può scegliere se si vuole questo DB su hardware condiviso o riservato, il modello che segue è Pay-as-you-grow. E' con alta disponibilità (99.99% mensili).

- **Tables:** Uno storage leggermente strutturato, è un database key-value quindi NoSQL, ciascuna entità può avere 255 proprietà ed essere grande al più 1 MB, come altri modello key-value abbiamo due chiavi: una chiave per partizione e una per la riga. La chiave identifica univocamente un'entità e usa Timestamp. Osservazione è fortemente **Consistente**, ed è altamente **Available**, quindi secondo il teorema CAP non è particolarmente tollerante alle **Partizioni**.

- **Queues:** Un sistema di code di messaggi simile a Apache Kafka.

- **CosmoDB:** è un servizio equivalente a MongoDB offerto da Azure, è un poliglotta ed è un key-value, graph, column based e document type DB, supporta un indice automatico in base alle query molto richieste esattamente come Mongo. Inoltre garantisce una latenza molto

bassa in tutto il mondo.

Azure offre un suo servizio di Auto Scaling, ma fra le varie possibilità offerte vi è la possibilità di creare auto scaling proprio basato sulle regole. Le regole possono essere:

- Constraint Rules: minimo e massimo numero di istanze in esecuzione per un certo role e lasciar decidere a Azure di decidere come e quando scalare;
- Reactive Rules: seguite in risposta a certi eventi, queste regole sono più complesse rispetto a quello che ci mette a disposizione Amazon, in particolare, possiamo aumentare il numero di Role in esecuzione ma Azure permette anche di cambiare il comportamento della nostra applicazione (Throttling) in maniera dinamica:
 - Rigetto delle richieste: Rigetta richieste di certi utenti in base a certe regole es. rigetta il 10% di richieste dalla Cina o da utenti che hanno già fatto n richieste in un determinato periodo per abbassare il carico;
 - Disabilitare o degradare certe funzionalità finché la situazione non si stabilizza;
 - Ritardare operazioni eseguite per applicazioni con bassa priorità

Come in Amazon si possono definire i Scaling Groups per raggruppare Role multipli e definire le Scaling Rules per tutti i Roles nel gruppo.

6 Containerization/Docker



Docker è un progetto open source che ha l'obiettivo di automatizzare il deployment dell'applicazione all'interno di un software container, fornendo un livello di astrazione superiore a quello

del sistema operativo. L'idea alla base è avere un sistema di packaging uniforme che possa essere eseguito ovunque (architetture x86(64) e kernel linux). Il motivo per cui è utile docker è che lavora sul kernel linux ed è così indipendente dalla particolare distro e posso installarci delle librerie che in generale sarebbero in contrasto con le librerie del sistema.

Docker aderisce alle policy delle OCI e non è l'unico sistema a farlo, tutti i sistemi che vi si aderiscono in teoria dovrebbero permettere l'interoperabilità. Uno degli obiettivi principali è isolare le applicazioni, cioè voglio che non vedano i processi del sistema (per motivi di sicurezza). Inoltre sono lightweight poiché condividono il kernel di OS e non c'è uno strato di virtualizzazioni in mezzo.

L'idea è quella di eliminare hypervisor ed eliminare GuestOS, permettendo l'esecuzione delle applicazioni direttamente sul OS. La tecnologia container ha la dualità immagini - container, immagine è un file/template che contiene i metadati e configurazione e questo file viene usato per creare container, la differenza rispetto alle VM è che il container non è una copia dell'immagine (vedi AMI) bensì una sua esecuzione, un'altra differenza è il tempo di esecuzione rispetto alle VM che ci mettono tempi in ordini di minuti siccome docker è un processo parte in ordini di secondi.

Le immagini sono salvate di solito su Hub o registro locale. Il **filesystem** di docker è solo di lettura ed è a livelli (UnionFS), creare l'immagine a livello permette la massima riusabilità delle immagini, si può ripartire a creare un'immagine a partire da una già esistente, ad esempio se ho un'immagine debian con emacs sopra posso partire da debian e mettere sopra nano e per fare ciò non devo riscaricare debian posso riusare già l'immagine di debian che possiedo. Riesce a crea-

re questo modello a strati seguendo un approccio Copy-on-Write, cioè crea una copia del file modificato. Tutte le tecnologie tranne il filesystem a strati sono dovute usando il kernel linux, uno di queste tecnologie è il **namespaces**: permette l'isolamento dei processi, individuando le cose che può e non può vedere il processo. L'altra tecnologia principale che permette l'esecuzione dei docker è **cgroups**: che ha lo scopo di limitare e monitorare l'accesso alle risorse(CPU, Memoria, I/O, network).

Docker è un demone linux che mostra un'interfaccia REST che è utilizzabile tramite una CLI (command line interface) fornita da docker. Il

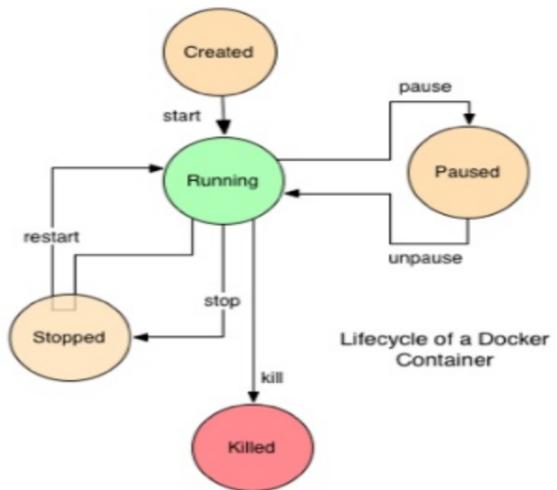


Figura 4: Ciclo di vita del Docker

ciclo di vita di un container è così come indicato in Figura 4 e i comandi principali sono:

- docker create: crea ma non esegue il docker
- docker run: crea ed esegue il docker
- docker stop
- docker start
- docker restart

- docker rm
- docker kill

Essendo un processo come gli altri docker di default può vedere tutta le risorse che il computer ha a disposizione, è possibile limitare con dei soft limits (limite inferiore delle risorse) e hard limits (limite superiore) l'accesso a queste risorse.

Quando si elimina l'immagine in esecuzione si perdono tutti i dati, quindi nasce il concetto di volume che sono cartelle condivise con il sistema (per non perdere i dati). I volumi sono di tre tipi:

- Bind Mount: è una cartella montata è in binding (collegata) a una cartella del sistema host;
- Volume Mount: sono anch'esse cartelle condivise ma gestiti dal docker, evitando problema di breach;
- tmpfs Mount: cartelle che non sono sul disco della macchina ma sulla memoria.

Tutto questo modello dei volumi quando ci troviamo nel cloud non funziona possono esservi diversi errori (fallisce una VM) e il volume è esplicitamente al Host, per rendere la persistenza del volume anche nel cloud docker è stato scritto in una maniera da poter usare dei plugin, il plugin per lavorare in locale (quello che abbiamo visto) si chiama local persist, esistono diversi plugin già creati dai vari provider cloud per funzionare sul loro sistema.

Il building process del docker è l'output di un dockerfile che contiene le istruzioni per far partire e il contesto che sono una serie di altre informazioni necessarie. Le istruzioni sono sequenziali e vengono eseguite dal demone docker. Tra i comandi esistenti i più importanti sono:

- **ENTRYPOINT**: definisce un processo che deve essere eseguita al lancio del container, può avere due forme:

```
ENTRYPOINT[“executable”, “param1”,\n          “param2”]    (exec form)\nENTRYPOINT executable param1\\n          param2      (shell form)
```

La differenza è che la shell form viene comandato dalla shell quindi ad es. si può terminare un processo con ctrl+c.

• **CMD** ha lo stesso significato della ENTRYPOINT, ma ha una differenza, se esiste un già ENTRYPOINT viene eseguito ENTRYPOINT e CMD gli passa i suoi parametri e se esistono molteplici CMD viene eseguito l'ultimo CMD.

Es:

```
CMD[“--port 27017”]\nENTRYPOINT /usr/bin/mongod\n# ENTRYPOINT riceve il parametro da CMD
```

La comunicazione tra i vari container si fa attraverso il docker networking, che è una tecnologia di virtualizzazione della rete, i docker connessi credono di avere a disposizione una propria connessione, questa connessione è un plugin del docker, questa connessione è inoltre distribuita e decentralizzata quindi ogni docker ha una parte d'informazione della rete. Questa idea viene formalizzata nel CNM (Container Network Model) che consiste di 4 elementi principali:

1. **Sandbox**: è una struttura che contiene le configurazioni della rete a livello di container, quindi gestisce le interfacce, la tabella di routing e una serie di informazioni del DNS. Una sandbox può avere più di un endpoint e collegata a più reti ma ogni container può essere collegato solo ad una Sandbox;
2. **Endpoint**: è l'implementazione dell'interfaccia di rete (es. veth - Virtual Ethernet) Un endpoint può essere associato ad una sola rete e una sola sandbox;
3. **Network**: è l'insieme di tanti endpoint che possono comunicare tra di loro;

- Cluster: l'insieme di docker collegati ad una rete.

Quando creiamo una rete questa può essere di tipo:

- NULL: Il container non ha accesso alla rete;
- Host: Il container e l'host hanno la stessa interfaccia di rete;
- Bridge: E' un servizio a cui vari endpoint sono collegati;
- Overlay: Una rete che viene creata tra più host docker;
- Remote: E' un'interfaccia definita in modo tale da essere implementata da terze parti.

6.1 Docker-Compose

E' un linguaggio dichiarativo esprimibile attraverso file yaml, che descrive com'è costruita un'applicazione multicontainer come la rete e volume. In pratica usa i comandi docker dietro e semplifica la vita allo sviluppatore. Il file docker-compose.yml è composta da tre sezioni: Services, Networks e Volumes.

Un **Docker Swarm** è un cluster di macchine che usano docker e si uniscono logicamente in un sistema, ai fini di deployment è come se diventassero un'unica macchina. Il cluster viene gestito non più dalle singole istanze di docker sulle macchine ma da una istanza sola che prende il nome di swarm manager, le altre macchine vengono chiamate worker (o nodes). Il manager può usare diverse strategie per eseguire i container nei vari workers, un esempio è emptiest node dove i container vengono fatti partire nel nodo più vuoto. Solo il manager può decidere di eseguire i comandi o autorizzare l'aggiunta nello swarm di altri worker, i worker quindi mettono a disposizione solo le risorse computazionali e ricevono ordini dal manager.

Le funzionalità di docker swarm:

- Gestione del Cluster integrata con docker-machine, non necessita di software aggiuntivi esterni;
- Modello di servizio dichiarativo;
- Gestione della scalabilità automatica, creando anche repliche di container e costantemente continua a controllare lo stato di vita dei workers;
- Riconciliazione dello stato attuale con lo stato dichiarato dall'utente;
- Gestione della multi host networking con reti overlay;
- Implementazione del servizio discovery per l'individuazione dei nodi in maniera unica;
- Load Balancing, in caso di esistenza di varie istanze dello stesso servizio, queste riceveranno il carico distribuito;
- Sicuro di default poiché i vari nodi comunicano attraverso protocolli http criptati utilizzando tecnologie TLS.

Si possono avere più di un manager ma solo uno è leader gli altri sono detti Reachable e sono una sorta di leader di backup nel caso il leader fallisca ottenendo alta disponibilità, possono usare dei protocolli di consenso per prendere decisioni insieme.

7 Serverless

Esistono altri 2 modelli per il cloud oltre a IaaS, Paas e SaaS: **CaaS** (Container as a Service) e **FaaS** (Function as a Service).

Serverless può essere inteso come BaaS (Backend as a Service) oppure FaaS. Il BaaS è l'incorporazione di servizi autonomi in modalità PaaS di un'applicazione ad esempio il servizio di autenticazione. Nel FaaS lo sviluppatore non deve sviluppare applicazioni intere, ma solo scrivere delle funzioni che siano più semplici possibili, queste

funzioni devono avere un ruolo atomico e granulare. Sono dei container computazionali, che contengono una funzione effimera, che viene eseguita in caso di un evento esterno e queste funzioni sono gestite totalmente dal cloud provider.

In questo caso non vi è un gestione centralizzata, infatti preferisce la coreografia (coordinazione) delle funzione rispetto all'orchestrazione (non c'è un decisore centrale che richiama le funzioni), ottenendo così:

- Flessibilità e Estendibilità: si possono aggiungere altre funzioni senza alterare quelle pre-esistenti;
- Divisione delle preoccupazione;
- Costo ridotto: Non si paga se la funzione non è in esecuzione, quindi se una funzione è raramente usata non pago un server costantemente accesso.

Inoltre non sono più vincolato solo dai linguaggi offerti dal cloud provider, qualunque linguaggio che compila su un processo UNIX (quindi nel docker) va bene. Vi sono dei constraints, ad esempio in AWS una funzione può al massimo durare 5 minuti, se una funzione richiede più di 5 minuti forse conviene prendere un IaaS o PaaS, oppure si possono splittare le funzioni in più funzioni. Inoltre non vi è la garanzia del fatto che gli stati persistano su invocazioni multiple successive poiché le variabili sono salvati su memoria o un disco locale.

Le funzioni in FaaS sono tipicamente triggerate da eventi definiti dal cloud provider ad es. AWS include aggiornamenti sul S3, task schedulati, ricezioni di messaggi in un servizio di coda (come Kafka) oppure per le rispondere alle richieste HTTP che usano API Gateway.

Un altro svantaggio è il così detto cold start - crea una nuova istanza, inizia le funzioni host ecc. e il periodo di accensione dipende dal linguaggio di programmazione, le librerie da cari-

care e la configurazione della funzione e la quantità di codice, pur essendo in ordini di secondi è molto lento rispetto a warm start, che riusa una istanza di Lambda function esistente da una chiamata precedente.

I benefici sono molteplici:

- Costi Operazionali, di Sviluppo e Esecuzione Ridotti;
- Computazione più **Green** ed efficiente.

Vi sono anche dei svantaggi:

- Il controllo del Vendor: I downtime, i limiti, cambiamenti di costo, aggiornamenti di API;
- Problemi di mantenimento;
- La durata di esecuzione;
- Latenza di inizializzazione;
- Testing, Debugging e monitoraggio difficile e limitato.

Quindi il serverless è non ottimale per le funzioni che hanno bisogno di stati e sono più lunghe in generale ad es. Deep Learning Training, Streaming Pesante, Analitiche Hadoop, Gestione DB, streaming video ecc. Sono molto utili per eventi veloci, stateless e event-driven come microservizi, IoT, Inferenze ML, streaming leggero ecc.

8 Apache Spark

L'idea di fondo non è quella di spostare i dati nelle varie macchine ma i codici (data locality), quindi una volta ottenuto il parallelismo e data locality e in più implementiamo un sistema che sia in grado di gestire i fallimenti a vari livelli allora otteniamo il framework chiamato Hadoop. Spark piano piano sta sostituendo una parte dell'ecosistema di Hadoop.

Per introdurre il concetto degli operatori serve capire cos'è il linguaggio funzionale: è un para-

digma di programmazione, che vede la computazione come processo di valutazioni di funzioni matematiche, queste funzioni devono essere immutabili e prive di stato. Le proprietà della programmazione funzionale:

- Composizione di funzioni per ottenere funzione di ordine superiore
- Le funzioni sono senza stato (sono pure), prendono stato grazie ad un accumulatore che è una variabile passata in ingresso a una funzione
- Non possono modificare lo stato di altri funzioni (no side-effect)
- Possibilità di Ricorsione
- Lazy-Evaluation, esecuzione solo quando viene veramente richiesta
- Le funzioni non modificano le strutture dati, ma ne creano altre durante il processo

Grazie a queste proprietà si ottiene la parallelizzazione automatica, performance migliorata, nessun Bug e gestione della memoria.

I due operatori più interessanti per Spark sono:

- **Map**: un operatore di ordine superiore, prende in ingresso una funzione e una input list e applica questa funzione in ingresso a tutti gli elementi della input list restituendo un'altra lista;
- **Reduce**: un operatore di ordine superiore, prende in ingresso una funzione e una input list e applica questa funzione ricorsivamente a tutti gli elementi restituendo una combinazione della lista.

8.1 MapReduce

Una loro applicazione combinata è MapReduce, che serve a processare una grande quantità di dati con algoritmi parallelizzati e distribuiti su un cluster. Il sistema shuffla i dati nelle fasi in mezzo a Map e Reduce. I nodi del sistema condividono

no un filesystem distribuito e vengono sfruttati nelle due fasi:

- Fase **Map**: Il master node partiziona i files in M splits attraverso una chiave, e assegna ai slaves i maps da eseguire. Questi slaves dopo aver eseguito il task di Map, scrivono il loro output sul disco partizionato in R regioni attraverso una funzione hash;
- Ad altri slaves viene assegnato il task **Reduce**, ciascuna slave legge i dati dal disco del mapper corrispondente e li raggruppano in base alla chiave ed eseguono la funzione di Reduce.

Questo è l'architettura usata in Hadoop 1.x, uno dei suoi svantaggi sono il blocco che si crea ai vari slaves, poiché quando lavorano i Mapper i Reducer sono fermi e quando lavorano i Reducer i Mapper sono quelli fermi, quindi non si sfrutta la massima capacità del cluster. In Hadoop 2.x e Spark si è cercato di colmare questa problematica. Un altro svantaggio è che ad ogni iterazione di Map e Reduce il file viene scritto sul disco che rallenta tutto il processo. Inoltre non tutti lavori possono essere scritti in alberi di operazioni Map e Reduce, ad es. i cicli sui dati stessi.

8.2 Spark Ecosystem

Qua entra in gioco un'applicazione open source : esso effettua i calcoli in memoria senza riscriverli sul disco se non necessario e permette di generalizzare i calcoli usando grafi aciclici ottenendo molta più flessibilità. Allarga il set delle operazioni da solo Map e Reduce in **trasformazioni** e **azioni**, dove trasformazioni e azioni sono un set di operazioni diverse che possono essere

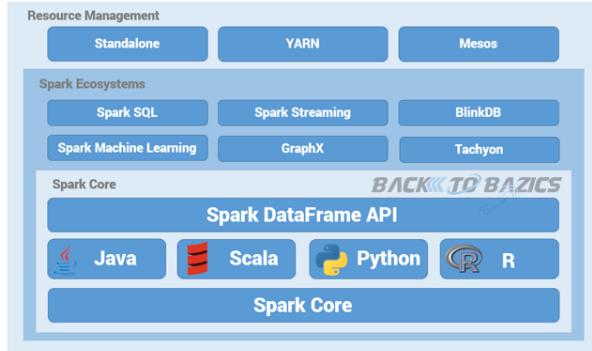


Figura 5: Spark Ecosystem

combinata in maniera arbitraria e Spark 2 ha implementato anche un ottimizzatore di query (Catalyst).

Apache Spark supporta:

- Manipolare i dati attraverso SQL, vedendo i file come delle tabelle SQL;
- Esiste una libreria MLlib per Data Science e Machine Learnig distribuito;
- Processamento di grafi di dimensioni enormi;
- Processamento di dati in tempo reale;
- File system Distribuito in-memory(cache);
- Query approssimate sui dati;
- Eredita da Hadoop il supporto dei file Avro, Parquet e le sorgenti di questi dati possono essere HDFS, Cassandra, Hive, Mongo ecc.;
- Sviluppo in linguaggi come Python, R, Scala, Java, .Net;

Al centro di tutto l'ecosistema c'è il Spark Core che si occupa di:

- Gestione del calcolo distribuito, quindi si occupa della distribuzione, coordinamento e scheduling dei task computazionali e inoltre si occupa di gestire i fallimenti e riduce al minimo la data shuffling tra i vari nodi;
- Fornire le API per l'astrazione delle **RDD**,

che è un collezione di oggetti, immutabili e fault tollerant partizionati su un cluster, che possono essere manipolati in parallelo.

Spark può essere depoyato in 3 modalità diverse:

1. Come una libreria che viene eseguita in un programma, fornendo le API delle RDD, ma si possono usare solo le risorse della macchina su cui gira il programma;
2. Può essere eseguito in maniera distribuita (su cluster), che può essere standalone (che non significa solo una macchina ma semplicemente il fatto che non ha bisogno di un scheduler esterno) oppure usando scheduler esterni tra cui YARN, Mesos e K8;
3. In modalità MR, in questo caso l'accesso ai sistemi di storage attraverso le API di Hadoop per usare HBase, S3, Cassandra ecc.

L'applicazioni Spark ha due elementi fondamentali più un terzo facoltativo:

1. **Spark Driver** che è la parte del codice che viene eseguita non in parallelo, questo driver crea uno SparkContext per gestire i jobs, lo SparkContext deve capire quali pezzi del nostro codice sono parallelizzabile e deve inviarli al cluster;
2. Nei vari cluster ci sono degli agenti **Executor** che eseguono gli ordini dal SparkContext e riescono a gestire una serie di task in parallelo, una volta concluso il task il risultato viene ri-inviato al driver;
3. Nel mezzo può esserci un **Cluster Manager** per permettere la comunicazione tra il driver e i vari nodi e di allocare le risorse.

Il context era un oggetto che creava delle connessioni ad es. per connettere SQL si creava SQLContext, per connettere Hive si creata HiveContext, ciò creava confusione e rendeva difficile una loro collaborazione. Questi diversi contesti, in Spark 2, vengono unificati in un unico oggetto

chiamato SparkSession, ottenendo un entrypoint unificato per gestire i dati in spark.

8.3 RDD

RDD (Resilient Distributed Dataset), essenzialmente una lista di oggetti distribuita e resiliente, distribuita nel senso che sono partizionati (in base a una chiave) in memoria di vari nodi ma viene vista come un'unica lista grazie alla chiave di partizionamento ciascuna partizione contiene un unico record che può essere operato indipendentemente (simile all'approccio Shared Nothing), ciò permettere la loro gestione in maniera parallela con le trasformazioni (Map, Filter) e in caso di fallimento vengono automaticamente ricostruiti.

Vengono di solito immagazzinati in memoria ma vi è la possibilità di scriverli sul disco, sono tipizzati (Integer, double, Objects), sono immutabili (quindi applicare una trasformazione significa creare un altro RDD) e sono lazy evaluated. Alcune possibili trasformazioni sono: map, filter, union, intersection, distinct e join (solo per RDD di tipo key-value).

Ciascun RDD tiene traccia delle trasformazioni usate per costruirlo, questa traccia è detta lineage, che può essere usato per ricostruire i dati persi. Le Action possono prendere un RDD e produrre un altro oggetto non RDD. Le trasformazioni sono lazy e non vengono eseguite finché non c'è un azione su RDD che li convoca.

Il Driver pianifica l'ordine di esecuzione convertendo le trasformazioni e azioni in un grafo aciclico diretto (DAG), queste DAG tracciano le dipendenze (Lineage) e i nodi del grafo sono RDD mentre gli archi rappresentano le trasformazioni. Le azioni sono il pezzo finale del grafo e triggeranno l'esecuzione della DAG, un'azione per esempio può essere reduce, count, collect, save.

Attenzione se io devo effettuare due azioni sullo stesso DAG, le trasformazioni del DAG vengono eseguite 2 volte, per evitare tale problema si può inserire il comando cache prima dell'esecuzione delle azioni. Alcune trasformazioni, come sortByKey, join, groupByKey ecc., possono avere uno shuffle che può rallentare il sistema, le trasformazioni possono essere:

- Narrow dependency: Ciascuna RDD viene consumato da al più una partizione figlio ma un figlio può avere molteplici genitori, quindi non cambia il numero di partizioni tra una trasformazione e l'altra, e non avviene lo shuffle;
- Wide dependency: Ciascun RDD può essere usato da molteplici partizioni figli, quindi cambia il numero di partizioni

Spark fa questa distinzione perché eseguirà le operazione Narrow tutte insieme in parallelo, spark divide in lavoro in stage, in base alla presenza di operazioni shuffle, all'interno di ciascun stage le operazioni possono essere eseguite in parallelo, inoltre le trasformazioni nei vari stage vengono composte per velocizzare il tutto.

I vantaggi di RDD sono il controllo a basso livello, grazie alle API, su cosa fa spark e sono inoltre typesafe: spark ci permette di fare operazioni in base al tipo di RDD e infine ci insegnano in profondità come funzione spark.

I suoi svantaggi sono il fatto che sono difficile da gestire (per un data scientist che deve usare i DataFrames), inoltre cambia la velocità di esecuzione anche in base al linguaggio usato (usando Java/Scala si ottiene un boost di performance anche 2x rispetto a python) e infine l'ottimizzazione del codice non avviene e quindi programmi sono meno efficienti.

8.4 Spark SQL

E' una libreria che funziona su Spark Core e fornisce come astrazione del RDD il dataframe, con tutte le proprietà dei dataframe, sono delle tabelle con righe e colonne con i header e indici ecc. A questo punto si può usare la tecnologia molto sviluppata dei database SQL, Spark SQL ha implementato anche un ottimizzatore che viene usato da tanti motori SQL. Questo aumenta anche il supporto alle varie tipologie di file e sorgenti con cui spark può comunicare.

I Dataframe essendo un'astrazione delle RDD hanno le sue proprietà: sono immutabili, distribuiti, rappresentano una collezione, sono lazily evaluated e le azioni sono eagerly evaluated.

Il risultato di usare i dataframe è non solo la stessa velocità indipendentemente dal linguaggio scelto ma addirittura un miglioramento rispetto a lavorare direttamente su RDD, questo anche perché i DataFrame sono compressi quindi usano meno memoria, inoltre viene tenuto in conto anche l'inefficienza dei programmatore mediamente parlando.

9 Stream Processing

La differenza tra il batch processing e stream processing è:

Batch Processing lavora su batch di dati limitati ottenuti da un data store e il risultato è un altro batch di dati. Batch ha accesso a tutti i dati, si può lavorare a qualcosa di grande e complesso, la latenza è misurata in minuti, di solito uno è interessato all'output (qualità) di dati che alla latenza.

Stream Processing lavora su un flusso continuo (infinito) di dati, il risultato è un nuovo

stream di dati e per ciò è dovuto al fatto che il processo stream non finisce. La funzione in questo caso è applicata ad un datapoint o una finestra piccola di dati recenti, inoltre la computazione deve essere leggera poiché lo stream deve continuare ad andare e la latenza deve essere near real time o al più secondi.

Gli elementi principali di un'applicazione stream sono:

1. Sorgente: che produce i dati in input;
2. Trasformazioni: prende l'input del dataflow e producono alcuni output;
3. Sink: riceve/consuma i dati dell'output delle trasformazioni;
4. Data Pipeline: è una sequenza di trasformazioni, rappresentate di solito da un grafo aciclico diretto;
5. Event/Tuple/message: E' l'elemento atomico del data flow (come una riga di un dataframe).

Generalmente i sistemi di stream di dati sono generalmente distribuiti, per gestire più dati in parallelo e quindi con maggiore velocità, ma avvolte vi sono delle operazioni sequenziali e si fa fatica a renderli distribuiti. I dati in ingresso possono essere partizionati e ciascuna operatore può eseguirsi in una maniera concorrente. Ottiene le proprietà dei sistemi distribuiti tra quali: Fault-tolerance, Load Balancing.

Lo stato dello streaming può essere molto utile, ad esempio per riconoscere qualche pattern negli eventi o per aggregazioni di dati, e lo stato di fault tolerance viene gestito in utilizzando i due meccanismi di **Checkpoint and Replay**: salvataggio periodico dello stato dell'operatore su un storage sicuro (es. HDFS), in caso di fallimento tutti i dati dopo l'ultimo checkpoint dovranno essere rianalizzati (Replay).

Nel caso di dati "illimitati" in streaming di solito si preferisce usare il windowing per tagliare

i dati dando a loro il senso temporale. Queste finestre possono essere:

- Fixed (Tumbling) window es. calcolo di qualcosa ogni tot unità temporale
- Sliding window es. calcola di qualcosa in un'unità temporale ogni tot unità temporale (possibilmente diversa da quella del calcolo)
- Session window, la sessione è definita come il periodo di attività terminato da un periodo di inattività più grande di una certa soglia, quindi sono finestre dinamiche e sono totalmente data driven, anzi sono molto compatibili con un sistema distribuito e ciascuna partizione avrà la sua finestra.

Esistono anche altri metodi di approssimazione, oltre a windowing, per gestire dati illimitati ad es. sampling con streaming K-means ecc., essendo delle approssimazioni vi sono degli errori. In generale sono algoritmi molto complessi poiché devono anche garantire un limite superiore degli errori.

SQL on Streams consiste nel vedere uno stream di dati come una tabella relazionale di dimensione infinita, a questo punto si può usare SQL per generare un'analisi o trasformazione, il sistema ci mostra una tabella (dinamica) ma sotto ha un sistema di streaming distribuito che genera a partire da uno streaming in entrata uno streaming in uscita.

9.1 Apache Storm

 **STORM** è disegnato per supportare lo stream di applicazioni e supporta fino ad 1 milione di messaggi al secondo, può scalare a miglia di nodi per cluster ed è fault tollerant e usando strumenti di terze parti, in particolare Trident è in grado di applicare anche la semantica temporale “exactly once”.

Il modello concettuale è un modello molto semplice ed include:

- Tupla: è l'unità core (il messaggio) e può essere vista come una coppia chiave valore;
- Stream: è la sequenza infinita delle tuple;
- Spout: è la sorgente dello stream e il suo compito è quelli di emettere delle tuple;
- Bolt: riceve le tuple, fa una certa computazione (può anche leggere e scrivere dati da uno store) ed optionalmente emette altre tuple.

La topologia è un DAG di spouts e bolts, nella quale ciascun spout/bolt esegue solo un task, quindi durante la definizione della topologia bisogna specificare come i spout e bolt sono tra loro collegati, il partizionamento dei dati dello stream fra i bolt è detto **stream grouping**, i vari tipi di stream grouping sono:

- Shuffle: il partizionamento è randomico, i spout mandano in maniera randomica i dati verso i bolts a cui sono collegati;
- Fields: il partizionamento è basato su un campo della tupla, può essere utile ad esempio nel conteggio di tuple con un certo valore in un campo;
- All: ciascun bolt riceve una istanza di una tupla;
- Custom;
- Direct: già la sorgente decide a quale bolt mandare la tupla;
- Global: Tutte le tuple generate da tutte le istanze di una sorgente vengono mandate a un unico target.

Il cluster totale è gestito da Nimbus che calcola gli assignment e li invia allo ZooKeeper e lo ZooKeeper li invia ai vari supervisor sui vari cluster che scaricano la topologia da Nimbus, questa topologia viene mandata in esecuzione attraverso un processo Java. Lo ZooKeeper si preoccupa anche di controllare la salute dei vari nodi con

HeartBeat e se un Worker muore, il supervisore lo riavvia e se continua a morire Nimbus assegna il worker (che può contenere più di un operatore) a un altro supervisor, mentre se muore tutto il nodo il lavoro viene assegnato ad un altro nodo. Se dovesse morire Nimbus, i supervisor continuano a lavorare ma il riassegnamento diventa impossibile, quindi conviene mettere Nimbus con alta disponibilità (magari tenerne uno di backup), mentre Zookeeper è fault tollerant per se.

I processi che garantiscono l'affidabilità (ad es. che una tupla venga per forza vista) sono:

- Ack and Fail: Ciascuna tupla generata ha un unico ID e ciascuna tupla processata deve essere ackata e fallita, in caso di processamento completato viene generato un ack e fail in caso di timeout, il messaggio viene tracciato da un task detto “acker” in caso di fallimento lo spout può far ripartire il messaggio se è programmato in tale modo;
- Anchoring: Un bolt può emettere una nuova tupla ancorata alla tupla input, quindi nel caso del fallimento lo spout tuple alla radice del DAG andrà rieseguito.

Esiste una libreria esterna Trident che permette di scrivere ad alto livello anche architetture molto complesse, mentre rimane stateful senza che sia l'utente a basso livello a definirla, permette di lavorare al livello micro batch, cioè non lavora ogni singolo messaggio che gli arriva ma accoppi un numero di messaggi, questo migliora il throughput ma peggiora la latenza. Riesce a passare da at least one di storm a exactly once.

9.2 Spark Streaming

Spark è fatto per funzionare con l'idea del batch, per implementare lo streaming sfrutta il così detto mini-batch, quindi si aumenta la latenza ma si

alza anche il throughput. Spark streaming quindi porta con sé tutte le proprietà già esistenti in spark tra cui: distributività, disponibilità e scalabilità. Il funzionamento inizia con i receiver che ricevono e emettono batch che in spark diventano RDD.

Grazie a spark streaming la creazione di un'architettura lambda avviene non solo nello stesso linguaggio ma addirittura nello stesso programma.

Alla base di spark ci sono gli RDD che possono essere distribuiti e replicati in memoria, quindi in caso di un problema di un nodo non perdo i dati, e grazie alla lineage posso anche capire da quale punto si dovrebbe far ripartire la computazione (a differenza di storm che deve far ripartire tutto). I minibatch discretizzati RDD si chiamano **Dstreams**, la loro misura è fatta in modo da avere una latenza di circa 1 secondo e hanno un potenziale di potersi combinare con i batch proccesing. Inoltre con la funzione union è stato possibile implementare il concetto di finestra.

Il master continua a salvare gli stati dei Dstream in un file checkpoint e in caso di fallimento del master questo può essere reinizializzato a partire da questo file checkpoint, permettendo un recupero rapido.

I Dstream richiedono una buona conoscenza di programmazione, quindi si è creata una sua astrazione sotto il nome di Spark Structured Streaming, che ha delle API a livello più alto rispetto a Dstreams. I Dstreams mostrano lo streaming come delle tabelle dinamiche, mentre le operazioni vengono eseguite incrementalmente, quindi il risultato è un altro dataframe con dati variabili nel tempo. L'analisi dei dati provenienti dal Output mode viene eseguita in base a un trigger: che può essere una funzione della sessione temporale.

9.3 Amazon Big Data

Amazon è il leader nel cloud computing, e offre un sacco di servizi direttamente lui pronti spesso anche per la scalabilità quindi non vi è neanche il bisogno di configurarli o mantenerli, e se un servizio non è direttamente offerto, è molto probabile che venga offerto nel marketplace.

La Big Data pipeline è costituita a un livello alto da:

- Data Ingestion/Data collection;
- Data Storage;
- Processare e analizzare i dati;
- Consumare o visualizzare.

e il suo obiettivo è minimizzare la latenza e il costo e massimizzare il throughput, nel cloud cerco di minimizzare la latenza ad esempio raggruppando le macchine vicine e inoltre massimizzo il throughput grazie alla scalabilità dello storage e tutte le macchine leggono alla stessa velocità quindi la velocità di lettura cresce linearmente col crescere dei nodi e per quanto riguarda i costi essi vengono minimizzati poiché pago solo ciò che uso. Il problema rimane il passaggio totale al cloud, ad esempio perché può essere costoso trasmettere tutti i dati, oppure potrebbero esserci problemi di privacy, esistono anche degli approcci ibridi in cui è possibile usare il proprio cloud per dati privati e Amazon per dati normali.

Lo stream storage di amazon è possibile con: Apache Kafka (esiste una cosa simile anche in amazon detta Amazon MQ), Amazon Kinesis e Amazon DynamoDB (simile a SQL streaming di spark). I file storage offerti possono essere HDFS, S3 e Amazon Glacier, sono tutte e tre supportati da Big Data Framework e sono altamente disponibili, la scelta avviene se i dati sono caldi (vengono acceduti molte volte) qua conviene usare HDFS o se sono freddi (pochi accessi) qua conviene Glacier, mentre S3 è una via di

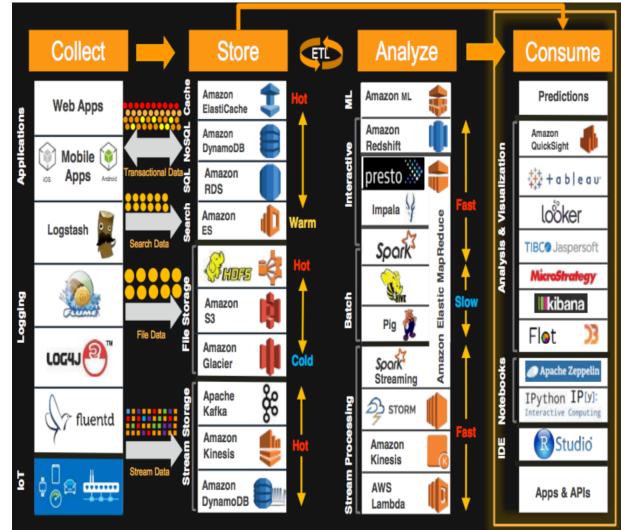


Figura 6: Insight of the Amazon Services

mezzo. Infine per i database e search engines abbiamo: Amazon ElastiCache, DynamoDB, RDS e ES.

Part II - Prof Melen

10 Introduzione

L'Iot è un infrastruttura globale per la società (informatica), che crea servizi avanzati attraverso interconnessione di cose basate su tecnologie informatiche e di comunicazioni interoperabili. Per il funzionamento dell'IoT la comunicazione Machine2Machine è essenziale.

Gli ingredienti principali quindi sono: Smart things, tanti oggetti intelligenti, applicazioni autonome e tecnologie specializzate.

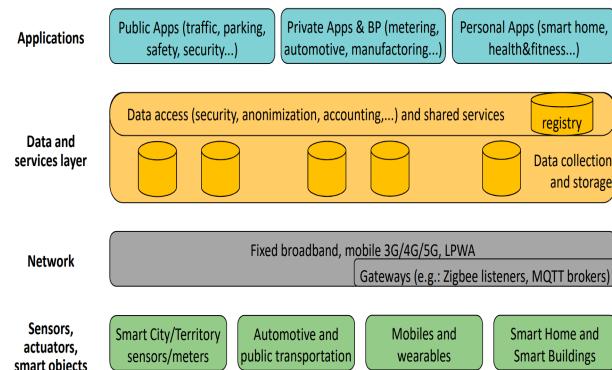


Figura 7: Uno dei possibili modelli a strati

Dalla Figura 7 si evince subito che le tecnologie abilitanti per IoT sono:

- Sensoristica;
- Connattività e Protocolli;
- Software e Algoritmi specializzate per IoT.

I settori dove queste tecnologie sono state sviluppate sono: automotiva, logistica, manifatturazione, energia e utilità, healthcare, gestione e costruzione delle case, settori pubblici come illuminazione ecc. e settore militare.

10.1 Smart Metering

Una delle prime cose che osserviamo è che un servizio di metering può avere la necessità di non essere alimentato con una rete elettrica (es. contatore del Gas) quindi si usano batterie, e se essa si esaurisce bisogna andare a sostiturla e questa sostituzione costa di più di un controllo manuale del contatore.

I contatori di Enel oggi hanno 2 catene di distribuzione dei dati:

1. Chain 1: va verso l'operatore che gestisce la rete, per i dati ufficiali della tariffazione;
2. Chain 2: va verso il cliente finali o venditori, per permettere la gestione delle offerte ecc.

L'idea dello smart metering serve per arrivare alla fine alla smart grid, questo serve poiché alcuni centrali per la produzione dell'energia hanno una produzione fissa es. centrali a combustione, ma altri hanno dei picchi e vuoti es. centrali eoliche o solari. Le centrali sono molto poco adatte ad immagazzinare energia, un metodo potrebbe essere offrire dei sconti al cliente per consumare energia in una certa fascia es. se ricarica macchina elettrica in una certa fascia orario ti costa di meno, ottimizzando così il consumo.

11 Sensori

Un circuito basilare è costituito da un generatore e un impedenza (es. resistenza, condensatore, induttanza), la corrente che passa attraverso questa impedenza genera una differenza di potenziale quindi un certo livello di tensione.

Usando i circuiti si possono misurare i fenomeni fisici, sfruttando o la tensione sull'impedenza (usando un voltmetro) o la corrente che passa attraverso il circuito (usando un amperometro). Le componenti del circuito possono cambiare a seconda di un fenomeno fisico, e questi cambia-

menti si possono misurare in base al cambiamento che apportano al circuito, ottenendo così un sensore. Esempio un circuito può cambiare resistenza in base alla temperatura, quindi cambia il voltaggio se la corrente rimane costante (dalla legge di Ohm $V = RI$).

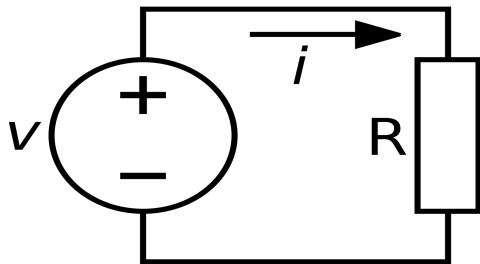


Figura 8: Esempio di un circuito semplice.

Alcuni esempi di fenomeni fisici che vogliamo misurare sono:

- Ambiente: Temperatura, umidità, pioggia, rumore, inquinamento;
- Veicoli: Posizione, velocità, parametri dei motori per la manutenzione, pneumatici, carburante;
- Smart Phones/Wearables: posizione, parametri vitali;
- Edifici: Metering, Sicurezza.

I sensori di solito si basano su un elemento sensibile e sfrutta le sue proprietà fisiche per effettuare la misura e quindi un sensore fatto per un certo ambiente non è detto che funzioni su un altro ambiente es. il termometro per misurare la temperatura corporea non può essere usato per misurare la temperatura del sole.

La pipeline della conversione in dati di un sensore è un po' complessa: La pipeline nella Figura

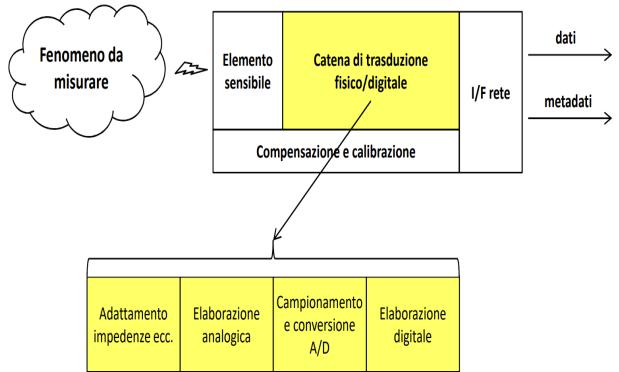


Figura 9: La pipeline/struttura di un sistema di sensore.

9 inizia con una conversione analog2analog che serve per l'amplificazione del segnale, che viene seguita da un elaborazione analogica per cercare di togliere errori/disturbi dopo questa fase il segnale viene convertito in digitale usando il campionamento (usando il teorema di Niels) e quantizzazione che introduce un po' di errore che serve per la fase della digital processing. Il vantaggio dell'elaborazione digitale è che è meno costosa computazionalmente rispetto all'elaborazione del segnale analogico, inoltre o funziona o non funziona mentre quello analogico in caso di errore funziona a metà creato dei problemi più gravi. In parallelo a questa catena di traduzione effetto fisico-digitale si hanno delle funzioni che servono per la calibrazioni e compensazioni. In uscita ho un Network Interface che manda i dati e metadati all'esterno.

Gli output di un sensore dipendono da:

- struttura dei dati:
 - Dati Binari, discreti o “continui”;
 - Valori Singoli o una Serie temporale;
 - Scalare o Vettoriale.
- Timing, che può essere:
 - Eventuale - è asincrono, in caso di un

- evento invia un segnale, es. gomma della macchina bucata;
- Periodico - ogni tot tempo invia un segnale;
 - On-Demand - Il segnale viene inviata solo in caso di una richiesta.

Il sensore può anche fare il preprocessing, es. può riempire buchi nei segnali attraverso interpolazioni e può fornire anche i metadati: i metadati sono la posizione, tempo e affidabilità. Attenzione, invio di dati in maniera periodica non garantisce sia esattamente periodico, es. il tempo del calcolo può avere fluttuazioni temporali.

11.1 Rapporto misurando e principi fisici

Un misurando può essere misurato usando l'effetto di tanti fenomeni fisici. Per esempio nel caso di temperatura alcuni sensori si basano su:

- Espansione Termica;
- Variazioni di resistenza elettrica;
- Tensione di polarizzazione nei diodi: i diodi fanno passare la corrente se c'è una certa tensione di polarizzazione ai capi e questo threshold dipende dalla temperatura;
- Effetto Seebeck: è una differenza di tensioni fra due conduttori di due leghe differenti che si ha quando i due conduttori vengono posti a una certa temperatura;
- Radiazioni Infrarosse.

Esempi di circuiti elettrici:

Nel Primo caso (Figura 10) il circuito può aprirsi o chiudersi in base alla temperatura, mentre nel secondo caso in base alla temperatura il circuito cambia il circuito secondario a cui è collegato. Questo sistema appena spiegato ha lo svantaggio di non essere integrabili in componenti piccole.

Termostati bimetallici (on/off)

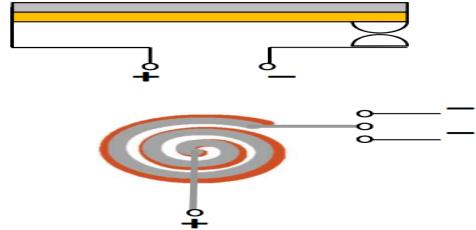


Figura 10: Sensori di Temperatura On/Off basati sull'espansione termica.

Termistore con resistore a coefficiente negativo (NTC)

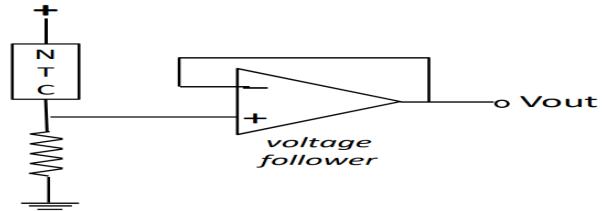


Figura 11: Sensori di Temperatura basato su termistore.

Il secondo esempio (Figura 11) che introduciamo si basa su un termistore con resistore a coefficiente negativo, chiamiamo V_1 il potenziale tra i due vertici del NTC e V_2 il potenziale tra i due vertici della resistenza, se la temperatura cresce V_2 aumenta (segue dal fatto che $V_1 + V_2 = V$ non cambia). Questo valore viene portato in un amplificatore detto voltage follower, questo follower serve anche da barriera perché la componente alla sinistra è molto sensibile e non deve essere perturbato in nessun modo, inoltre ha il vantaggio di avere dimensioni molto piccole e quindi di entrare anche dentro componenti molto piccole. Un altro vantaggio è che i dispositivi a coefficiente negativo costano poco, poiché si potrebbe

usare benissimo anche un resistore con coefficiente positivo ma costano molto di più.

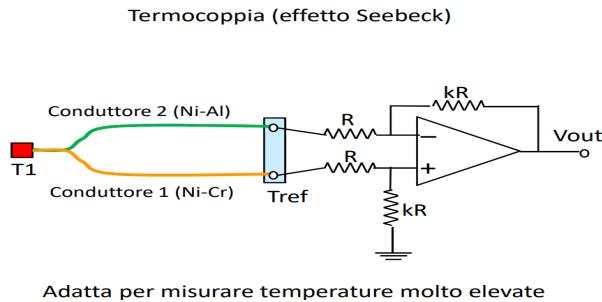


Figura 12: Termocoppia con effetto Seebeck.

Si può usare anche la termocoppia che è un dispositivo che si basa sull'effetto seebeck che usa delle coppie leghe metalliche (speciali) diversi (Figura 12), scaldando un endpoint collegato a due fili di queste due leghe metalliche genera una tensione tra i due fili anche a una distanza molto lunga, quindi si possono misurare temperature di certi ambienti dove un circuito finirebbe per fondersi.

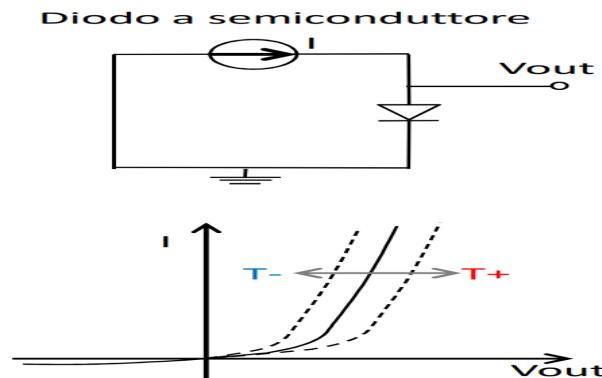


Figura 13: Circuito basato sul Diodo.

Un altro esempio è quello che sfrutta i diodi (Figura 13), supponiamo l'esistenza di un generatore di corrente costante e un diodo sensibile alla

temperatura, misuro il potenziale, e si nota che il potenziale (data la stessa corrente) cambia in base alla temperatura. Per misurare invece la

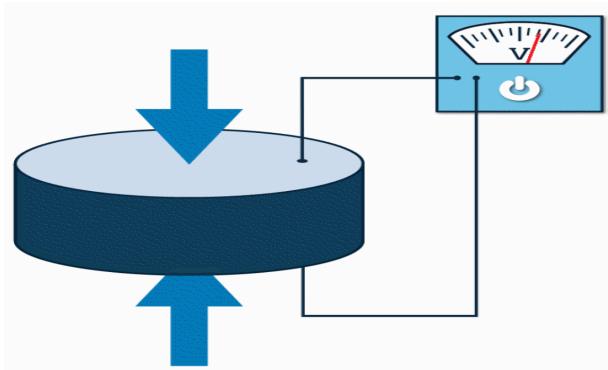


Figura 14: Effetto Piezoelettrico.

pressione di solito si usa l'effetto piezoelettrico, nel quale comprimo lungo un certo asse un cristallo (che deve essere non centro simmetrico), e questa compressione genera un impulso elettrico (Figura 14), questo materiale misura non l'intensità della pressione ma solo la variazione quindi può misurare le vibrazioni.

Esiste anche l'effetto piezoresistivo, che è un conduttore costituito da strati un materiale piezoresistivo es. Silicio (speciale), applicando pressione ottengo un impedenza che varia le sue caratteristiche a seconda della pressione, ottengo quindi un misuratore di pressione, che a differenza dei materiali piezoelettrico misura anche la pressione costante. Bisogna tenere conto che i valori della variazione sono molto bassi quindi si usano dei circuiti ponte che fanno da amplificatore (Figura 15).

Vi sono anche dei circuiti che misurano lo spostamento fisico di una membrana (ad es. per misurare la pressione dell'atmosfera), di solito vi è un pezzo ferro magnetico collegato alla membrana e il suo movimento affligge il campo magnet-

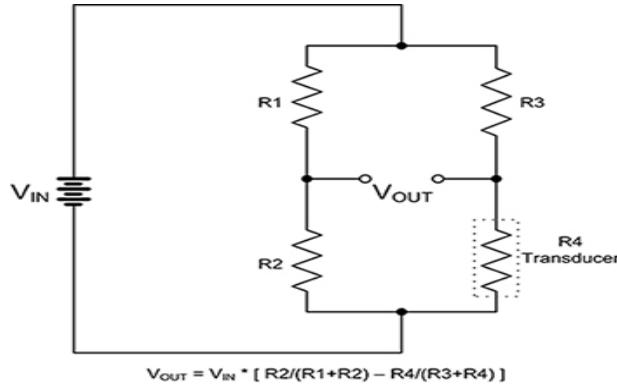


Figura 15: Esempio di un amplificatore di segnale.

tico, quindi dalla varianza del campo magnetico riesco a capire la pressione ma purtroppo non è molto precisa come misurazione.

Un modo meno naive di misurare la pressione di una membrana è sfruttare l'effetto Hall (Figura 16), nel quale un magnete è collegato alla membrana e il suo movimento affligge la differenza di potenziale su un materiale conduttore inserito in un circuito in cui gira una corrente, calcolando la variazione del potenziale (creata dal cambiamento del percorso degli elettroni) si può calcolare il potenziale. L'effetto Hall è molto più generico e può funzionare in molti altri casi in cui bisogna rilevare un movimento fisico di una componente.

Una delle categorie molto importanti dei sensori di pressione sono i sensori di ruomore es. microfono.

Uno dei sensori di prossimità più usati sono i sensori di touch screen che non si basa sulla pressione, sotto lo schermo vi sono degli elettrodi, e generano dei campi magnetici, e il touch è la variazione del campo magnetico (cambio la capacità, infatti è detto capacitive sensor).

I sensori di prossimità possono essere:

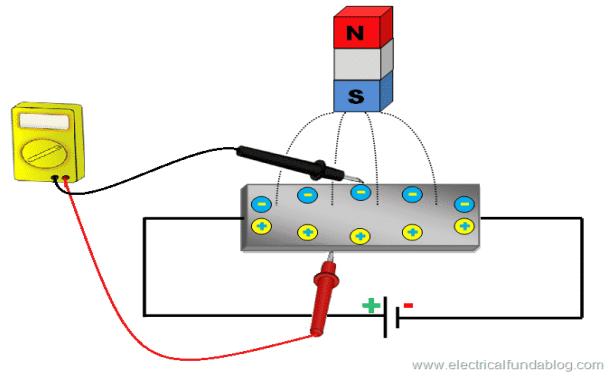


Figura 16: Effetto Hall.

- Attivo: Si basano essenzialmente su ultrasuoni invio un ultrasuono e una perturbazione/variazione della sua risposta indica un movimento, altri meccanismi attivi sono basati su laser, nel caso di laser le risposte e variazioni sono estremamente precise, che possono indicare anche la distanza dell'oggetto;
- Passivo: Si basano principalmente su infrarossi.

Un altro settore molto importante della sensoristica è quella riguardante il movimento, quindi ho bisogno di conoscere l'accelerazioni e rotazioni nei 3 assi. IMU (Interia Measurement Unit) è un dispositivo in grado di darmi informazioni riguardanti l'accelerazione e rotazione nei 3 assi, che quindi misurano la forza che si esercita su una massa nelle tre direzioni ortogonali.

I MEMS (Micro-Electro-Mechanical-Systems) sono dei sistemi che scalano, su dimensioni estremamente ridotte, i principi dei sensori macroscopici, e normalmente sono integrabili in maniera particolarmente efficace.

Il sensore dell'accelerazione è fatto come nella Figura 17, costituito da 3 dischi dove il disco A e B ricevono corrente alternata e nel caso del

movimento del disco C cambia la capacità del condensatore formato e quindi il potenziale e misurando la variazione del potenziale si ottiene la forza di accelerazione in una certa direzione.

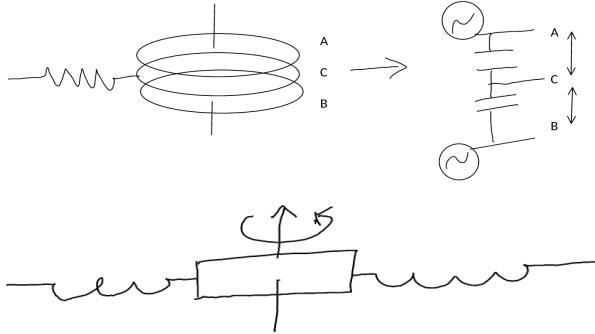


Figura 17: Sensore per misurare l'accelerazione e rotazione.

Il sensore di rotazione non è un giroscopio poiché è di difficile implementazione. Il sensore è formato da una massa sospesa da due molle come nella Figura 17, se provo a farlo ruotare lungo il suo asse ortogonale, mentre è in oscillazione, lui genera una forza opposta. Siccome deve oscillare mi serve un materiale piezoelettrico a cui fornisco corrente elettrica alternata. A questo punto misurando queste forze riesco a misurare la coppia di forza con sensori di pressione. Mi servono 3 di questi sensori per ciascuna direzione.

11.2 Sensore Ideale e Reale

Nel sensore ideale dato il fenomeno X, che assume valori tra 0 e X_{\max} , l'output varia linearmente in maniera continua con X. Nel sensore reale è molto diverso dal sensore ideale:

- Lo zero non esiste;
- La risposta non è lineare con X;
- La misurazione ha livelli di sensibilità quindi certe variazioni dell'input non vengono per-

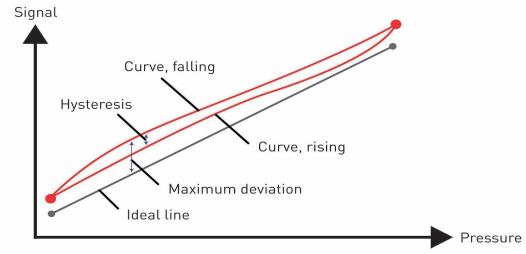


Figura 18: Differenza Segnale Reale ed Ideale.

cepite dal sensore (si può pensare un grafico fatto a gradini in base alla sensibilità);

- La linea della variazione non è unica, è influenzata da fattori esterni es. temperatura;
- esistono drift temporali, quindi la stessa misura può cambiare nel tempo.

Visto che sono troppi problemi si prova ad eliminare alcuni usando processi di compensazione e calibrazione, dopo aver corretto il più possibile rimane comunque imperfetto ma bisogna sapere quanto non si è corretto.

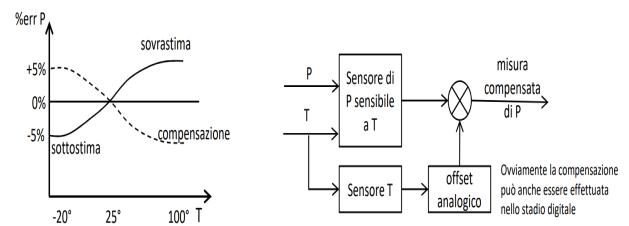


Figura 19: Es. di una compensazione

Assumiamo che ci sia una variazione della misura in base alla temperatura come indicato nella Figura 19, allora creo una variazione opposta a quella del disturbo ottenendo una misura compensata, chiaramente la compensazione non po-

trà essere identica al disturbo, di solito viene accettato sotto un livello di variabilità.

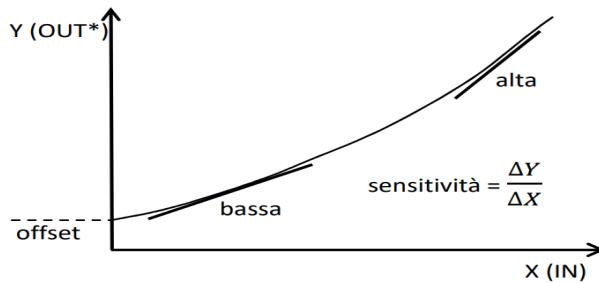


Figura 20: Errore prima della Calibrazione.

La calibrazione serve per rendere la caratteristica lineare, ciò può essere effettuato attraverso una trasformazione matematica che può essere anche a livello analogico ma è molto più facile a livello digitale.

11.2.1 Caratteristiche dei Sensori

Le caratteristiche dei sensori possono essere statiche e dinamiche.

Le caratteristiche statiche sono le caratteristiche che non dipendono dal tempo o dipendono debolmente dal tempo e sono:

1. Accuratezza: Scarto percentuale fra la misura e il valore reale, intuitivamente l'errore;
2. Precisione: La varianza delle misure (dello stesso ambiente), quanto sono vicine le misurazioni, intuitivamente il Rumore;
3. Rumore: Fluttuazione della misura (sia interno che esterno);
4. Stabilità nel tempo: Assenza di drift nel tempo, cioè il comportamento del sensore non cambia nel tempo;

5. Risoluzione: Minimo valore di variazione dell'input che produce variazione dell'output attenzione non è per forza identico al numero delle cifre decimale che lo strumento dà, ma non le cifre significative non possono essere superiori alla risoluzione;
6. Selettività: Insensibilità del sensore alle interferenze;
7. Campo di misura: Il campo può essere (di solito lo è) più ristretto rispetto al range della quantità misurata (termometro con Hg non può misurare $-157^\circ C$), posso anche restringere il campo laddove la misura è più affidabile;
8. Isteresi: Differenza tra la misurazione in crescita e in decrescita, per un esempio grafico vedi Figura 18.

Le caratteristiche dinamiche sono le caratteristiche dipendenti dal tempo, quando inizio una misura devo aspettare un minimo prima della misurazione corrispondente. Assumiamo un sensore calibrato, pensando la X come un impulso, l'output si avvicina a X, l'avvicinamento può essere di ordine 0 (immediato, sistema ideale), ordine 1 in maniera esponenziale ($y = X(1 - e^{-t/\tau})$) e ordine 2 invece è più complesso ed è dato dalla soluzione di un'equazione differenziale del secondo ordine (Vedi Figura 21).

Nel primo ordine τ decide il tempo dopo il quale il segnale raggiunge il valore necessario, mentre nel caso del secondo ordine il segnale oscilla attorno al valore (in maniera periodica) prima di rientrare in una banda di variabilità accettabile.

Riassumendo le cause che rendono una misura imperfetta sono:

1. Intrinseche: poco accurato, non preciso, non compensato ecc.;
2. All'ambiente: ambiente rumoroso, con interferenze, si può risolvere con sensori più

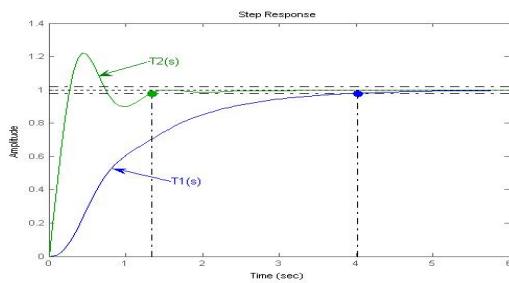


Figura 21: First vs Second order Response.

precise e costose o usando più sensore e combinare le misure da più sensore oppure isolando il sensore il più possibile;

3. Guasti: Danneggiamenti accidentali, sovraccarichi - problema è può mandare dati a caso;
4. Imprecisione nei metadati: es. impreciso nel dare l'istante di misurazione,
5. Rete: es. può non trasmettere dati per causa di un guasto.

12 Reti

Le reti sono a **pacchetto**: In questo caso l'informazione è fatta a blocchi con un etichetta detta header, che contiene l'informazione controllo. Questi reti usano il concetto di **protocollo**, che è un insieme di definizione che descrive il formato dei dati, sequenze logiche e infine i tempi (che serve in caso di timeout di un messaggio).

Questi protocolli a pacchetto sono fatti a **strati**: tutte le funzioni che fanno funzionare la rete sono tante e complicate, è estremamente difficile avere una visione completa di tutto, se tutto fosse scritto in un unico blocco, correggere i bug diventerebbe impossibile e i test diventerebbero enorme. Il livello più basso è quello fisico, la stratificazione della rete può essere vista (in maniera

molto alta) come:

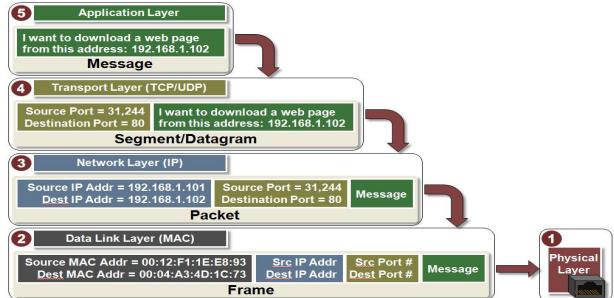


Figura 22: Strati della rete.

Il livello più basso lavora su bit, il data link permette di definire i pacchetti, lo strato della rete vede la rete nel suo complesso quindi riesce a capire i percorsi migliori ecc. Il trasporto invece virtualizza la rete per renderla più affidabile, l'applicazione invece serve per utilizzare la rete. I protocolli usati sono diversi quando si usa l'internet normale e l'internet per la sensoristica. Ad es. per internet si usa HTTP per le applicazioni, TCP o UDP per trasporto, IP per la Rete e Ethernet (802.3) o WiFi (802.11) per il Data Link, mentre per la sensoristica non posso spesso usare questi protocolli perché richiede molte risorse, ad esempio per Data Link si può usare BLE (Bluetooth Low Energy), ZigBee, W-MBUS (per le misurazioni), LoRa, PLC e per il trasporto si usa MTTP, CoAP, WebSockets.

Osserviamo che diversi protocolli implicano diverse strutture fisiche (e logiche), ad esempio le reti locali possono avere diverse strutture fisiche, infatti può essere un bus in cui la rete è in comune a tutti i terminali che prendono turni per parlare, la struttura fisica è associabile a quella di un cavo elettrico. Questa topologia è diversa da quella a stella nel quale ogni terminale parla solo con il Gateway.

Si ha una divisione in 2 macrocategorie per le

reti geografiche:

- **Diretto:** In questo caso il terminale deve essere più sofisticato per poter comunicare, avviene solo il semplice trasferimento di informazione in questo caso.
- **Gateway:** In questa situazione posso avere reti locali che usano protocolli diversi e il gateway si preoccupa di tradurli in protocolli standard, il gateway può fare anche cose in più es. salvare messaggi, risparmiare energia chiudendo alcuni canali non utilizzati ecc.

A livello locale invece si può dividere la rete nelle due famiglie: Wireline e Wireless.

12.1 Rete Wired

Una rete Wired è costituita da fili fisici che raggiungono il terminale, questi collegamenti possono essere in rame o fibra ottica.

Una delle famiglie più importante costituita dalle reti ethernet (per gli amici 802.3), queste reti nascono per le reti tra i calcolatori e l'obiettivo è sempre stato quello di costruire reti ad alte prestazioni per la comunicazione a grandi velocità e tempo di risposta molto elevate. Oggi le velocità arrivano a velocità anche molto elevate.

La rete ethernet non è pensata per i bassi consumi e il cablaggio costa molto perché i cavi sono di un qualità molto alta.

In uno dei tantissimi standard per l'ethernet vi è uno un pelino più speciale: **PoE**(Power Over Ethernet), se si ha ethernet organizzata sul rame, si può usare questo collegamento per trasmettere sia dati che potenza elettrica dai 25 ai 90W, questo mi permette con un unico cavo raggiungere un terminale collegandolo e alimentandolo. Un cavo ethernet in rame è fatto da doppini (coppie di fili di rame arrotolati), questi doppini sono molto simili ai cavi telefonici ma sono di un qua-

lità molto più elevata. I fili vengono fatti come doppini perché in questo modo diventa immune alle interferenze esterne, ma se vengono avvolti con la stessa frequenza riescono a parlare tra di loro molto bene (“auto interferenza”) quindi si usano passo di intrecciamento diverso.

Il PoE serve per fare la Power Line Communication **PLC**, il problema è che i fili sono antenne (non intrecciate) quindi si accoppiano molto facilmente, l’idea per risolverlo è molto semplice, si prende il collegamento già esistente in casa e un condensatore che fa passare alcune frequenze, ma questo non era facilmente realizzabile per moltissimo tempo perché richiede molta potenza calcolo (che oggi abbiamo) per il Digital Signal Process sopra il Power Line.

Quando si parla di Power Line ci sono due mondi diversi:

- **Narrow Band:** Sono protocolli che trasmettono pochi bit al secondo (5-10 kbps), che vanno benissimo per applicazioni di misurazione(metering), ha due canali A e C, che sono due bande riservate dove il canale A va da 3 a 95 kHz e il canale C va da 125-150 kHz. Questi due canali hanno utilizzo diverso il canale A viene usato per le applicazioni di misurazione utilizzata dal gestore per le fatturazioni mentre il canale C può essere utilizzato come canale secondario che va all’utente per diventare applicazione controllo. Questi canali vengono usati per collegare tanti contatori elettrici a un punto di raccolta dell’informazione delle misurazioni. Questo protocollo è governato da un accesso a contesa (CSMA/CA).

- **Broad Band:** ci sono degli standard che si chiamano home plug, che permettono di trasferire a velocità elevate, sono pensati per lavorare in contesto limitato. Sostanzialmente 3 versioni: 1.0 (velocità < 100Mbit/s), AV (< 200Mbit/s) e AV2 (< 1Gbit/s).

Un altro standard è **BACNET**: importante in uno specifico settore del riscaldamento e condizionamento aria, va dal livello fisico a quello logico/applicativo, a livello fisico assomiglia a un doppino telefonico.

12.2 Rete Wireless

Premessa: Segnale radio è molto diverso dal segnale fisico, ad esempio il segnale radio si attenua di più del segnale fisico, perché la potenza del segnale decresce come $\frac{1}{r^2}$, inoltre l'antenna può essere omnidirezionale o direzionale, e non posso utilizzare certi protocolli, ma grazie alle antenne direzionali riesco a bloccare le interferenze provenienti da altre direzioni, che è anche uno svantaggio perché devo sapere la direzione dove girare l'antenna, inoltre la capacità di attraversare gli ostacoli dipende dalla frequenza, le onde con frequenza bassa non hanno problemi ma quelli con alta frequenza sì, quindi maggiore velocità voglio maggiore sarà il disturbo che essa riceve.

Non si possono usare tutte le frequenze che si vogliono, le frequenze possono essere licenziate o non licenziate dallo stato. Le frequenze licenziate sono date in concessione ai concessionari come Radio, TV e Cellulari, essi pagano allo stato per ottenere il diritto esclusivo su una banda di frequenza tutelata per legge (un altro tizio non può usare la loro frequenza). Mentre le frequenze non licenziate sono libere ma non possono essere usate a caso, ma solo da dispositivi adeguati e certificati in base al paese nel quale sono. Es. le frequenze a 2.4 GHz che vengono usate anche dai dispositivi WiFi, ma non posso modificare/costruire uno strumento che usa questa frequenza perché il dispositivo deve essere certificato, poiché ci sono dei limiti di potenza che questi dispositivi possono avere, altri esempi di

frequenze non licenziate sono: 169 MHz e 868 MHz.

Abbiamo un altro problema: se trasmettiamo a 2.4 GHz ho canali molto grandi ($\approx 20\text{MHz}$) perché non si può trasmettere a una frequenza così precisa, mentre quando trasmetto a basse frequenze la larghezza dei canali si riduce moltissimo e quindi non posso avere HD.

I bit al secondo sono quanti bit trasmetto su una certa frequenza, nel caso di una frequenza di trasmissione proporzionale alla frequenza fondamentale dell'onda (di solito $2\times$) e qua anche in caso di perdita di potenza riesco a capire se il dato se era 1 o 0, se invece prova a trasmettere sulla stessa frequenza più informazione es. 11, 10, 01, 00 al posto di 1 e 0 in caso di perdita di potenza non riesco a capire l'informazione quindi aumento la velocità ma rischio di perdere informazione.

Protocolli della **Rete Wireless** sono Rete Locale Radio (**WiFi** per amici 802.11), è l'equivalente wireless di ethernet, cioè pensata per interconnettere terminali ad alte velocità. Il primo WiFi arrivava fino a 11 Mbit/s, l'ultimo standard di WiFi permette di raggiungere teoricamente 10 Gbit/s.

Non era pensato per un protocollo per risparmiare energia ma con la nascita degli smartphone si cambia il protocollo per consumare meno energia. Comunque il WiFi consuma tanto per essere usato insieme a sensori isolati alimentati a batteria, però in dispositivi come frigo, lavatrici che già consumano tanta energia, non hanno problemi ad avere sensori che comunicano usando WiFi.

Il WiFi è specificato a due frequenze 2.4 e 5 GHz. Un altro protocollo che funziona sempre su 2.4 GHz e adattato per il mondo della sensoristica è il **Bluetooth**. Ha un meccanismo di utilizzo delle risorse che è molto peculiare, al posto di

usare una banda di frequenze fissa ne utilizza 79 diverse ogni uno di questi a 1 MHz, e salta da una banda all'altra, questo tipo sistema è detto FHSS (Frequency Hopping Spread Spectrum). Il vantaggio è che il trasmettitore sta su una banda per un tempo molto piccolo riducendo gli effetti del rumore causati su una certa banda.

Di solito ogni applicazione che lo usa è molto specifica es. audio con cuffie o macchine, si possono anche comunicare dati generici quindi nel mondo della sensoristica. Ma per entrare in questo mondo della sensoristica usa un sotto standard detto Bluetooth Low Energy, consuma molto poca energia usando solo 40 canali a 2MHz che quindi trasmette 1 Mbit/s. Una delle applicazioni più importanti sono i così detti iBeacon, sono dei sistemi che irradiano intorno fornendo la sua posizione nello spazio, e quindi permettono di determinare la prossimità di un terminale da un certo posto, viene usato principalmente per delle applicazioni di proximity marketing (esempio IKEA), ultimamente viene usato anche per applicazioni domestiche.

Un altro protocollo molto importante è **ZigBee** (per amici 802.15.4), è una rete con organizzazione gerarchica nata per la domotica con velocità minore di 250 kbit/s, quindi per velocità basse.

Nella gerarchia vi sono due tipi di nodi: reduced functional (RF) e full functional (FF). Un nodo FF è un nodo che può raccogliere un numero anche molto grande di collegamenti con nodi RF, il nodo RF è nodo che comunica sporadicamente con il nodo FF quindi è una comunicazione di tipo asincrono e utilizza un meccanismo di comunicazione simile a WiFi (CSMA/CA) ma è diverso dal WiFi perché WiFi si aspetta periodicamente dei beacon, nel caso di ZigBee quando la stazione RF ha qualcosa da trasmettere lui trasmette, quindi può rimanere fuori comunicazione anche per periodi molto lunghi. Il CSMA/CA

funziona nel modo seguente quando devo trasmettere mi metto in ascolto e se il canale su cui devo trasmettere è libero, trasmetto, in caso di conflitto si entra in un backoff esponenziale randomico: rinvia il segnale dopo un numero casuale di secondi in un range pre-determinato, in caso di conflitti continui il range viene aumentato. Vi è la possibilità di usare oltre a CSMA/CA anche dei slot dedicati per ascolto, nel quale dopo un contatto viene garantito uno slot temporale per le successive connessioni.

ZigBee usa il canale a 2.4 GHz, tuttavia lo si può usare anche a 868 MHz e questo si riesce usando ZWave, di solito 868 MHz si usa solo per applicazioni locali es. apertura del cancello. Di solito l'architettura è a centro stella, il centro stella traduce dei protocolli e apre delle sessioni, quindi il centro stella può fungere da bridge quindi la comunicazione avviene dal RF al FF centro, ma deve avere degli indirizzi di rete, ma usare IP diventa pesante perché ogni header pesa 20 bytes, quindi si usa un protocollo IP semplificato: 6LoWPAN. Otteniamo due possibili configurazioni: se trasmettiamo dati al gateway che poi pensa a rimaneggiarli oppure i dati possono attraversare il gateway in maniera trasparente usando 6LoWPAN.

Wireless M-BUS è il protocollo più diffuso per il metering di cui quelli più importanti sono quelli del Gas, Acqua e Elettricità. Usa due bande di frequenze 868 MHz e 169 MHz (169 ha più penetrazione rispetto a 868), è pensato per il basso consumo ed per sistemi alimentati a batteria purtroppo molto antico come protocollo. Ha un architettura Master-Slave dove lo slave è il meter e il master è il Gateway che raccoglie i dati.

Vi sono due modi per comunicare indicati anche nella Figura 23, nel primo lo slave ha una misure e manda un messaggio SND-NR (Send No Response), dopo il quale lo slave apre una finestra

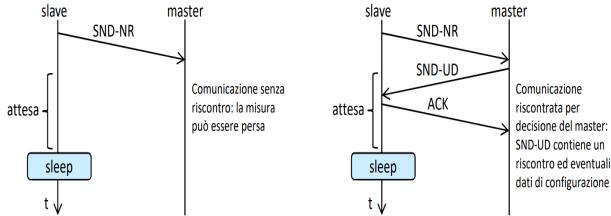


Figura 23: Invio/Ricezione segnale MBUS.

di ricezione nella quale aspetta un eventuale messaggio dal master, se non riceve risposta entra in sleep, e questo sleep è di durata imprevedibile finché non decide di mandare un altro messaggio, nel secondo caso invece il master risponde al messaggio dello slave mandando un messaggio SND-UD (Send User Data) che sono un riscontro oppure dati di configurazione es cambio precisione, frequenza campionamento ecc. a cui lo slave risponde con un ACK (acknowledgement) e va in sleep. Bisogna notare che in questo schema qui non vi è una garanzia che il messaggio/dati siano stati trasmessi, il fallimento potrebbe essere avvenuto a qualunque punto e nessuno si accorga dell'errore.

La velocità dipende dai canali, nel caso di 169 MHz vi sono due bande: a 12,5 kHz che trasmette fino a 4.8 kbit/s e 50 kHz fino a 19.2 kbit/s, mentre nel caso di 868 MHz le velocità variano dai 32 ai 100 kbit/s. Osserviamo che le comunicazioni sono molto sicure perché uso circa 3 frequenze per trasmettere un bit. I messaggi sono della seguente forma, dove DIF sono le cifre decimali e VIF è il valore:

DEV TYPE	temperature
DIF	2-digit
VIF	<valore>

Tutto il messaggio è dell'ordine di 4|8|20 bits quindi è molto compatto, il gateway che riceve

il messaggio diventa lui il terminale che ragiona su questa misurazione, in un messaggio possono esserci più misurazioni, questo serve per la compensazione del fatto che qualche dato potrebbe essersi perso in precedenza.

12.3 Rete Geografica

La rete fissa geografica ha delle prestazioni elevate e supporta protocolli anche molto sofisticati, le tipologie delle reti di accesso pubbliche sono:

- Rame - ADSL2+
- Fibra - FTTB/FTTH
- Misto - FTTC

La rete radio mobile è l'alternativa senza filo della rete fissa, la rete radio mobile sono una sequenza di architetture e tecnologie:

- 2G: GSM introdotta nel 1990, era pensato per la voce e sms;
- 3G: UMTS introdotta nel 2000, inizialmente andava a 384 kbit/s ma successivamente nel 2007 con la nascita del primo iPhone si raggiungono a decine di Mbit/s;
- 4G: LTE/LTE-A introdotta nel 2010, teoricamente raggiunge centinaia di Mbit/s come velocità;
- 5G: ?N/D? verrà introdotta nel 2020, teoricamente si potranno raggiungere anche 10 Gbit/s;

Con il 5G ci si accorge che con le reti mobile stavamo andando in una direzione opposta a quella che ci serve, essenzialmente è stato definito un triangolo di 3 possibili direzioni di evoluzione:

Il vertice in alto è eMB (Enhanced Mobile Broadband) nel quale si vuole passare da 100 Mbit/s a 10 Gbit/s, il vertice a sinistra è MMTC (Massive Machine Type Communication) nel quale si vuole aumentare 100 volte il numero di dispositivi che comunicano tra di loro, la terza direzione è URLLC (Ultra Reliable Low Latency Communication).

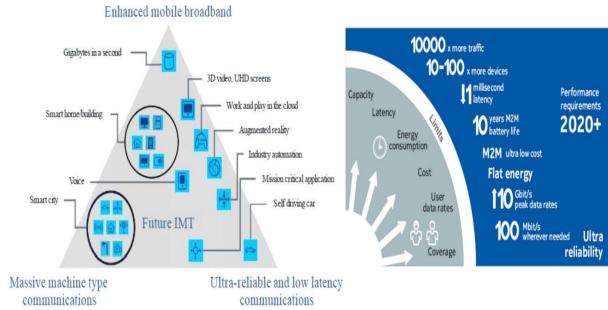


Figura 24: Tre triangoli dello sviluppo.

cy Communications) nel quale si vuole ridurre la velocità di risposta da 10 ms a 1 ms e che sia sempre in piedi.

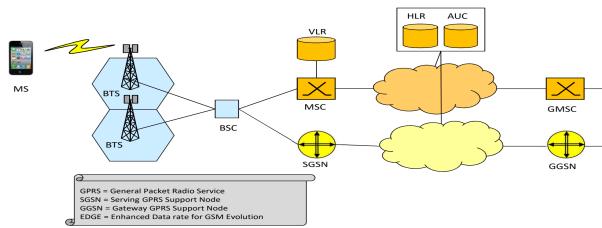


Figura 25: Architettura rete 2G.

Per le reti 2G, abbiamo un pezzo a circuito collegato ad un antenna che si collega a due componenti, uno serve per gestire la localizzazione e l'altro per raccogliere pacchetti di dati che si aggiunta dopo, e serviva perché il primo pezzo era un po' debole. La SIM è essenziale per identificare il cliente e il suo contratto e permette anche di poter cambiare il terminale.

Il 3G è una versione turbo charged della rete 2G, quindi ha un architettura molto simile ma ogni funzione è migliorata parecchio e la rete è stata resa più sicura.

I problemi sono che hanno iniziato a usare il 2G per far comunicare i vari dispositivi come calda-

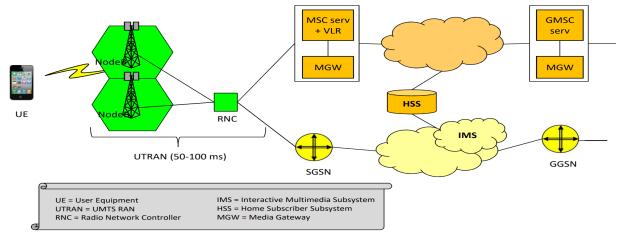


Figura 26: Architettura rete 3G.

ie, macchine ecc. perché i dispositivi 2G costavano poco, ma il 2G come tecnologia è ormai morta e le frequenze del 2G devono essere buttate via e usarli per il 5G, ma il problema sono i terminali che usano il 2G sono rimasti soprattutto in contatori e cambiarli costerebbe tantissimo quindi è sarà più facile riciclare le frequenze del 3G.

La SIM è un oggetto fisico che è proprietà dell'operatore e contiene una certa quantità di dati come il numero di telefono, dati di sicurezza e alcune applicazioni. Nel mondo dell'IoT vi sono tre problemi sostanziali con le SIM:

Se devo cambiare l'operatore devo cambiare le SIM presenti in tutti i dispositivi IoT poiché le SIM sono proprietà dell'operatore.

La SIM è un oggetto fisico quindi in casi di macchinari che generano vibrazioni continue finisce per deteriorare subito per via dei contatti meccanici.

Il problema di Roaming, in caso un'azienda compra tantissimi contratti Wind ma dov'è posizionato il sensore il segnale non arriva deve richiedere Roaming permanente ad un altro operatore ma perché questo operatore debba servire un cliente non suo permanentemente quando il roaming dovrebbe essere una cosa temporanea.

E qua nascono due soluzioni La e-SIM e la embedded SIM.

La e-SIM è una SIM logica/virtuale, in questo caso si tende a distinguere la smart card dall'o-

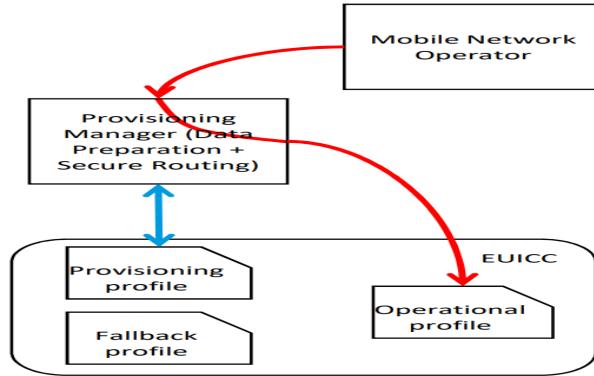


Figura 27: E-SIM.

peratore, la SIM diventa un'applicazione che ha tre profili diversi (Figura 27):

- Provisional Profile: Questo profilo serve solamente a un particolare operatore radio mobile che fa il configuratore, manda tutti i dati necessari all'operatore per accendere la Operational Profile;
- Operation Profile: E' il profilo caricato dal Provisional Profile che funziona come la SIM, e questo operatore può essere cambiato sempre da provisioning provider;
- FallBack Profile: E' una sorta di piano di Backup, viene usato nel caso Operation Profile non può essere usato risolvendo così il problema di Roaming, poiché basterebbe usare un altro operatore come Backup nel caso di assenza di segnale ecc.

Ci sono anche delle embedded SIM, sono delle SIM che si saldano sui dispositivi risolvendo le problematiche meccanici, la soluzione migliore sarebbe un'integrazione delle e-SIM con le embedded SIM.

Il 4G è una rete totalmente a pacchetto quindi c'è il solito Gateway e siccome è totalmente su IP anche lo voce passa su IP. Per la comunicazione Machine2Machine vi sono due protocolli:

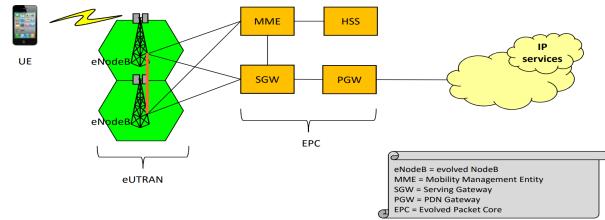


Figura 28: Architettura rete 4G.

- LTE-M: vengono tagliati all'interno delle bande disponibili delle fette di banda dedicate a questo tipo di comunicazione, funziona a 1.4 MHz e trasmette fino a 1Mbit/s (molto cauto come approccio) qua il guadagno è nella sensibilità dello strumento. Ha un power budget di 160 dB, facendo una prova in basement si vede che trasmettendo a 800MHz la percentuale di trasmissione è 99%, e a 2600MHz la percentuale di trasmissione è 86%;
- NarrowBand IoT: ha più potenza disponibile di 164dB ma raggiunge velocità fino 250 kbit/s quindi è pensato per il basso consumo, qua nella prova basement abbiamo con 800MHz il 99% e con 2600MHz 92%;

Questi protocolli sono stati costruiti come modifiche agli oggetti già esistenti (radio mobile) quindi si possono usare insieme alla rete già esistente (non devo cambiare la rete) ma ereditano il problema della comunicazione asincrona quindi una parte dell'energia si spreca per rimanere sincronizzati.

Una delle soluzioni è la creazione delle Reti ad Hoc a bassissimo consumo e grandissima penetrazione solo per queste applicazioni, queste reti si chiamano LPWAN (Low Power Wide Area Network) di cui una delle applicazioni più importanti sono LoRaWAN (Long Range WAN), grazie a questo protocollo si passa da 3 anni usando NB-

IoT a 10 anni usando LoRa.

LPWAN: Non funzionano su bande licenziate quindi non sono gestite da operatori radio mobili e non sono mobili e non prevedono una copertura come le reti radio mobili, es. quello di Vodafone, un esempio è Sigfox pensata per consumi bassissimi e gittata lunga (30-50km) quindi ricoprono le zone con poche antenne, velocità è circa centinaia di bit al giorno, serve di solito per il metering molto famosa in Francia, problema è che è totalmente un sistema chiuso, tutto proprietario. Un esempio ancora più rilevante è LoRa(LoRaWAN) che è un tecnologia di accesso(LoRa) (la parte di WAN invece riguarda la rete) ed è proprietaria (patentata da Sentex) e Sentex fabbrica i chip e li vende e lo si può attaccare dove voglio ma a differenza di Sigfox si conoscono le specifiche di LoRa.

LoRaWAN funziona su bande non licenziate tipicamente 868MHz, quindi è sottoposta a certe regolazioni che varia a seconda dei paesi, in generale a 868MHz vi sono diversi canali, che possono essere usati se la massima potenza è sotto 20dBm (e 165 dB). Ci sono anche delle limitazioni sulle duty cycle, quindi non posso trasmettere costantemente.

LoRa usa CSS (Chirp Spread Spectrum) per la modulazione(come metto i bit sul canale radio) che si basa su un'onda come indicata nella Figura 29 (come un uccello che chirpa), il segnale cresce di frequenza fino al massimo (Upchirp) oppure il contrario (Downchirp), usando upchirp e downchirp permette di codificare il messaggio:

Alcuni tipi di spread spectrum sono:

- FHSS (del BT), qua ho una banda a disposizione e questa viene divisa in molte sottobande e salto tra queste bande durante la trasmissione;
- DSSS(Direct Sequence Spread Spectrum) viene usato da alcuni standard WiFi;

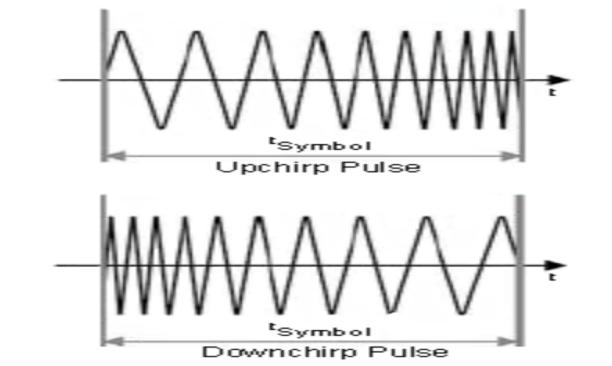


Figura 29: Chirp effect on a wave.

- OFDM che assomiglia molto a FHSS (ho tanti sottocanali), ma qua trasmetto a tutti i sottocanali contemporaneamente a bassa velocità;
- CSS (chirp) usata in LoRa e alcune applicazioni radar.

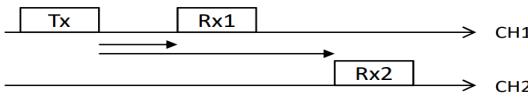
Le varie modulazioni servono i problemi specifici, ad es. per evitare disturbi su alcuni canali sia FHSS che OFDM vanno bene, mentre il CSS è una modulazione estremamente efficiente e robusta.

LoRa è molto efficiente dal punto di vista del basso consumo e grande penetrazione, infatti consuma $32mA$ durante trasmissione e $1\mu A$ durante lo sleep, per confrontare con NB-IoT dove abbiamo $120mA$ durante trasmissione e $5\mu A$ durante lo sleep. Questo è dovuto al fatto che LoRa è un protocollo totalmente asincrono, quindi poco durante lo sleep, mentre NB deve periodicamente comunque aggiornare il terminale della sua presenza (sincronizzazione).

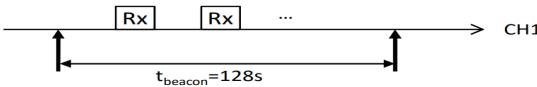
In LoRa vi sono 3 protocolli per accedere alla rete che sono delle opzioni sempre crescente che si possono applicare al protocollo base:

- Protocollo (A) usato per i battery operated Sensors: manda un segnale poi attende un

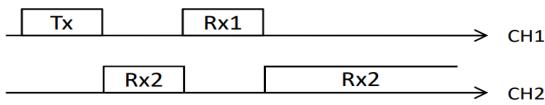
tempo t_1 per una risposta su un canale e dopo su un altro canale apre un'altra finestra per la ricezione dando così due possibili istanti nei quale può ricevere un riscontro;



- Protocollo (B) usato per i battery powered actuator: questo consuma di più perché deve ricevere dei comandi e implementarli in un determinato periodo (sorta di deadline), quindi lui usa un canale un frame di 128 secondi con dei beacon (per la sincronizzazione) e riserva per un determinato terminale alcuni slot;



- Protocollo (C) usato per i mains powered actuators: qui non c'è un problema di consumo perché viene alimentato dalla rete elettrica e quindi usa due canali il primo è simile al Protocollo (A), mentre il secondo canale riceve in tutte le fasi in cui non è attivo il canale 1 (infondo deve risparmiare il più possibile quindi chiude il canale se è attivo il primo).



Chiaramente il caso A è il caso teoricamente meno consumante, e anche in pratica la differenza di consumo è netta.

Nell'architettura LoRaWAN (Figura 30) un sensore può vedere più GateWay e trasmette il messaggio su tutte i gateway che vede, in questo caso tutti gateway mandano lo stesso messaggio al Network Server che poi procede ad eliminare

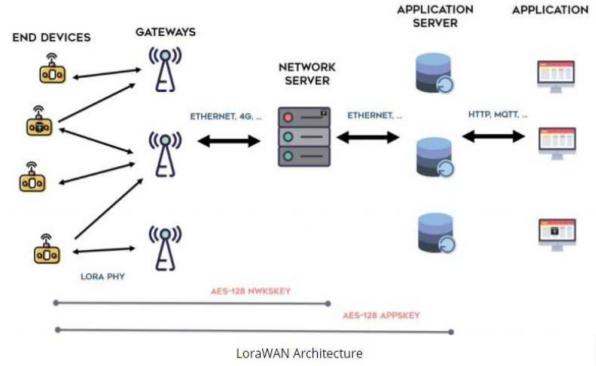


Figura 30: L'architettura di una LoRaWAN.

le eventuali copie, quindi c'è una totale libertà e flessibilità nel mettere i gateway, chiaramente aumentando i gateway aumenta la probabilità della ricezione del messaggio (Availability). Dopo il Network Server possono esserci delle Application server. Per la sicurezza abbiamo due livelli inscatolati: il primo dal end device (sensore) al network server e il secondo livello che è end to end dal end device al application server ma siccome le application server sono libere di essere quello che vogliono, quindi il secondo livello di sicurezza è lasciata per un implementazione libero ad esempio si può usare MQTT.

13 Trasporto

Il livello del trasporto virtualizza la rete per le applicazioni, assumiamo di avere due terminali e in mezzo a loro un host, i pacchetti trasmessi (segmenti) sono solo visibili alle applicazioni per i livello di rete sono solo dei pacchetti con qualche dato.

I due protocolli più usati per il trasporto di base sono UDP e TCP, dove UDP è un protocollo molto elementare e fa pochissime cose: prende

un pacchetto dalle applicazioni e lo porta alle direttamente alle applicazioni del terminale ricevente attraverso le porte (che devono essere rese visibili), dopo aver ceduto controlla se tutto è andato bene, inoltre non crea una sessione end to end per la trasmissione del messaggio ma considera ogni messaggio a se stante. Sopra UDP si possono aggiungere dei protocolli un esempio è UDP + RTP (Real Time Protocol) per gestire applicazioni che trasmettono in tempo reale, UDP capisce se c'è un errore ma non ritrasmette in caso di errore. L'header di un pacchetto UDP è abbastanza leggero - 8 byte.

TCP è un protocollo alternativo completamente diverso, in cui viene creata una sessione di trasporto (collegamento di trasporto) all'interno del quali i dati sono ordinati. TCP ragiona sul flusso di byte e verifica che ci sia un ordinamento e in caso di mancanza di dati può ricostruire il messaggio con il meccanismo della ritrasmissione. L'header di un protocollo TCP pesa 20 byte. Gestisce anche la congestione della rete, adattandosi alle potenzialità della rete cercando di saturare le sue capacità.

Il protocollo TCP funziona così: All'inizio per aprire una connessione il terminale A con B invia il segnale di SYN e se B vuole aprire la connessione rinvia un segnale SYN-ACK e poi c'è una terza conferma da parte di A che invia un segnale ACK a B dopo di che inizia una sessione di trasporto, anche la chiusura funziona in 3 fasi, nel quale B invia un segnale FIN a A che viene risposto o con un ACK o ACK-FIN (se non chiudere o chiudere) e poi B rinvia un segnale ACK. Per la trasmissione invece a ogni invio di dato, riceve un ACK in caso di recezione con successo del dato e in caso non riceve ACK per un certo periodo rinvia il dato, questo meccanismo diventa sempre più complesso con presenza/invio di tanti dati. E in caso di morte di uno dei ter-

minale, il terminale vivo continua a mandare i pacchetti senza ricevere riscontro fino ad entrare in timeout e la connessione si chiude.

Si può notare che il TCP non va bene per invio dei dati della sensoristica (tranne eccezioni come le telecamere) visti i tantissimi controlli della qualità, ma anche UDP non va bene perché in realtà non mi aggiunge nulla, il vantaggio di UDP è che riesco a usarlo per indirizzare e distinguere tante applicazioni o per un supporto per applicazione che richiedono trasmissione in tempo reale.

Il livello superiore del trasporto (applicazione) può usare HTTP per fare uno scambio di oggetti (Documenti):

- HTTP 1.1 è un protocollo client-server del 1997, dove il client apre una connessione TCP con il server e poi trasmette il documento, alcune delle operazioni sono GET, HEAD, POST, PUT, DELETE. Questo protocollo è molto verboso nel senso che l'header dei pacchetti è grande e questo perché:

1. Le richieste sono in ASCII (es. GET), che già pesa abbastanza pesante (circa 1 kbyte);
2. Inoltre si aggiungono delle opzioni alla richiesta che occupa anche esso memoria;
3. È pensata per reti che non hanno problemi di consumo.

Il vantaggio è che HTTP è implementato ovunque, quindi non richiede la costruzione di né il client né del server.

- HTTP 2.0 (2015) Il vantaggio è che si passa dal modello richiesta-risposta del HTTP 1.1 al modello richiesta-tante risposte (usando il meccanismo di server push), es. mandami aggiornamento sul cambiamento di uno certo stato (di solito si usava JavaScript nel 1.1 mentre in 2.0 entra a far parte del protocollo

stesso). Inoltre nel 2.0 vi possono essere dei stream indipendenti (in HTTP 1.1 si usavano dei pipeline), ma la cosa più importante è che nel HTTP 2.0 esiste una compressione del header, risolvendo il problema del overhead pesante.

Esiste un modo per avere TCP senza usare HTTP (poiché HTTP può essere molto pesante per il mondo della sensoristica), un modo è scrivere a mano il codice che lo fa sicuramente è un modo molto efficiente ma esiste il concetto di WebSockets (RFC 6455) che usa HTTP in una maniera molto particolare, usa HTTP GET usando l'opzione ‘Upgrade: WebSockets’, che cambia il protocollo passando da HTTP a WebSockets, la risposta contiene la stringa ‘Switch Protocol’. Il vantaggio è che è compatibili con tutto ciò che è compatibili con HTTP (cioè tutto), WebSockets è un stream TCP end-to-end su cui vengono solo trasmessi bytes su pacchetto con l'overhead che è costruito con un opCode e campo lunghezza, il campo lunghezza è un campo variabile. I messaggi che opCode può mandare sono DATA, PING, PONG, CLOSE. Questo overhead ha una dimensione minore di 2 byte se la lunghezza è inferiore a 126 byte, a questo overheader segue un campo MASK (che pesa 4 byte) che serve per evitare che chiunque possa imitare un messaggio e il campo DATI. PING e PONG servono solo per controllare se l'interlocutore è ancora vivo. Il segnale CLOSE serve per chiudere la connessione totalmente.

Il modello che vogliamo raggiungere (per il mondo della sensoristica) è quello di tipo publisher-subscriber (fino ad adesso abbiamo visto client-server) nel quale qualcuno pubblica dei dati e sono disponibili a tutti quelli che fanno subscribe, che viene raggiunto con il protocollo MQTT.

MQTT sta nello strato applicativo, e sostitui-

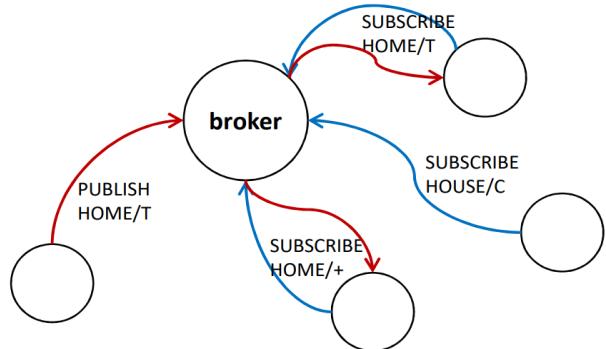


Figura 31: L'architettura del protocollo MQTT.

sce HTTP, a differenza di HTTP non c'è collegamento tra chi crea e chi riceve gli eventi. È un modello di tipo publish subscriber, e per diventare subscriber si un messaggio nel quale viene comunicata la gerarchia a cui si vuole sottoscrivere, la gerarchia viene indicata con questo formato: gerarchia/sotto-gerarchia/sotto-sotto-gerarchia... preceduto da SUBSCRIBE. Un subscriber può anche essere un publisher. Il Broker sta in mezzo ai publisher e subscriber, non è un sistema di code, lui semplicemente distribuisce i messaggi ai subscriber quindi non si preoccupa della persistenza dei dati ricevuti dal publisher ma si può creare una connessione persistente (con QoS indicati dopo) nel quale l'ultimo messaggio viene memorizzato relativo a un certo topic. MQTT funziona sopra TCP/TLS (TLS è un protocollo di trasporto sicuro basato su TCP) che a loro volta funzionano su IP, ma sono stati studiati MQTT su diversi protocollo es. MQTT su ZigBee o su UDP (sempre su IP).

I messaggi di MQTT possono essere divisi nei seguenti gruppi:

1. CONNECT, CONNACK e DISCONNECT: sono messaggi tra publisher e broker per connettersi e creare una connessione

- applicativa;
2. SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK: sono messaggi tra subscriber e broker per connettersi, con SUBSCRIBE viene specificato anche l'argomento nella gerarchia;
 3. PUBLISH per pubblicare i dati e gli eventi, può avere dei flags, ad es. per indicare la Quality of Service: QoS=0, è solo un publish senza nessun controllo, mentre usando QoS=1 il PUBLISH viene risposto con il PUBACK, infine QoS=2 è una catena di 4 messaggi, PUBLISH/PUBREC/PUBLISH/PUBCOMP, osserva QoS=0 è at most once, QoS=1 è al least one e infine QoS=2 è exactly once;
 4. PINGREQ, PINGRES per controllare lo stato di vita.

Il messaggio CONNECT contiene un LAST WILL & TESTAMENT es. EMERGENCY/SHUTDOWN nel caso di anomalia e il LAST WILL & TESTAMENT viene inviato a tutti i suoi subscriber, è un meccanismo molto utile nel caso vi è un subscriber che non fa altro che controllare se i publisher stanno bene. Questo sistema presuppone che il broker sia molto robusto, i broker devono stare in vita per una maggiore affidabilità quindi si possono avere due Broker, il che non è così complicato, poiché allinearli sono così difficile visto che deve persistere pochi dati.

Il messaggio MQTT è molto leggero, il control header pesa 1 byte, il packet lenght varia tra 1-4 byte in base alla lunghezza del messaggio:

Control	Packet	Message	Data
Header	Length		

Questo protocollo è più di tipo middleware cioè sembra più un protocollo di messaggistica (ma non totalmente ad esempio non il servizio delle

code dei messaggi) che applicativo (è applicativo per forza visto che sta sopra il protocollo di trasporto. E' il protocollo più diffuso nel mondo del IoT.

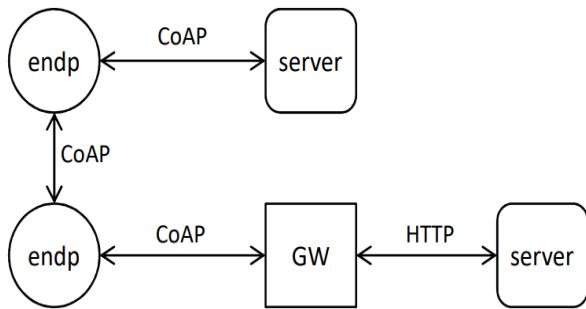


Figura 32: L'architettura del protocollo CoAP.

Un altro protocollo **CoAP**, è un protocollo che è mappabile su un sottoinsieme di HTTP, in particolare usa comandi GET, PUT, POST, DELETE di HTTP, è pensato per ambienti IoT. CoAP è pensato per viaggiare sopra UDP, optionalmente si può inserire, tra UDP e CoAP, uno strato di sicurezza come DTLS. Più tardi è stato mappato su quasi tutti i protocolli di trasporto tra cui anche SMS.

Gli endpoint in CoAP possono comunicare direttamente con dei server e tra di loro senza passare per un Gateway, poiché è intercambiabile con HTTP, questo è il vantaggio di usare le istruzioni si traducono in HTTP direttamente. Il meccanismo CoAP non è completamento Publish Subscriber, ma esiste una pseudo-subscription usando opzione Observe con il comando GET, che osserva e manda ogni aggiornamento di uno stato al richiedente. E' diverso dal MQTT, infatti l'Observe è più debole visto che non posso ricevere aggiornamento su più oggetti (sotto directory della directory osservata) a meno di specificare a mano tutti gli oggetti offerti come servizio.