POLITECNICO DI TORINO

Corso di Laurea Magistrale in

Ingegneria Informatica

Tesi di Laurea Magistrale

Validazione di soluzioni software basate su virtualizzazione a fronte delle attuali soluzioni basate su appliance dedicate per il settore elettro-energetico

Relatore

prof. Risso Fulvio Giovanni Ottavio

Candidato

Zappulla Carmelo

Anno accademico 2020/2021

Sommario

INTRODUZIONE

Le numerose sperimentazioni condotte nel campo delle power-grid hanno dimostrato che le tradizionali centrali elettriche a base di carbone potranno presto cooperare con forme di energia rinnovabili ed emergenti, come vento e sole, per ridurre il consumo di carbone e le conseguenti emissioni di anidride carbonica.

Con forme di energie alternative disponibili oggi in quantità illimitata, la necessità di generare energia in modo decentralizzato sta aumentando e come risultato, i tradizionali sistemi di energia devono far fronte a nuove sfide.

Poiché il sistema di rete elettrica potrebbe avere oltre milioni di consumatori e la domanda della sua affidabilità e sicurezza è estremamente critica, gli operatori necessitano di tecnologie nuove ed emergenti.

Attraverso l'utilizzo di tecnologie ICT, una massiccia distribuzione di IoT device e una robusta infrastruttura di comunicazione, la rete elettrica può avere un controllo capillare, affidabile, efficiente e una sicurezza dell'intero sistema riducendo la possibilità di blackout elettrico o le emissioni di anidride carbonica.

In questo modo, i consumatori possono ridurre al minimo gli eccessi e la spesa per l'energia utilizzata, regolando il funzionamento degli elettrodomestici intelligenti per utilizzare forme di energia rinnovabile oppure evitare il consumo durante le ore di punta.

Il paradigma gerarchico di computing Fog/Edge/Cloud (FEC) è visto come una soluzione efficace nel gestire il grande volume di dati time-sensitive che viene prodotto da device intelligenti creando un nuovo sistema elettro-energetico che prende il nome di Smart-Grid.

Questo documento ha il compito di validare i vantaggi derivanti dall'utilizzo di tecnologie di virtualizzazione e orchestrazione di dispositivi IoT nella rete elettrica odierna tramite la realizzazione di una piccola rete infrastrutturale costituita dai principali dispositivi coinvolti in ambito elettro-energetico.

L'attività di ricerca si propone quindi di valutare l'utilizzo di dispositivi general-purpose, elencandone i vantaggi e gli eventuali svantaggi rispetto alle tradizionali appliance con

hardware dedicato per guadagnare funzionalità aggiuntive in termini di agilità, basse latenze di comunicazione e minore larghezza di banda offerte da tecnologie di computing FEC abilitando la nascita di un nuovo tipo di applicazioni e servizi.

Questo elaborato è strutturato come segue:

- **Capitolo 1:** descrive i concetti del Cloud Computing in relazione al Fog/Edge Computing, spiegando come una piattaforma Fog/Edge è in grado di abilitare il passaggio da grid tradizionale a Smart-Grid.
- **Capitolo 2:** fornisce una classificazione delle principali tecnologie di virtualizzazione disponibili sul mercato mostrandone i punti di forza delle rispettive e gli svantaggi.
- **Capitolo 3:** descrive i principali dispositivi coinvolti in ambito elettro-energetico e le migliorie che si ottengono conseguentemente ad un massiccio dispiegamento di dispositivi IoT e all'utilizzo di infrastruttura ICT all'interno della grid. Infine illustra vantaggi e svantaggi legati all'uso della virtualizzazione a fronte dell'utilizzo di appliance dedicate per il raggiungimento di un dato servizio.
- **Capitolo 4:** illustra le caratteristiche più importanti di Docker che rappresenta la soluzione più popolare di containerizzazione fornendo tool accessori come Docker compose, anch'esso illustrato in questo capitolo, per lanciare applicazioni costituite da più containers.
- **Capitolo 5:** discute gli applicativi messi a disposizione dal pacchetto iPDC software utilizzato in questo lavoro di ricerca; illustra lo standard IEEEc37.118, la comunicazione tra PMU-PDC e gli steps da seguire per abilitare il setup della comunicazione tra questi.
- **Capitolo 6:** descrive l'architettura e l'implementazione adottata per la realizzazione di una Smart-Grid costituita da dispositivi PMU-PDC virtualizzati ossia operanti come funzione software su hardware general-purpose; infine, viene testata la correttezza delle comunicazioni tra i principali dispositivi dispiegati all'interno della grid.
- **Capitolo 7:** vengono elencate le possibili migliorie da adottare per rendere più smart

la grid implementata; tra tutte, la presenza di un orchestratore Kubernetes e la presenza di SDN sembrano le soluzioni maggiormente adottate al giorno d'oggi.

1. Cloud e Fog/Edge Computing

In questo primo capitolo introduttivo viene messa a paragone la tecnologia di Fog/Edge Computing con quella più tradizionale del Cloud evidenziando i vantaggi e le eventuali differenze.
Viene successivamente descritto come tali tecnologie possono introdurre migliorie nel settore elettro-energetico contribuendo alla nascita della Smart-Grid.

1.1 Cloud Computing

Il C*loud Computing* è un modello per consentire l'accesso alla rete onnipresente, conveniente e su richiesta a un pool condiviso di risorse di elaborazione configurabili (ad esempio reti, server, archiviazione, applicazioni e servizi) che possono essere rapidamente forniti e rilasciati sotto forma di architettura distribuita, con il minimo sforzo di gestione o interazione con i fornitori di servizi [1].

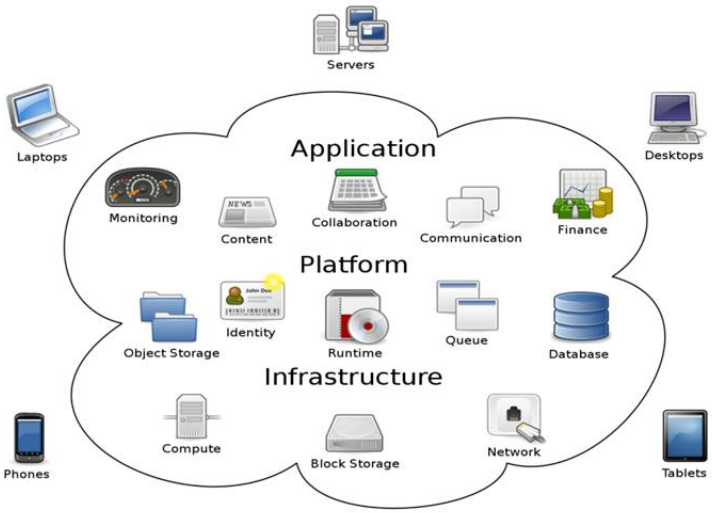


Figura 1: architettura Cloud Computing

I vantaggi principali nell'uso del Cloud Computing possono essere schematizzati come segue:

- *elasticità*: possibilità, nell'ordine di minuti o secondi, di aumentare, diminuire o addirittura azzerare (scale to ZERO) l'utilizzo delle risorse a disposizione di un utente. Fra tutti, tale vantaggio risulta essere il più importante in grado anche di differenziare il Cloud Computing dalla sola virtualizzazione. In ambiente Cloud la scalabilità delle risorse avviene in maniera rapida, trasparente e pressoché illimitata, coinvolgendo più dispositivi, mentre in ambiente virtualizzato dipende dalla macchina che gestisce le risorse;
- *economia di scala*: ottenuta attraverso la fornitura di servizi Cloud Computing utilizzando data center di grandi dimensioni, in grado di servire molti utenti contemporaneamente con ottimizzazione delle risorse ed abbattimento dei costi, che consente agli utenti finali prezzi inferiori rispetto ai sistemi classici;
- *pay-as-you-go*: un utente paga solamente per i servizi effettivamente utilizzati senza sostenere un investimento iniziale, in quanto non è necessario comprare le infrastrutture ma è possibile utilizzare i servizi forniti pagando solo per il loro effettivo utilizzo.

1.2 Fog/Edge Computing

Il *Fog Computing (o Edge Computing)* è un'architettura orizzontale, a livello di sistema, utile a distribuire risorse e servizi di elaborazione, archiviazione, controllo e networking ovunque lungo l'infrastruttura che li connette al Cloud, immagazzinando dati, distribuendo controllo e funzionalità di rete più vicino agli utilizzatori – dispositivi [2].

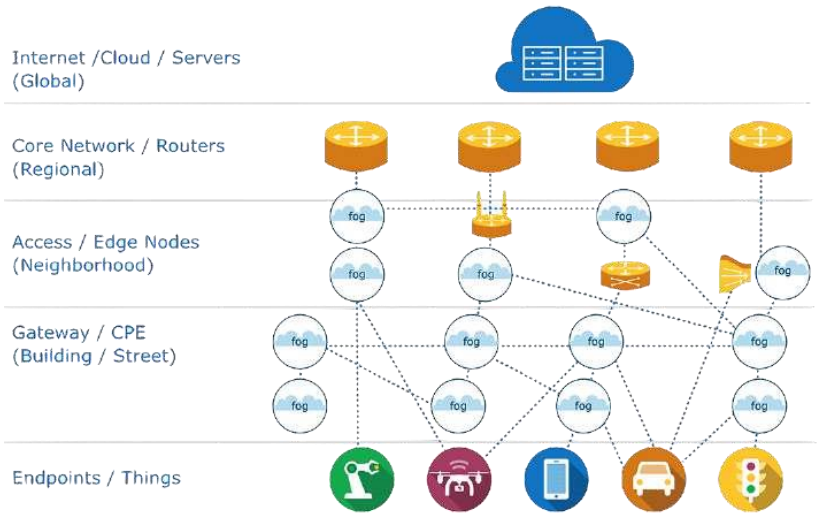


Figura 2: architettura Fog/Edge Computing

Tra le caratteristiche principali, le più importanti sono:

- *posizionamento periferico e consapevolezza della posizione*: vengono supportati end-point con servizi avanzati ai bordi della rete, comprese applicazioni con bassi requisiti di latenza (giochi, streaming, realtà virtuale);
- *distribuzione geografica*: le infrastrutture indirizzate dal Fog richiedono architetture ampiamente distribuite per la fornitura dei dati velocemente;
- *numero elevato di nodi*: come conseguente dell'ampia distribuzione geografica;
- *predominanza dell'accesso wireless;*
- *supporto per analisi on-line*: i nodi Fog sono posizionati nella rete dati per svolgere un ruolo significativo nella raccolta e nel trattamento dei dati vicino alla loro fonte (elaborazione dati vicino agli utilizzatori);
- *interazione in tempo reale:* conseguenza derivante dall'unione delle caratteristiche sopra elencate.

1.3 Confronto Cloud/Fog/Edge Computing

La differenza principale tra Cloud e Fog/Edge Computing è legata principalmente al numero di dispositivi coinvolti, a dove questi vengono posizionati e distribuiti e

di conseguenza al tempo di latenza delle comunicazioni.
Si può osservare che, per le sue caratteristiche, il Cloud Computing non è adatto per applicazioni che richiedono in tempo reale un'azione quasi immediata da parte del server con tempi di risposta di centinaia di millisecondi.
Il Fog/Edge Computing invece tende a portare il processing il più vicino possibile alle fonti dati, senza necessità di trasferirli in Cloud o verso altri sistemi remoti per le attività di processing. Dunque, soddisfa i requisiti più stringenti di bassa latenza, ridotto consumo di banda e sicurezza delle applicazioni.
Naturalmente, quest'ultimo non può sostituire totalmente il Cloud Computing che è ancora preferito per i processi di elaborazione batch[1] e che necessitano di maggiore scalabilità delle risorse.
Quindi, in funzioni del caso d'uso occorre considerare la possibilità di adottare architetture Cloud Computing, Fog/Edge Computing e on mix dei due paradigmi.

1.4 Fog/Edge Computing e Smart - Grid

Sin dagli arbori, la potenza viene prodotta creando una forza per far girare un grande generatore elettrico che a sua volta produce elettricità che viene poi trasferita nella rete elettrica. Indipendentemente da come viene prodotta l'energia, gli impianti e le strutture di produzione sono collegate alla P*ower-Grid* nazionale e la loro potenza viene trasferita, attraverso linee di trasmissione ad alta potenza e interconnesse tra di loro, verso un centro di controllo trasferendola a diverse regioni o aree.
Generalmente la potenza trasferita dal centro di controllo arriva alle sottostazioni che riducono in primis la corrente e solo successivamente questa viene pompata nelle linee elettriche affinché arrivi ad una comunità per essere utilizzata nelle case e imprese.
Una *Smart-Grid* può essere paragonata ad una Power-Grid tradizionale ma

[1] Sistema per eseguire elevati volumi di job sui dati di tipo ripetitivo. Il metodo in batch consente di elaborare i dati quando sono disponibili risorse di elaborazione sufficienti e con un'interazione minima o nulla da parte dell'utente.

intelligente-cooperativa dove dispositivi WSAN[2] e WAMS[3] vengono affiancati da una distribuzione massiva e capillare di dispositivi IoT[4] al fine di garantire un controllo a granularità più fine in tutta la rete.
Insieme a tecniche di misurazione, trasformano il flusso unidirezionale, quindi dai sensori al centro di controllo, in flusso bidirezionale, ovvero dal centro di controllo verso i sensori/attuatori.

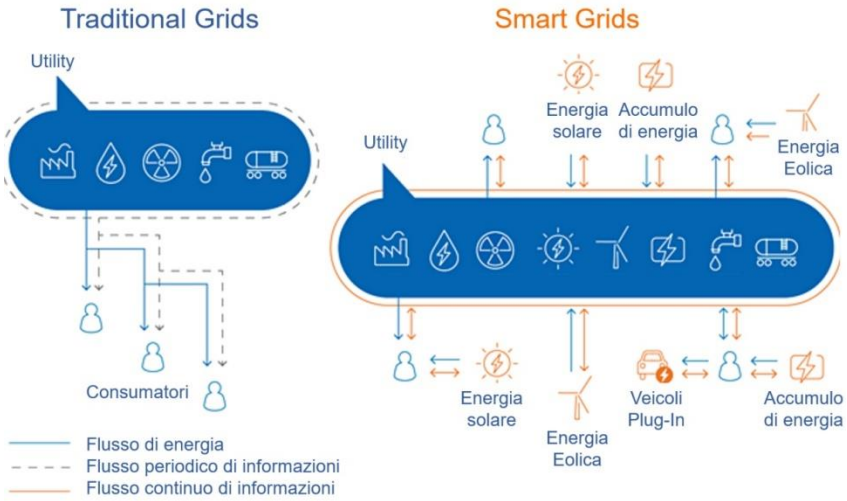


Figura 3: confronto tra una grid tradizionale e una Smart-Grid

Le Smart-Grid sono pensate come le reti elettriche di prossima generazione in cui i processi di gestione e distribuzione dell'elettricità sono progettati digitalmente per monitorare, proteggere e ottimizzare automaticamente in tempo reale i flussi bidirezionali sia di energia che di informazioni tra fornitori di servizi e consumatori.
Le migliorie introdotte dall'uso di una Smart-Grid sono enunciate in seguito:

- bilanciamento tra la richiesta di energia e la conseguente fornitura di questa;

[2] Dispositivi elettronici autonomi in grado di prelevare dati dall'ambiente circostante e di comunicare tra loro.
[3] Rete intelligente che applica misurazioni in tempo reale in sistemi di controllo automatici per gestire un'infrastruttura di trasmissione elettrica affidabile, efficiente e sicura.
[4] Acronimo di Internet of Things: indica 'oggetti' informatici – general purpose che acquisiscono intelligenza grazie al fatto di poter comunicare dati su se stessi e accedere ad informazioni aggregate da parte di altri.

- capacità di distribuire elettricità in modo ottimale dalla fonte al consumo;
- migliore gestione dell'energia, riducendo al minimo le interruzioni di corrente;
- trasporto della sola quantità di energia richiesta sulla base delle misure lette;
- integrazione delle risorse energetiche verdi nel sistema di distribuzione dell'energia;
- segnalazione ai consumatori ogni volta che il consumo di energia raggiunge valori elevati o anomali.

Tramite l'utilizzo di macchine IoT connesse tra di loro e un flusso continuo di informazione scambiate tra esse, si riesce a monitorare in maniera capillare l'ambiente circostante e di conseguenza si possono prendere decisioni intelligenti e veloci.

A tal fine, latenza[5] e jitter[6] diventano vincoli stringenti per il corretto funzionamento dei dispositivi dato che questi necessitano di una rapida risposta per garantire la loro operatività in maniera efficace.

Fog/Edge Computing è una piattaforma capace di soddisfare tali requisiti e garantire i vantaggi sopra descritti; infatti, estende il Cloud Computing all'interno della struttura di rete in modo che i processi di calcolo, i dati di archiviazione e la rete di comunicazione possano essere eseguiti localmente più vicino alla fonte.

Le caratteristiche del Fog/Edge quali organizzazione gerarchica, distribuzione massiva, consapevolezza della posizione, rendono la piattaforma adatta per sopportare l'elevato numero di dispositivi IoT necessari per la nascita di una Smart-Grid.

[5] Intervallo di tempo che intercorre fra il momento in cui viene inviato l'input/segnale al sistema e il momento in cui è disponibile il suo output.

[6] Variazione statistica nel ritardo di ricezione dei pacchetti trasmessi, causata dalle code interne ai router congestionati.

2. Virtualizzazione

In questa sezione viene presentato il concetto di *virtualizzazione* analizzando i dettagli della tecnologia, quali sono i diversi tipi di virtualizzazione attualmente disponibili e le particolarità della "Operating-System Level Virtualization", meglio conosciuta come *containerizzazione*.

2.1 Definizione

La virtualizzazione consiste nel sostituire hardware dedicati a specifiche funzionalità con hardware di computing generico (COTS[7]), su cui vengono allocate delle risorse "virtualizzate" ed eseguiti (es., macchine virtuali – VM, containers).

2.2 Tipi di virtualizzazione

Le tecnologie di virtualizzazione oggi disponibili sul mercato sono parecchio eterogenee e sfruttano differenti paradigmi.
I diversi tipi di virtualizzazione sono:

- *Full Software Virtualization*: tutto l'hardware viene simulato all'interno dell'esecuzione di un software, sul quale poi viene eseguito un SO completo, che crede di agire su una infrastruttura fisica;
- *Lightweight Virtualization:* a differenza della full virtualization, utilizza la virtualizzazione a livello di sistema operativo (o a livello di applicazione) e di conseguenza l'hardware non viene simulato;
- *Hardware-Assisted Software Virtualization:* differisce dalla precedente per il fatto che alcune funzionalità hardware vengono "accelerate", grazie alla

[7] Acronimo di common of the shell: indica dispositivi di computing con hardware generico dove il software ne stabilisce il comportamento che tale dispositivo general-purpose dovrà eseguire.

"collaborazione" dell'hardware sottostante, che fornisce un supporto nativo alla virtualizzazione;

- *Paravirtualization*: dove l'hardware non viene virtualizzato al 100%, per cui il guest OS agirà in maniera consapevole interfacciandosi in parte con l'hardware sottostante. Questo permette un minore overhead (l'hardware non è completamente replicato in software) e la possibilità di interfacciarsi direttamente con i driver dell'host OS, riducendo anche l'overhead di esecuzione.

Nel seguito, verranno analizzate nel dettaglio la full virtualization e la light virtualization e le applicazioni software principali che ne fanno uso.

2.2.1 Full virtualization: Virtual machine

Il termine macchina virtuale (VM) indica un software che, attraverso un processo di virtualizzazione, crea un ambiente virtuale che emula tipicamente il comportamento di una macchina fisica (PC, client o server) grazie all'assegnazione di risorse hardware (porzioni di disco rigido, RAM e risorse di processamento) ed in cui alcune applicazioni possono essere eseguite come se interagissero con tale macchina; infatti se dovesse andare fuori uso il sistema operativo che gira sulla macchina virtuale, l'hypervisor[8] (sistema di base) non ne risentirebbe affatto. Ogni VM include una copia intera del sistema operativo, una o più applicazioni e tutte le librerie necessarie.

[8] Conosciuto anche come virtual machine monitor (VMM), è il componente centrale e più importante di un sistema basato sulle macchine virtuali.

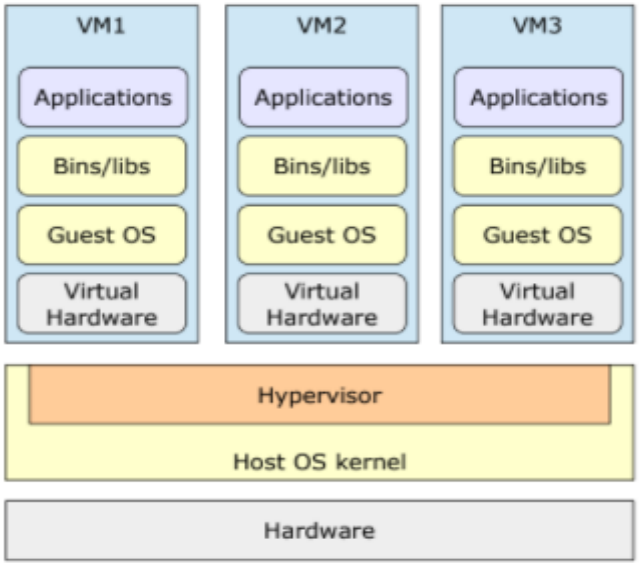


Figura 4: architettura di una macchina virtuale

Il significato più comune oggi è quello di un programma che emuli un calcolatore (di solito un calcolatore astratto, cioè a cui non corrisponde un calcolatore reale). I programmi applicativi sono scritti in un linguaggio compilato (cioè tradotti nelle sue istruzioni native) e, una volta compilati, sono eseguiti sulla macchina virtuale software, che può agire o come interprete o come compilatore 'al volo'[9].

Il programma compilato può "girare" su qualsiasi piattaforma su cui "giri" la macchina virtuale. L'hypervisor è il componente chiave per un sistema basato sulla full virtualizzation; esso garantisce non solo la corretta esecuzione ma anche l'isolamento tra le diverse virtual machine dispiegate al suo interno.

2.2.2 Light virtualization: Container

La containerizzazione è anche conosciuta come "Operating-system level virtualization". In questo tipo particolare di "virtualizzazione" il kernel del sistema operativo in esecuzione sul device host permette l'esistenza di più istanze in user-space isolate tra loro, dette "containers". Dal punto di vista delle

[9]Compilazione just-in-time: tipo di compilazione, conosciuta anche come traduzione dinamica, effettuata durante l'esecuzione (on-the-fly) del programma piuttosto che precedentemente.

applicazioni in esecuzione all'interno di ciascun container, questo appare come una macchina a sé stante, da cui sono visibili soltanto le risorse, i dispositivi, i file, e i dispositivi hardware assegnati a quel container.

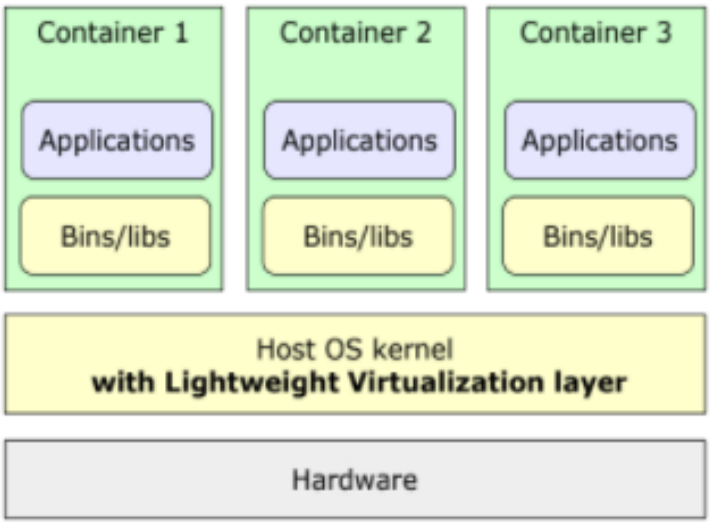


Figura 5: architettura di un container

Si può subito notare che, a differenza delle macchine virtuali la cui rappresentazione è mostrata in Figura 4: architettura di una macchina virtuale, il sistema operativo di tutti i container è condiviso e coincide con quello in esecuzione sulla macchina host.

Un container include al proprio interno tutto quello di cui ha bisogno per essere eseguito: codice, tool di sistema, librerie e parametri di configurazione.

I software che vengono racchiusi in un container possono essere eseguiti su qualsiasi tipo operativo come Linux, Windows e MacOS e, nonostante ciò, funzioneranno in modo analogo. Dunque, esso assume sempre lo stesso comportamento di esecuzione su qualsiasi host senza essere vincolato alle caratteristiche di sistema quali versione kernel e/o distribuzione nel quale viene lanciato.

L'idea dei container non è nuova ma la si può vedere come una naturale evoluzione dei chroot e delle FreeBSD jails [3] disponibili già da molto tempo.

Esistono varie soluzioni che implementano il concetto di containerizzazione. LXC [4] è sfruttato da varie soluzioni di containerizzazione, e combina i cgroups del

kernel linux e il supporto a differenti namespaces per implementare gli ambienti d'esecuzione isolati.

Con LXC è possibile supportare molte delle funzionalità di isolamento tipiche dei containers (file system, CPU quotas, networking), ma alcune funzionalità come disk quotas e limite dell'I/O sono supportate soltanto parzialmente. Un engine che supporta la creazione di containers Windows è invece Hyper-V Containers [5]. Docker [6], che nelle sue prime versioni si basava su LXC, è oggi la soluzione più popolare di containerizzazione, che fornisce numerose funzionalità avanzate e tool accessori di utilità come Docker Compose [7], che permette di definire e lanciare applicazioni costituite da più containers, e DockerSwarm [8], che fornisce clustering nativo tra Docker engines differenti. In aggiunta, Docker è il sistema di containerizzazione adottato dai principali software di orchestrazione (es., Kubernetes).

2.3 Virtual machine VS Container

Seppur macchine virtuali e container sembrano molto simili tra loro sfruttando gli stessi benefici legati all'isolamento delle risorse, possono essere visti come modi alternativi e distinti per la creazione di un ambiente sul quale esegue un programma software, presentando parecchie differenze legate intrinsecamente al modo scelto per virtualizzare il nodo di computing.

I vantaggi e gli svantaggi principali delle *macchine virtuali* possono essere schematizzati come segue:

- ✓ *Flessibilità*: ogni applicazione ha il suo ambiente di esecuzione specifico (versione kernel, librerie) per il corretto funzionamento di questa;
- ✓ *Esecuzione del software legacy:* la possibilità di utilizzare programmi sviluppati con linguaggi di programmazione non recenti e che non sono sostituibili dati i costi per una nuova implementazione per i sistemi più recenti;
- ✓ *Eccellente isolamento:* l'esecuzione di una virtual machine, in condizioni nominali, non 'disturba' l'esecuzione di altre virtual machine dispiegate su una macchina host poiché l'isolamento è dettato dall'hardware (cpu,

memory);

- ✓ *Live migration:* copia e spostamento di una macchina virtuale su macchine fisiche differenti, anche durante l'esecuzione (migrazione delle macchine virtuali);
- ✓ *Sicurezza:* l'hypervisor responsabile di eseguire le macchine virtuale guest è molto ridotto esponendo una superficie di attacco minore;
- × *Overhead:* occupazione aggiuntiva di memoria (centinaia di MB) e di CPU per l'esecuzione dei sistemi operativi guest;
- × *Upgrade Time*: necessita di configurare e tenere aggiornato ogni istanza del sistema operativo guest;
- × *Booting Time*: decine di secondi o più prima che la macchina virtuale sia up e running non sono accettabili per applicazioni "fire and dye" [10].

Viceversa, i vantaggi e gli svantaggi principali dei *container* sono:

- ✓ *Tempo di avvio più rapido*: con la virtualizzazione del solo sistema operativo i tempi di avvio si riducono notevolmente rispetto all'esecuzione di una virtual machine;
- ✓ *Deployment semplificato*: ogni container include non solo l'applicazione - servizio ma anche tutto il pacchetto utile per eseguirlo, semplificando ogni operazione di deployment e facilitando la distribuzione su differenti sistemi operativi senza ulteriori sforzi di configurazione;
- ✓ *Efficienza operativa:* i contenitori occupano minor spazio su disco rispetto alle VM e hanno un consumo di CPU basso, rendono il sistema più agile, migliorando l'efficienza operativa, lo sviluppo e la gestione delle applicazioni;
- × *Sicurezza:* il sistema operativo che ospita i container ha un numero di librerie maggiore rispetto all'hypervisor per garantire il corretto funzionamento dei container; ne consegue una superficie d'attacco maggiore;
- × *Flessibilità:*il sistema operativo di tutti i container deve coincidere con quello in esecuzione sulla macchina host, dato che il kernel è condiviso; ne

[10] Applicazioni caratterizzate da tempi di vita abbastanza ridotti, designate per il solo soddisfacimento delle richieste in ingresso al seguito delle quali si spengono (dye) per poi ritornare ad essere attivi (fire) non appena un'altra richiesta si presenta.

consegue l'impossibilità di usare un sistema operativo differente per ogni container;

- *Live migration:* a differenze delle virtual machine risulta essere più difficile da attuare poiché necessita dell'identificazione delle zone di memoria usate e proprie del container (che si vuole migrare) in kernel space prima di procedere con la migrazione su un'altra macchina host.

3. WAMS: wide area measurement systems

WAMS si è presentata come una tecnologia emergente per il monitoraggio in tempo reale della Power-Grid e ha svolto un ruolo significativo per garantirne un funzionamento sicuro e stabile.
Una rete WAMS è costruita su un sistema di comunicazione affidabile che collega le centrali elettriche, centri di controllo e sottostazioni dove, tramite l'utilizzo di un sistema di controllo, riesce a garantire la generazione della sola quantità di energia elettrica.
Se domanda e offerta non sono in equilibrio, gli impianti di generazione e le apparecchiature di trasmissione possono spegnersi il che, nei casi peggiori, può portare a un grave blackout regionale.
WAMS consente di avere un controllo più accurato osservando il sistema di alimentazione in modo sincrono in una scala temporale più elaborata.

3.1 Principali dispositivi

I dispositivi dispiegati nei punti nevralgici di una rete elettro-energetica sono:

- Phasor measurement system (PMU)
- Phasor data concentrator (PDC)

Nelle sezioni in seguito verranno descritti i dispositivi fisici coinvolti nella generazione/elaborazione dei dati critici all'interno della Power-Grid.

3.1.1 Phasor measurement system

Nello scenario della power grid attuale, la Phasor Measurement Unit (PMU, IEEE-C37.118.1-2011 [9], [10]) è un IED (*Intelligent Electronic Device*), cioè un

device fisico special-purpose tipicamente equipaggiato con software minimale, specifico dei dispositivi embedded che offre supporto all'esecuzione del processo di acquisizione dati di misura (funzione software potenzialmente mission-critical). Una PMU integra sensoristica on-board collegata a nodi della power grid attuale (quali battery lead/lithium storage, Direct Current supply, R-L-C load, H&H load, MCI, Smart Home TWM, ecc.); le sue misure sono dette "sincrofasori" poiché sono sincronizzate nel tempo grazie ad un sistema di riferimento temporale, solitamente GPS.

Tra le caratteristiche principali delle PMU si hanno:

- elevata frequenza di misura;
- precisione;
- bassa latenza;
- resilienza ad interferenze esterne (ad esempio le armoniche).

Essi vengono installati in maniera distribuita sulle reti elettriche MT e AT per misurare le tensioni nodali e le correnti di linea, ma anche la frequenza e il Rate of Change of Frequency (ROCOF), ossia il tasso di variazione di frequenza nel tempo. Le misure rilevate entrano nel Wide Area Monitoring Systems (WAMS), consentendo un monitoraggio real-time delle condizioni della rete determinandone lo stato di salute.

3.1.2 Phasor data concentrator

A un livello superiore rispetto ai dispositivi PMU attualmente dispiegati nella power grid si trovano i dispositivi Phasor Data Concentrator (PDC, IEEE C37.244-2013 [11]) che, differentemente dal caso del device PMU, vengono eseguiti su dispositivi hardware general-purpose (server tradizionali) come normali processi, spesso su un sistema operativo Windows.
Questi raccolgono i dati provenienti da molteplici PMU, aggregandoli e rendendoli disponibili in maniera più astratta e funzionale a tutte le applicazioni di monitoring e di controllo di più alto livello, quali ad esempio visualizzazione delle

informazioni, allarmi, applicazioni di analisi sofisticate, funzioni di controllo o di automazione, alimentando tutte le applicazioni di monitoraggio e controllo che necessitano di tali dati.

Le serie temporali dei sincrofasori raccolti da un PDC possono essere processate e inviate ad altri PDC come unico data-stream, quindi utilizzate per varie applicazioni quali dashboard (UI visualizzazione di info/allarmi), storage in time-series DB, on-line/off-line analytics, secondo la modalità operativa del PDC stesso (Data Aggregation, Data Forwarding, Data Communications, Data Validation), nonché applicazioni di controllo/automazione della power grid.

In base alla dimensione di quest'ultima, i dispositivi PDC sono organizzati secondo layer gerarchici (PDC locali, PDC corporate, PDC regionali). Generalmente, i PDC locali aggregano e allineano temporalmente i sincrofasori provenienti dalle PMU, inviando i propri data-stream ai PDC corporate che possono effettuare un controllo qualitativo dei dati ricevuti, quindi li inviano ai PDC regionali che inoltrano il data-stream complessivo alle applicazioni suddette.

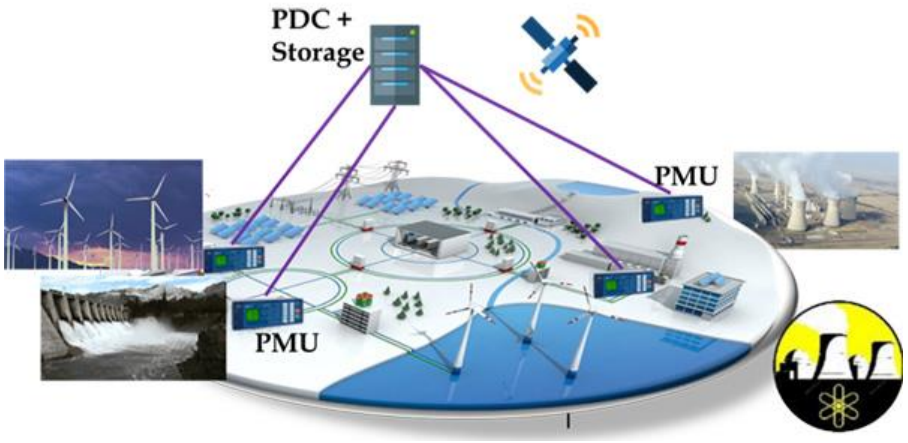


Figura 6: PMUs e PDC per il monitoraggio della power - grid

Di seguito sono indicati i principali task condotti da un dispositivo PDC:

- *Data aggregation*: il PDC allinea temporalmente i dati ricevuti ed invia lo stream di dati aggregati alle applicazioni, ai PDC superiori e al database;
- *Data forwarding*: viene selezionato ed inviato verso i livelli superiori solo un sottoinsieme di dati;

- *Data communications*: la comunicazione del PDC verso le PMU deve seguire quanto prestabilito dallo standard IEEE C37.118.2 in cui si definisce come supporto fisico una linea seriale o la rete Ethernet (sia TCP/IP che UDP/IP);
- *Data validation*: il PDC deve essere in grado di effettuare una validazione dei dati verificando il Time Quality dei sincrofasori provenienti dalla PMU, oltre a verificare l'integrità dei dati attraverso il Cyclic Redundancy Check (CRC). I frame, che possono essere di 4 tipologie (data, configuration, header e command) provenienti dalle PMU (i primi 3) o ricevuti dalle PMU (l'ultimo), oltre a contenere i dati contengono anche informazioni aggiuntive[11].

3.2 Architettura IoT per WAMS

La modernizzazione della rete, ovvero la transizione dalle reti elettriche alle reti intelligenti è possibile tramite l'uso di soluzioni digitali e dispositivi IoT gestiti in maniera automatizzata tramite tecniche di orchestrazione.

Il successo mondiale del concetto di Internet of Things eil numero crescente di piattaforme, dispositivi e reti risulta essere una grande opportunità per il settore elettro-energetico.

Ciò che ha contribuito a rendere attuabile il successo del paradigma IoT è la virtualizzazione del dispositivo (si veda capitolo 2. Virtualizzazione).

Oggi giorno diversi progetti mirano a creare architetture di rete/comunicazione facendo largo uso di dispositivi general-purpose virtualizzati poiché in grado di rendere programmabile e smart l'ambiente nel quale sono dispiegati promuovendo interoperabilità, riutilizzabilità e sopratutto flessibilità dei servizi.

Tramite la virtualizzazione si ha la capacità di estendere le funzionalità e le

[11] Tramite i valori assunti dai bit 14-15 nel campo STAT del Syncrophasor Data Frame (SDF), un PDC desumo che la PMU inviante tale frame è o in test mode (es. inizializzazione, allineamento, ecc.), oppure non ha inviato alcun SDF (es. PMU down), per cui deve interpretare come non validi i dati ivi presenti oppure contrassegnare come non valido il SDF inviato al PDC di livello superiore in cui, allo scadere del wait timer, ha inserito la sequenza di padding.

caratteristiche dei dispositivi riuscendo ad integrare oggetti eterogenei.
Seguendo questa filosofia di pensiero, le appliance dedicate[12], classificate come dispositivi special-purpose hanno iniziato a perdere interesse e di conseguenza valore.
I motivi che hanno contribuito a ridurre l'attrazione nei confronti di questi da parte delle aziende sono:

- *prezzo*: risulta essere sicuramente il fattore principale riconducibile al costo del chip dedicato progettato per la realizzazione del task hardware;
- *mancata assistenza*: evento che si verifica quando il dispositivo va in 'out of date'[13];
- *mancata flessibilità*: un dispositivo special-purpose viene progettato e costruito per la realizzazione di un determinato task senza possibilità di variazione in seguito;

Poiché i dispositivi IoT sono visti come indistinguibili a causa dello stesso hardware di cui dispongono, il focus è posto sui dati che le applicazioni devono generare e processare.
Si arriva all'adozione di un paradigma liquido, in cui il software, responsabile di virtualizzare in maniera dinamica il dispositivo, gioca un ruolo fondamentale per la fornitura del servizio.
Nei capitoli successivi viene spiegato come un uso massivo di tecnologie ICT e di dispositivi IoT virtualizzate orchestrati per gestirne, in maniera automatizzata, l'elevato numero di nodi in una rete elettro-energetico (al posto degli attuali dispositivi embedded) è essenziale per la realizzazione di una Smart-Grid.

[12] Sistemi embedded elettronici di elaborazione a microprocessore progettati appositamente per un determinato utilizzo (special-purpose), ovvero non riprogrammabili dall'utente per altri scopi, spesso con una piattaforma hardware ad hoc, integrati nel sistema che controllano e in grado di gestirne tutte o parte delle funzionalità richieste.
[13] Termine per indicare un dispositivo ormai obsoleto che non ha ormai nessuna manutenzione da parte dell'azienda che l'ha progettato.

3.3 Smart-Grid

Una Smart-Grid è l'output risultante e derivante dalla massiccia distribuzione dei dispositivi IoT e l'utilizzo di tecnologie ICT nelle aree di misurazioni ottenendo un sistema elettrico intelligente e digitalizzato che fornisce elettricità in modo ottimale dalla fonte al consumo consentendo una migliore gestione dell'energia, riducendo al minimo le interruzioni di corrente, trasportando solo la quantità di energia richiesta e notificando ai clienti la loro impronta energetica e il modo per gestirne il consumo.

Ne risulta essere una rete di generazione, trasmissione e distribuzione di energia potenziata da capacità di controllo, monitoraggio e telecomunicazioni digitali.

Senza Smart-Grid, se si verificasse un guasto nella sottostazione locale, questo verrebbe realizzato solo quando il cliente chiama per porre le sue lamentale.

Posizionare un sensore in rete all'interno di un trasformatore o lungo i cavi potrebbe individuare e segnalare un problema o impedire che si verifichi in primo luogo.

Inoltre, tramite l'utilizzo di tecnologie di orchestrazione che controllano e gestiscono il ciclo di vita dei nodi il guasto viene notificato e risolto, se possibile, in maniera automatizzata e trasparente.

Gli operatori di sistema, di conseguenza, possono dedicare maggior tempo all'analisi della rete limitandosi a monitorarne lo stato tramite strumenti software di orchestrazione che ne riflettono lo stato in tempo reale.

3.4 Implementazione differenti dei principali dispositivi

Questa sezione analizza le caratteristiche delle applicazioni e le possibili strategie implementative dei principali dispositivi coinvolti (PMU/PDC) in ambito elettro-energetico.

Viene inoltre definito il *Componente funzionale critico (CFC)* che tornerà utile ai fine delle valutazioni degli eventi ICT rilevanti.

Il CFC è spiegato nel modo seguente:

- *Fisico (CFC Fisico):* dispositivo fisico embedded special-purpose / appliance dedicato, attualmente in produzione (es. device National Instruments nel caso PMU) che integra in modo inscindibile la componente hardware e software; in particolare, esegue la funzione mission-critical di misurazione/controllo PMU/PDC come un usuale processo software.
- *Virtualizzato (CFC Virtualizzato)*: funzione software mission-critical di misurazione/controllo PMU/PDC virtualizzata in container Linux (la funzione containerizzata viene eseguita come "main process" del container stesso) in esecuzione su un dispositivo host hardware general-purpose (detto *nodo di computing*). Sotto alcune condizioni, questa soluzione è in grado di disaccoppiare il dispositivo host fisico (hardware general-purpose che offre supporto di computing alla virtualizzazione) dalla funzione software containerizzata che deve essere eseguita, permettendo (manualmente) di rilocare la funzione software su host diversi.
- *Virtualizzato ed Orchestrato (CFC Orchestrato):* funzione software mission-critical di misurazione/controllo PMU/PDC virtualizzata in container Linux (la funzione containerizzata viene eseguita come "main process" del container stesso), poi incapsulato entro un Pod per essere orchestrato da un orchestratore, che lo pone in esecuzione grazie al supporto di processing offerto da un dispositivo host hardware general-purpose (detto *nodo di orchestrazione*). Sotto alcune condizioni, questa soluzione è in grado di disaccoppiare il dispositivo host fisico (hardware general-purpose che offre supporto di computing a virtualizzazione e orchestrazione) dalla funzione software containerizzata che deve essere eseguita. Questo permette di rilocare automaticamente la funzione software su host diversi grazie alla possibilità di essere eseguito come "pod", di essere automaticamente posta in esecuzione su uno qualunque dei nodi del cluster grazie al componente di orchestrazione.

3.5 Analisi di eventi ICT critici

Al fine di evidenziare le possibili differenze tra una implementazione del servizio attraverso l'utilizzo di dispositivi fisici dedicati (scenario "classico"), oppure attraverso la loro virtualizzazione e la loro esecuzione su piattaforme dotate di supporto alla virtualizzazione, vengono analizzati una serie di eventi critici che causano l'interruzione del servizio (es. si verifica che il componente funzionale ha un guasto sw/hw critico) che sono stati presi a riferimento per valutare le migliorie apportate dall'adozione di una Smart-Grid.

In questo caso, verranno distinti il caso in cui le piattaforme virtualizzate sono "stand-alone" oppure sono "orchestrate" da un sistema esterno.

Allo scopo di identificare i potenziali vantaggi della virtualizzazione e successivamente dell'orchestrazione, nell'analisi dei suddetti eventi vengono distinti tre casi:

- Il cfc è in esecuzione su un dispositivo fisico dedicato;
- il cfc è in esecuzione all'interno di un ambiente virtualizzato (nello specifico, containerizzato);
- il cfc, oltre ad essere virtualizzato, è gestito, configurato e coordinato in maniera automatica da un orchestratore.

3.5.1 Il CFC ha un guasto hw/sw critico

Viene qui preso in considerazione il caso in cui il CFC Fisico smetta di operare (ad esempio, a causa di crash del SO embedded a bordo del device, di malfunzionamento hardware o di alimentazione, ecc.). In tal caso è necessario rendere nuovamente operativo l'intero componente funzionale, considerato come unicum inscindibile di embedded device special-purpose in cui è integrato il software (in particolare nel caso di PMU). Ciò può essere fatto, ad esempio, effettuando manualmente il troubleshooting che porti a stabilire la causa origine del down-time e, quindi, a ripristinare (ad esempio forzare il reboot del dispositivo) o sostituire in toto (sempre manualmente) il CFC Fisico stesso.

Nel caso di CFC virtualizzato, esso, ad esempio, può aver subìto un crash del

proprio “main process” interno (software PMU/PDC), per cui il suo ripristino può avvenire più agilmente ricreandolo e allocandolo manualmente sullo stesso dispositivo host di supporto alla virtualizzazione (es. sullo stesso nodo di computing).

Nel caso di CFC orchestrato, il container può aver subito un crash del proprio “main process” interno (software PMU/PDC), per cui il ripristino operativo del componente funzionale (inteso come Pod incapsulante il container con il software PMU/PDC) è effettuato automaticamente dall’orchestratore che, senza alcun intervento umano, provvede a ricrearlo, schedularlo e allocarlo o sullo stesso nodo di computing di orchestrazione, oppure su un altro eventualmente disponibile alla schedulazione (in base a metriche proprie dell’orchestratore stesso, ad es relative alle risorse CPU/RAM libere), fatto salvo il limite relativo all’eventuale dipendenza da periferiche hardware (ad es. PMU).

La seguente tabella schematizza la gestione dell’evento in questione, identificando vantaggi e svantaggi delle principali soluzioni.

	Fisico	**Virtualizzato**	**Orchestrato**
Impatto del guasto	• Irreversibile, necessario intervento umano manuale in loco per ripristino.	• Reversibile, ma è comunque necessario un intervento umano (da remoto) per il reboot sul nodo di computing.	• Reversibile, non necessario alcun intervento umano per reboot e re-allocazione delle risorse.
PROs	• Semplicità di ripristino nel caso di sostituzione del device hardware.	• La funzione containerizzata può essere ricreata manualmente da remoto.	• L’orchestratore rialloca ed esegue automaticamente la funzione containerizzata dove più opportuno. • Reazioneimmediata.
CONs	• Reattività imposta da latenze umane. • Necessario un operatore sul	• Reattività imposta da latenze umane.	

	campo per il ripristino.		

3.5.2 Il CFC non è più raggiungibile

Si considera il caso in cui il CFC Fisico sia correttamente operante, ma per un problema di rete esso non sia più raggiungibile attraverso la rete dati della SG. Non conoscendo a priori tale causa, è necessario avviare procedure di ricerca guasti sulla rete dati (switch, router ecc.) che interconnette il dispositivo, in tutta la tratta fino al dispositivo stesso.

Questo evento ha una gestione sostanzialmente simile al caso precedente, con i relativi vantaggi e svantaggi. Dal punto di vista dell'infrastruttura di rete, sia nel caso di tecnologie tradizionali (routing) che nel caso di SDN, è possibile verificare l'esistenza di un percorso secondario per raggiungere il CFC.

I vantaggi e svantaggi delle principali soluzioni relativamente a questo evento sono schematizzati nella tabella precedente.

3.5.3 Il CFC è operativo, ma si comporta in modo anomalo

Questo evento considera il caso in cui il componente funzionale PMU in esecuzione sul device embedded special-purpose riporta un errore di funzionamento dell'hardware sottostante (es. fault ai sensori on-board e/o all'hardware di processing/microcontrollori[14] che non consentano di produrre misure integre delle metriche di interesse). Tale evento è accidentale/asincrono (es. è un guasto hardware, crash della funzione software, ecc. fin qui visti) ma, rispetto agli altri, presenta la peculiarità di implicare la continuità operativa del

[14] L'errore potrebbe avvenire nella parte di processing, es. microcontrollori o periferici a bordo IED/Raspberry-Pi, e/o nella sua parte "passiva" di sensoristica off-board, es. sensore di tensione connesso al bus GPIO del Raspberry-Pi.

componente funzionale, ossia questo componente è raggiungibile via rete e potenzialmente genera un flusso dati apparentemente valido.

Infatti, a differenza di tutti gli altri eventi discussi e analizzati in precedenza, quello qui analizzato non causa l'interruzione della funzionalità operativa (es. esecuzione) del componente funzionale PMU (cioè non causa un'interruzione del flusso dati di misura (es. sincrofasori) causata invece da un possibile crash del componente), che continua a essere operativo, rendendo tale evento molto più subdolo rispetto agli altri eventi asincroni esaminati in quanto non rilevabile semplicemente monitorando il numero di bytes in uscita dal dispositivo.

Un possibile meccanismo di rilevazione di tale anomalia richiede importanti personalizzazioni delle implementazioni delle tecnologie di virtualizzazione e, soprattutto, di orchestrazione, in quanto è necessario entrare nella semantica del protocollo applicativo e rilevare un'anomalia dei dati trasmessi. Concettualmente, il problema è analogo alla verifica di operatività di un server web: un meccanismo protocol-agnostic (ad es. un testing basato sulla possibilità di aprire una nuova sessione TCP) permette di rilevare il fatto che il web server è ancora attivo (risponde alle richieste di apertura di una connessione TCP), ma non permette di capire che il web server sta ritornando un errore (es "HTTP 500 – Server error" anziché "HTTP 200 – OK"). Per la verifica del secondo aspetto, è necessario una sonda in grado di interpretare i codici di risposta a livello applicativo, ossia una sonda *HTTP-aware*.

In presenza di un simile meccanismo di rilevazione, ad esempio mediante tecnica di *snooping* a livello applicativo sugli SDF stessi oppure integrati all'interno del container PDC, eventuali alert così generati potrebbero essere da esso inviati a un control center remoto in cui si provveda manualmente a istanziare un altro container PMU su un dispositivo host di computing adiacente (alternativo a quello soggetto ai possibili fault hardware accennati sopra), quindi ridondato, che disponga di sensoristica connessa alle stesse linee/nodi di potenza della SG.

È dunque evidente come i comportamenti operativi dei componenti funzionali PMU appena analizzati implichino complessità implementativa molto elevata sia sul piano della virtualizzazione, sia sul piano dell'orchestrazione. Infatti, l'analisi condotta conduce alla considerazione paradossale secondo cui potrebbe essere

meglio se il componente funzionale PMU subisse il crash (interrompendo la propria esecuzione) rispetto al caso in cui invii al componente funzionale PDC synchrophasor data-stream marcati come non validi (es. SDF marcati sul bit 14 del campo STAT che indichino malfunzionamento), in quanto il flusso dati è comunque presente e un orchestratore *vanilla* (generico, non opportunamente customizzato secondo le complesse feature viste) non rileverebbe alcuna anomalia. Pertanto, si dovrebbe agire su un'importante customizzazione sia del software che implementa il componente funzionale PDC virtualizzato che riceve i dati di misura dal componente funzionale PMU (snooping applicativo), sia dell'orchestratore (algoritmo di scheduling).

I vantaggi e svantaggi delle principali soluzioni relativamente a questo evento sono schematizzati nella tabella seguente.

	Fisico	**Virtualizzato**	**Orchestrato**
Classificazione	• Irreversibile, necessario intervento umano manuale in loco.	• Reversibile, necessario intervento umano manuale (in loco/in remoto) per ripristino della funzione software (es. PMU) su computingnode alternativo. Necessario in seguito intervento umano manuale in remoto (o in loco) per verifiche più approfondite su stesso computingnode soggetto a malfunzionamento per indagarne le cause e risolverlo (troubleshooting approfondito).	• Reversibile, non necessario alcun intervento umano per ripristino della funzione software (es. PMU) su computingnode alternativo. Necessario in seguito intervento umano manuale in remoto (o in loco) per verifiche più approfondite su stesso computingnode soggetto a malfunzionamento per indagarne le cause e risolverlo (troubleshooting

			approfondito).
PROs	• Semplicità di ripristino nella sostituzione del device hardware/IED.	• La funzione software (es. PMU) viene manualmente eseguita (provisioning, scheduling ed esecuzione) sul computingnode/deviceIoT scelto dall'operatore umano come alternativo a quello soggetto a fault (agilità di risposta al guasto).	• La funzione software (es. PMU) viene automaticamente eseguita (provisioning, scheduling ed esecuzione) sul computingdevice più opportuno – "controllo adattivo" (velocità di risposta automatica al guasto). • Automazione completa dei task di rescheduling (provisioning, allocazione ed esecuzione su computingnode più opportuno – "controllo adattivo"). • Reattività secondo time-scale (latenze) computing.
CONs	• Un device ridondato deve essere manualmente avviato dall'operatore ASAP affinché un componente funzionale alternativo a quello guasto	• Necessario implementare a livello applicativo/container il monitoring continuo automatico di healthcheck del componente funzionale (controllo dell'applicazione	• Necessario implementare a livello applicativo/container il monitoring continuo automatico di healthcheck del componente funzionale

	produca nuovamente dati utili. • Monitoring hardware difficile su embedded device. • Reattività secondo time-scale (latenze) umane.	a livello semantico). • Stop manuale della VM (*full-virt*) o kill del container (*light-virt*) nel computingnode/deviceIoT soggetto a malfunzionamento e, successivamente, reboot della VM o respawn del container manuali sul computingnode/deviceIoT alternativo. • Allocazione risorse su nodo di computing non adattiva.	(controllo dell'applicazione a livello semantico).

4. Strumenti utilizzati

In questo breve capitolo verranno spiegati gli strumenti utilizzati per la realizzazione del lavoro di tesi sperimentale. È opportuno esaminare tali strumenti prima della discussione dei dettagli implementativi.

4.1 Docker

Questa tecnologia, che nelle sue prime versioni si basava su LXC, è oggi la piattaforma software più popolare open source di containerizzazione che permette di creare, testare e distribuire applicazioni con la massima rapidità creando un lightweight, portable e self contained package.
Docker raccoglie il software in unità standardizzate chiamate container (si veda 2.2.2 Light virtualization: Container) che offrono tutto il necessario per la loro corretta esecuzione, incluse librerie, strumenti di sistema, codice e runtime che, indipendentemente dall'ambiente di esecuzione, può essere eseguito ovunque. L'applicazione containerizzata dunque assume sempre lo stesso comportamento di esecuzione su qualsiasi host senza essere vincolata alle caratteristiche di sistema quali versione kernel e/o distribuzione nella quale viene lanciata.
Di seguito sono elencate le principali features aggiuntive che Docker offre rispetto alla soluzione LXC e che risultano rilevanti per l'implementazione descritta in questa sezione.

- *Application portability:* come detto in precedenza, Docker permette di creare/distribuire ed eseguire self contained package, incapsulante l'applicazione da eseguire, su qualsiasi dispositivo di computing in maniera indipendente dall'ambiente di esecuzione;
- *Union file system:* considerato che, i container replicano l'intero file system necessario per la corretta esecuzione, dunque, l'image-size del container stesso può essere elevata (es. 100 MB o più), l'avvio del container potrebbe causare un ritardo non trascurabile (download time di 100MB è circa 1

secondo in una 1Gbps network). Tramite una visione stratificata del file-system, Union File System (es. overlayFS) permette a più container di condividere tra loro il loro base-file system (se già presente sul dispositivo di computing) evitando così di scaricare l'intera immagine; procedendo, solamente i file non presenti verranno scaricati e andranno ad affiancare, al top, lo shared-file system differenziando l'ambiente di un container da un altro e riducendo i tempi di download in maniera notevole.

- *Automatic build:* Docker permette, in maniera automatica, di creare un container personalizzato partendo dalla specifica dei suoi componenti base (es. specifica del SO, download di determinate librerie) e delle azioni da eseguire, tramite la stesura di un Dockerfile. Esso è uno YAML file assimilabile ad una ricetta ben strutturata, composta da più steps scritti dal developer che, se condotti bene e in maniera sequenziale nel tempo, creano il container desiderato.
- *Integration con Kubernetes:* Docker è il sistema di containerizzazione adottato dai principali software di orchestrazione (es., Kubernetes) usato per creare ed eseguire le applicazioni desiderate in container incapsulati in Pod.
- *Isolation:* come avviene anche per LXC, l'isolamento di un Docker container con eventuali altri Docker container dispiegati nell'intero cluster è una feature consolidata e funzionante (anche se deployati sullo stesso nodo di computing).
- *Bootspeed*: tramite docker engine, la procedura di pull (download) dell'immagine di un docker container da un repository (es. dockerHub[15]) e l'esecuzione stessa è semplice e veloce.

4.1.1 Networking

Un aspetto importante di Docker è sicuramente il networking.

Nel momento in cui si manda in esecuzione un container, è possibile selezionare

[15] DockerHub è un servizio fornito da Docker per condividere containers-image. È il più grande repository al mondo comprendente progetti open source e fornitori che creano e distribuiscono il loro codice in containers.

tramite il comando *–network* quale rete utilizzare [12]. Le possibili configurazioni di rete sono:

- *bridge:* è la rete di default. Se non viene usata la keyword --network, il container viene instanziato nella rete 172.17.0.0/16 associata all'interfaccia virtuale docker0. Conoscendo l'IP di un altro container, è possibile comunicare con esso;
- *none:* quando un container viene collegato a questa rete non riceve alcun indirizzo IP se non quello di loopback. Non può accedere alla rete esterna e a nessun altro container. In pratica viene totalmente isolato;
- **host**: permette ai container ad essa collegati di condividere tutto lo stack network dell'host. Tutte le interfacce dell'host saranno quindi visibili dal container;
- *custom*: è possibile creare delle nuove reti (tramite docker volume create <network_Name>), assegnandone nomi e range di indirizzi personalizzati. Tramite il server DNS interno che il demone di Docker lancia alla creazione della rete custom definita dall'utente, i container che vi vengono instanziati possono comunicare tra loro tramite il nome oltre anche agli indirizzi IP. Se la risoluzione dei nomi non va a buon fine con il server DNS locale, allora vengono contattati direttamente i server DNS esterni.

È possibile, inoltre, esporre una porta custom per garantire la raggiungibilità del docker container, tramite:

- *EXPOSE*: è opzionale e serve ai fini di documentazione. Viene inserita nel Dockerfile (file utile per la creazione di un container) per indicare quali porte pubblicare per raggiungere il container;
- *PUBLISH*: utilizzando "publish" (o la sua forma contratta –p) è possibile demandare direttamente a Docker l'apertura delle porte richieste nel momento in cui viene istanziato il container. A differenza della keyword EXPOSE, questa non è opzionale. È possibile richiedere il mapping con una porta specifica dell'host tramite -p P1:P2 affinché il traffico in arrivo sulla porta P1 dell'host venga reindirizzato verso la porta P2 del docker container dispiegato nella propria network.

4.1.2 Volume

La possibilità di specificare un volume alla creazione del docker-container permette di rendere persistente i dati generate o condividere variabili tra essi [13] .
È possibile creare un volume tramite la il comando docker volume create <Volume_Name>.
Tramite la direttiva –volume (o –v nella sua forma contratta) è possibile passare un path composta da 3 campi (x:y:mod), seperati da due punti al docker container che si vuole eseguire.
I campi devono essere nell'ordine corretto e il significato di ogni campo non è immediatamente evidente.
Nel caso di volumi denominati, il primo campo è il nome del volume ed è univoco su una determinata macchina host. Per i volumi anonimi, il primo campo viene omesso.
Il secondo campo è il percorso in cui il file o la directory sono montati nel contenitore.
Il terzo campo è facoltativo ed è un elenco di opzioni separate da virgole, come ro (facoltativo).

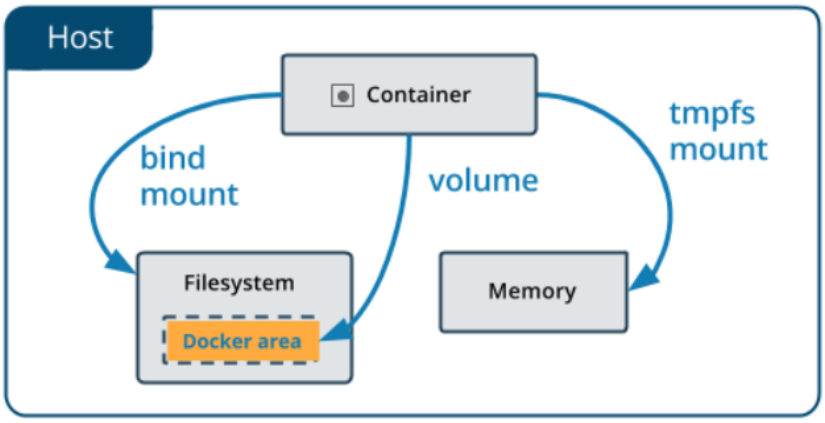


Figura 7: illustrazione di un volume-mount

4.2 Docker-compose

Compose è uno strumento per la definizione e l'esecuzione di applicazioni basate su containers [14].

Tramite la stesura di un file YAML ben strutturato è possibile, con un solo comando, configurare ed eseguire i servizi dell'intera applicazione.
In modo analogo alla definizione di un'applicazione docker tramite la creazione di un dockerfile, un file docker-compose.yml definisce i servizi che possono essere eseguiti insieme in un ambiente isolato.
Compose funziona in tutti gli ambienti: produzione, gestione temporanea, sviluppo.
Un file compose ha una struttura definita nel quale è possibile definire una lista di servizi che verranno lanciati sotto forma di docker-container. È possibile personalizzare il singolo servizio tramite una serie di keywords specificate in sequenza (image, container_name, networks, volumes etc.) che andranno a differenziare un docker-container da un altro in base all'utilizzo che se ne vuole fare.
Tramita la direttiva docker-compose up è possibile eseguire ed applicare l'intero file con formato yaml; verrà fatto il parsing e creato quindi l'ambiente di sviluppo atteso. Viceversa, il comando docker-compose down provvederà a pulire l'ambiente partendo dalla rimozione dei singoli docker-container e delle reti nel quale essi sono dispiegati.

5. iPDC software

Per la realizzazione del test-bed emulante una piccola Smart-Grid è stato scelto un software legacy iPDC [15] contenente i sorgenti scritti in linguaggio di programmazione C.

iPDC software è un pacchetto contenente tre differenti moduli:

- iPDC: applicativo riproducente un PDC conforme allo standard IEEEC37.118;
- DBserver: applicativo usato come logger e per il salvataggio dei dati verso un MYSQL database;
- PMU simulator: applicativo responsabile di emulare il funzionamento di una PMU confome anch'esso allo standard IEEEC37.118.

A differenza del caso reale, il software riproducente il funzionamento della PMU non genera il syncrophasor-data-stream tramite la lettura di sensoristica on-board ma emula tale comportamento generando, in software, il flusso di dati che verrà poi inoltrato verso il PDC.

Invece, dato che un PDC nel caso odierno, è rappresentato come un normale processo spesso operante su sistema operativo Windows, l'applicativo iPDC riproduce in modo fedele il funzionamento di un dispositivo di tale tipo.

Dal paragrafo successivo, gli applicativi iPDC e PMU simulator verranno indicati con i nomi compressi PDC e PMU.

5.1 Standard IEEEc37.118

È lo standard ampiamente adottato tra i maggiori vendor in ambito elettro-energetico.

Lo standard comprende 4 tipi di frame, i quali sono:

- Command frame (CMD – frame): frame contenente il comando usato dai PDC per richiedere una determinata azione verso le PMU sottostanti o anche dai PDC di ordine superiore (e.g. corporazione) verso i PDC di più basso livello.

E' usato essenzialmente per richiedere la configurazione (CFG – frame) di un PDC/PMU o per abilitare/disabilitare la trasmissione dei dati (Data – frame);

- Configuration frame (CFG – frame): frame contenente le informazioni di configurazione di una PMU come valori di fase, valori digitali e analogici e frequenza;
- Data frame (Data – frame): frame mandato dalle PMU versi i PDC o anche da quest'ultimi versi i PDC di ordine superiore. Contiene i dati di fase, analogici e lo stato degli input digitali di ogni canale configurato in precedenza.
- Header frame: frame in formato ASCII contenente le informazioni di una PMU, le sorgenti dei dati, gli algoritmi di scaling e altre informazioni.

5.2 Comunicazione tra PMU e PDC

I PDC possono comunicare con le PMU o con altri PDC tramite il protocollo TCP o UDP. L'unica differenza tra i due protocolli è la qualità della comunicazione: il protocollo UDP, infatti, risulta essere meno affidabile rispetto al suo duale TCP, il quale richiede una banda maggiore per l'instauro della connessione.
Ipotizzando di avere l'applicativo PDC installato su una macchina e l'applicativo PMU su un'altra, la comunicazione con protocollo TCP tra questi avviene come illustrato in Figura 8: comunicazione TCP tra PMU e PDC.

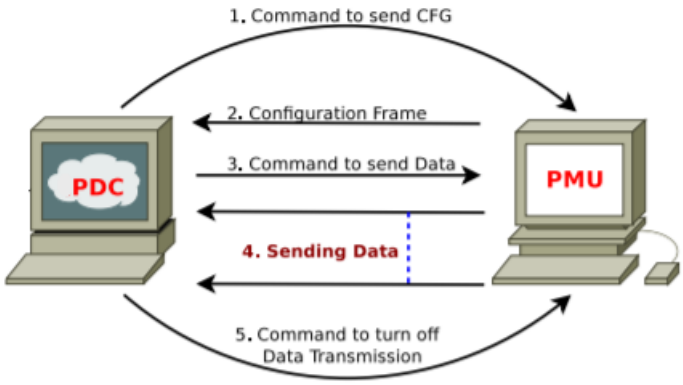


Figura 8: comunicazione TCP tra PMU e PDC

Nel dettaglio, la comunicazione può essere suddivisa in tanti step dove PMU e PDC si scambiano i comandi dello standard IEEEC37.118 nel seguente modo:

- Il PDC manda un CMD - frame alla PMU per richiedere la sua configurazione (CMD – frame del tipo CFG – frame) ovvero il numero di fasori, il numero dei canali analogici e digitali di cui dispone;
- La PMU risponderà al comando mandando la sua configurazione;
- Successivamente alla ricezione della configurazione da parte della PMU, il PDC richiederà a questa l'inoltro dei dati che i vari canali stanno simulando (CMD – frame del tipo Data – frame);
- La PMU allora inizierà a mandare i dati verso la PDC richiedente;
- Se il PDC nota un cambiamento analizzando i dati (bit 13 e 14 del campo STAT) mandati dalla PMU, manderà a questa un comando CFG per ricevere i nuovi parametri di configurazione;
- A seguito della ricezione della nuova configurazione, il PDC inizierà nuovamente a ricevere i dati dalla PMU avendo prima mandato il comando di Data – frame.

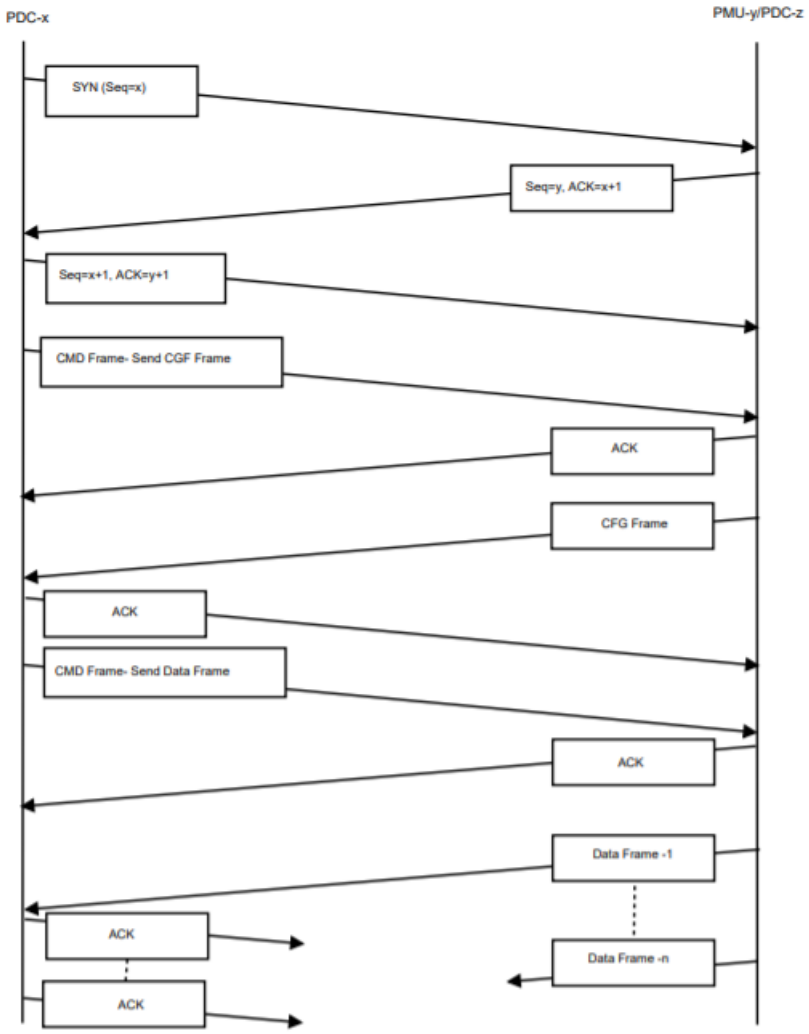


Figura 9: comunicazione TCP

Infine, l'applicativo DB server è invece usato per il logging del sincrophasor-data-stream che un PDC riceve dalle PMU sottostanti e viene raggiunto tramite protocollo UDP sulla porta 9000.

Tale applicativo risulta utile poiché non solo è responsabile di loggare a video il flusso dei dati ricevuti dalle macchine PMU sottostanti ma ha il compito di interfacciarsi con un DBMS come MySQL per la memorizzazione persistente dei dati.

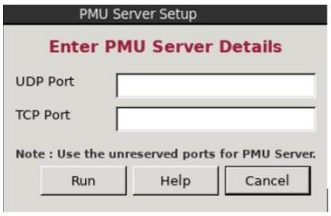
5.3 Configurazione e setup della comunicazione

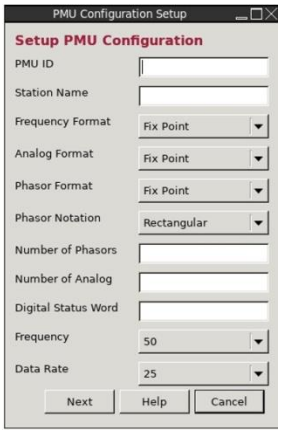
In questo paragrafo vengono illustrate le principali azioni rese possibili dalla GUI degli applicativi del pacchetto iPDC software spiegando il loro utilizzo per l'instauro della connessione.

5.3.1 PMU

A seguito del lancio dell'applicativo PMU, la graphical user interface si presenta con una serie di opzioni necessari per la sua configurazione.

PMU server/configuration setup: prima finestra che appare chiedente all'utente l'inserimento delle informazioni iniziali di setup come numero di porta (UDP e TCP) per la ricezione dei command-frame (si veda paragrafo 5.1 Standard IEEEc37.118) mandati dai PDC.
Viene successivamente chiesto di inserire l'id-code per l'identificazione in modo univoco del dispositivo, il nome della stazione nel quale esso opera, il numero di fasori, dei canali analogici e infine digitali.





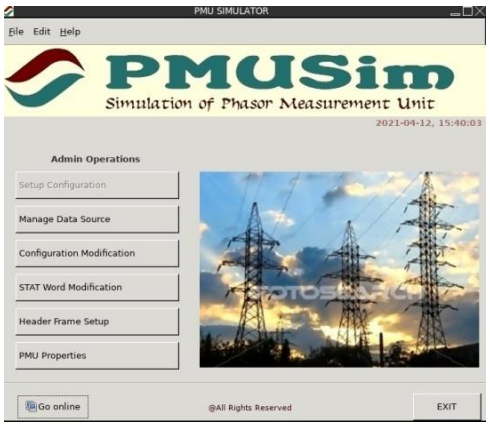
Manage data source: tramite questa opzione l'utente ha la possibilità di scegliere tra una generazione simulata dei dati o il caricamento di essi da un file CSV.
*Configuration Modification:*opzione utile per permettere la modifica del frame di

configurazione aggiungendo o rimuovendo canali.

STAT word modification: l'utente ha la possibilità di cambiare i bit del campo STAT.

Header frame setup: l'utente può aggiungere informazione di dettaglio circa gli algoritmi di scaling usati, di filtraggio e altre informazioni riguardanti la PMU.

PMU properties: a seguito dell'interazione con tale opzione vengono mostrati all'utente i dettagli della PMU appena configurata.

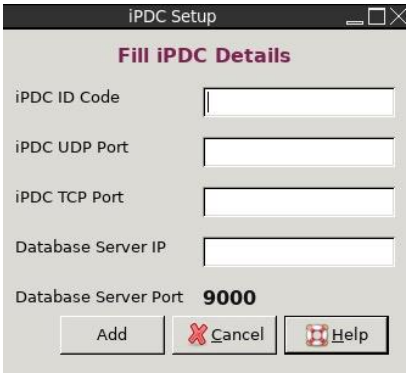


5.3.2 PDC

La graphical user interface di un PDC si presenta in maniera quasi analoga con un primo pop-up necessario per la configurazione e una serie di opzioni per eseguire le funzioni basi.

Enter PDC setup: pop-up necessario per la configurazione del PDC. Chiede all'utente l'inserimento del codice identificativo di tale dispositivo, le porte UDP e TCP usate per ricevere i command-frame provenienti dai PDC di livello superiore e l'indirizzo dell'applicativo DB server, il quale agisce da logger stampando a video il flusso dei dati ricevuti e si adopera a memorizzare in maniera persistente questi verso un DBMS come MySQL.

La porta scelta per la comunicazione con il logger è la 9000 di default con protocollo UDP.



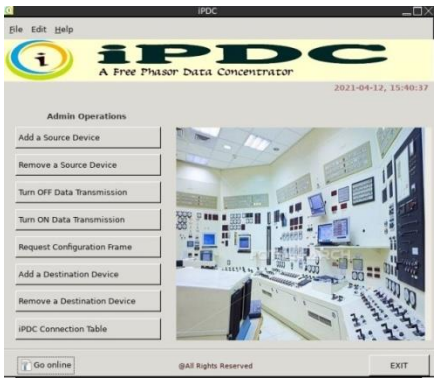
Add/Remove a source device: tramite questa opzione l'utente ha la possibilità di inserire /eliminare una sorgente dati (PMU o PDC di livello superiore nel caso di interazione PDC-PDC) precedentemente configurata.

Viene chiesto di inserire l'id-code della sorgente dati, il suo indirizzo IP, la porta e il protocollo di trasporto per la corretta interazione.

Turn OFF/ON the data transmission: l'utente ha la possibilità di vedere tutte le sorgenti dati presenti nel sistema e mandare i comandi di start/stop trasmissione (CMD – Frame).

Add/Remove a destination device: con questo comando l'utente può inserire/ rimuovere una sorgente dati di ordine superiore inserendo il suo indirizzo IP e il protocollo di trasporto usato. Non viene chiesto nessun inserimento di porta poiché viene considerata la porta di default inserita in fase di configurazione (PDC setup).

iPDC connection table: mostra le sorgenti di dati inferiori (PMU/PDC) e superiori (PDC) con le quali il dispositivo sta comunicando.



6. Architettura ed implementazione

In questo capitolo viene spiegato come i principali componenti (si veda 3.1 Principali dispositivi) presenti in una power-grid sono stati virtualizzati, resi quindi delle funzioni software OT mission-critical di misura il cui supporto di computing è offerto da hardware general-purpose a basso costo.
Lo scopo del lavoro di ricerca è la realizzazione di una piccola Smart-Grid composta da PMU e PDC rappresentate come funzioni software eseguenti all'interno di containers e testare quindi le loro comunicazioni in ambiente virtualizzato.

6.1 Architettura generale

Per la realizzazione del test-bed emulante una piccola Smart-Grid si è fatto uso degli applicativi del pacchetto software iPDC (si veda capitolo 5. iPDC software).
Nell'implementazione condotta, tali applicativi sono stati dunque containerizzati (si veda 2.2.2 Light virtualization: Container) per riprodurre l'infrastruttura voluta costituita dai seguenti componenti:

- PMU: funzione software mission-critical di emulazione delle suddette grandezze che viene virtualizzata attraverso un container dispiegato all'interno di uno dei nodi rappresentati l'infrastruttura voluta;
- PDC: soluzione virtualizzata (tramite virtualizzazione leggera) per l'applicativo iPDC che consente di creare i servizi applicativi operanti su hardware general-purpose (es. Nebbiolo Tech Fog node, "white label" server et similia).
- DB server: applicativo anch'esso virtualizzato tramite virtualizzazione leggera;
- MySQL: ulteriore container eseguente l'immagine di un DBMS per la memorizzazione persistente dei dati ricevuti da un PDC.

Nei paragrafi successi verrà spiegato come avvengono le interazioni tra i componenti funzionali appena elencati e come vengono eseguiti gli applicativi PMU/PDC che dispongono di una Graphical User Interface all'interno di containers.

6.2 Implementazione

È stato ripetuto nei precedenti capitoli che gli applicativi messi a disposizione dal pacchetto iPDC software presentano una GUI e quindi necessitano di un ambiente grafico per la loro corretta visualizzazione ed esecuzione.
La soluzione adottata per questa tesi di ricerca tende verso l'utilizzo di tre container ausiliari eseguenti nello stesso network namespace responsabili di fornire un ambiente grafico.
Nel dettaglio, questi container che per semplicità identifichiamo con l'acronimo triade grafica, sono identificati da:

- *X VNC server:* container che crea X VNC con FluxBox (window manager), fornendo l'ambiente grafico di base. Tiger VNC è stato scelto per l'implementare dell'X VNC server.
  La sua immagine esegue:
    - *X server*: permette di disegnare sul display gli elementi grafici abilitando l'interazione del mouse e della tastiera e la possibilità di muovere le finestre;
    - *VNC server*: desktop grafico che permette di trasferire mouse e tastiera da un computer ad un altro, trasmettendo le interazioni tra questi attraverso una rete.
- *WebSockify*: container responsabile di incapsulare il protocollo usato dal VNC server in un websocket e inoltrare il traffico verso un browser per il suo rendering e viceversa, le interazioni fatte su browser vengono incapsulate in socket dati per essere poi inoltrati e consumati dal VNC server.
- *noVNC client:*container riproducente l'immagine di un web server come nginx richiesto quando un client inizia le sue connessioni verso il desktop

remoto. Viene utilizzato per trasformare il VNC server all'interno di un protocollo web-based abilitando cosi l'utente (client) tramite l'utilizzo di browser, di connettersi alla porta ove il remote desktop server è in ascolto e mostrare dunque il display remoto.

La triade grafica descritta in precedenza è eseguita all'interno dello stesso network namespace e gli applicativi da renderizzare quali PDC/PMU in versione containerizzata condividono con l' X VNC server un socket per la loro interazione.

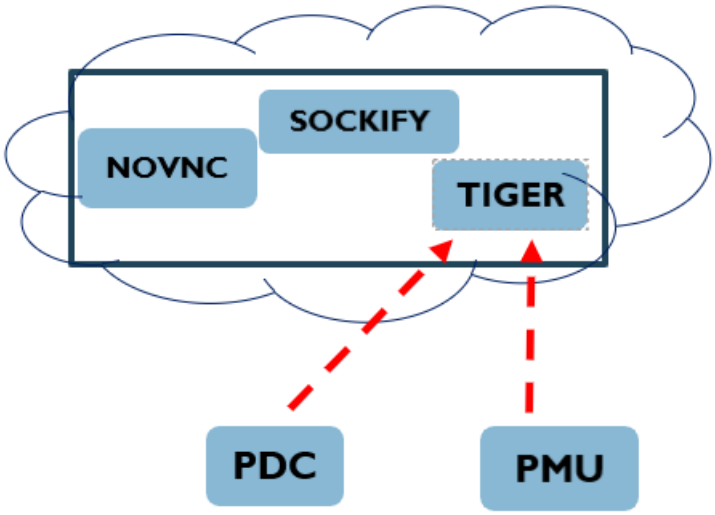


Figura 1: Rendering applicazioni containerizzate PMU/PDC

Gli applicativi PMU/PDC sono eseguiti nel contesto del server grafico X VNC server mostrato tramite il web server (nging) noVNC.

6.2.1 Creazione immagine: Dockerfile

Per realizzare un dispiegamento dei processi software sopra elencati all'interno di docker-container, sono stati creati dapprima i rispettivi dockerfile e, successivamente, tramite un tool di automazione quale Docker Compose, sono state lanciate le esecuzioni dei container che istanziano tali immagini.

Tramite Docker si possono infatti costruire immagini automaticamente leggendo le istruzioni da un dockerfile.
Un dockerfile è un documento di testo in formato YAML assimilabile ad una ricetta ben strutturata che contiene tutti i comandi che un utente può eseguire sulla command line per assemblare un'immagine. Tramite la build di docker un utente può creare una build automatica che esegue parecchie istruzioni in successione.
Lo scopo della build è la creazione di un template immutabile che andrà a contenere librerie, strumenti di sistema, codice e runtime per l'esecuzione di un container.
Nello specifico, i dockerfile creati che permettono di virtualizzare i componenti software di interesse per la creazione del test-bed risultano essere i seguenti:

- X VNC, web Sockify, noVNC client rappresentanti i dockerfiles della triade grafica per il rendering degli applicativi containerizzati PMU/PDC;
- PDC dockerfile per la creazione dell'immagine PDC;
- PMU dockerfile per la creazione dell'immagine PMU;
- DB dockerfile per la creazione dell'immagine DB.
- MySQL dockerfile-custom basato sull'immagine di MySQL disponibile su dockerHub ma personalizzata al fine di creare al suo interno le tabelle necessarie per la persistenza dei dati ricevuti da un PDC.

Una volta ottenute le apposite container-image, si è creata l'infrastruttura di rete virtualizzata nella quale dispiegare i container che assolvono le funzioni software di controllo e acquisizione dati (PDC, PMU e DB), quindi testare la corretta interazione tra essi.
Tramite l'utilizzo di Docker Compose è stato allora possibile creare ed eseguire in maniera automatizzata tali Docker container.

6.2.2 Esecuzione tramite Docker-compose

Esattamente come ci si aspetta in uno scenario reale, ogni Docker container che esegue il software PMU è dispiegato in una rete virtuale propria ed è incaricata di simulare l'acquisizione delle metriche elettriche dall'esterno e originare i

sincrofasori, quindi inviarlo verso un Docker container di controllo che esegue il software PDC. Quest'ultimo, insieme al DB logger e al MySQL container necessario alla memorizzazione persistente dei dati ricevuti, sono stati invece creati ed eseguiti all'interno di una stessa rete virtuale che fosse differente da quelle appositamente creata per l'esecuzione dei PMU-containers. Infine, dato che l'interazione dei Docker container appartenenti a reti virtuali differenti risulta essere impossibile e volendo comunque garantire un corretto grado di isolamento degli stessi, si è fatto in modo da abilitare la comunicazione tra essi attraverso la rete fisica dell'host, sul quale sono stati creati dei port-mapping responsabili di redigere il traffico applicativo verso i componenti dispiegati nelle rispettive reti di appartenenza, ottenendo perciò la comunicazione attesa.

I docker-compose file (si veda 4.2 Docker-compose) usati per la creazione dell'infrastruttura di rete si differenziano per la tipologia di container che creano ed eseguono.

Il primo docker-compose file è usato per la creazione e il dispiegamento dei container chiamati di servizi rappresentati dalla triade grafica e dai MySQLs per i PDC (sia base che di ordine gerarchico superiore); il secondo invece è utilizzato per la creazione dei container applicativi eseguenti i componenti funzionali critici PMU/PDC.

6.2.3 Test-bed

L'infrastruttura di rete a cui si è voluto tendere è rappresentata in Figura 10: Test-bed - dove sono presenti nei nodi terminali tre PMU che acquisiscono i sincrofasori inoltrandoli verso un PDC di primo livello che agisce da aggregatore inoltrando a sua volta i dati compatti verso un PDC di livello superiore avendo cura di memorizzare in maniera persistente i dati ricevuti verso un DMBS locale.

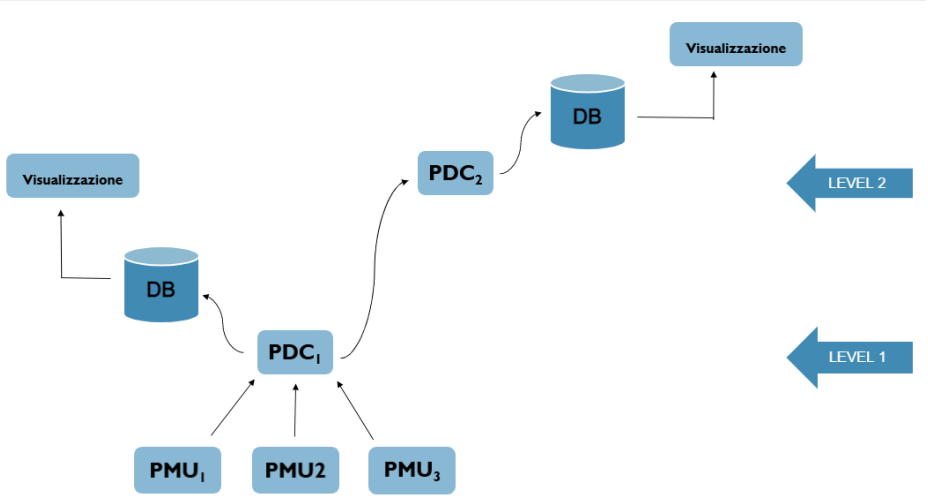


Figura 10: Test-bed

La duale infrastruttura rappresentata da container applicativi e di servizio che è stata invece sviluppata in questa tesi di ricerca è rappresentata in basso:

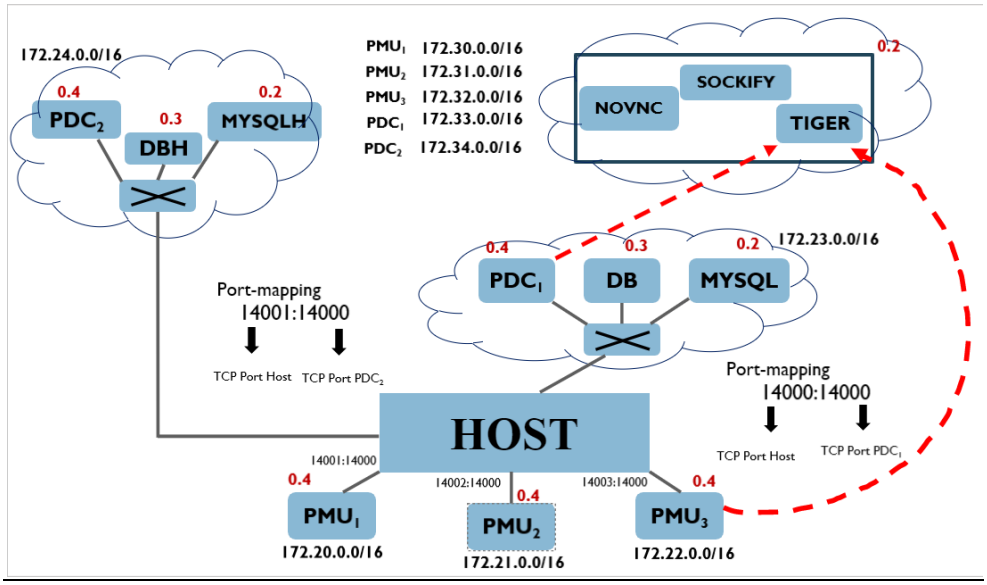


Figura 11: Test-bed rappresentante la Smart-Grid implementata

16

[16] Gli applicativi PMU e PDC usano rispettivamente le porte TCP 15000 e 14000 per essere contattate; la distinzione per indirizzare il traffico ad una PMU rispetto ad un'altra (o verso PDC_1 piuttosto che PDC_2) è fatta sulla base del portadell'host alla quale viene fatta il mapping con la porta interessata.

In basso sono presenti tre differenti reti virtuali nelle quali sono dispiegate le tre PMU containerizzate che avranno il compito di mandare i sincrofasori simulati verso il PDC_1 dispiegato in una rete virtuale differente.

Quest'ultimo ha una duplice funzione: da un lato si interfaccia con il DB logger che stamperà a video il flusso dei dati ricevuti dai nodi terminali e avrà cura di salvare i data-frame verso un DBMS locale MySQL e inoltrerà il flusso dei sincrofasori aggregati verso un PDC_2 di ordine gerarchico superiore dispiegato in una sua rete virtuale.

La comunicazione tra le differenti entità è resa possibile tramite la macchina host sulla quale è stata creata la Smart-Grid.

L'host agisce da ponte tramite una feature di docker quale il port-mapping per rendere possibile la comunicazione che altrimenti non avverrebbe per garantire l'isolamento tra questi.

Per semplicità, sono stati assegnati in maniera statica gli indirizzi di tutti i docker-container presenti nella rete affinché questi non cambino da un'esecuzione all'altra. Sono riportati inoltre i port-mapping creati sulla macchina host per redirigere il traffico verso il container terminale d'interesse e permettere la comunicazione.

In alto sono presenti 5 reti virtuali ulteriori, differenti dalle reti nel quale operano i vari componenti, per permettere di raggiungere il desktop server tramite il noVNC client (nginx), mostrare il display remoto e permettere la configurazione e interazione di ciascuno degli attori presenti nella rete (PMU/PDC) tramite la GUI degli applicativi.

7. Test e validazione

In questo capitolo finale viene validata la correttezza delle interazioni tra le entità presenti successivamente al setup della comunicazione tra essi controllando il contenuto dei messaggi scambiati e le informazione presenti all'interno delle basi dati MySQLs.
Viene infine dedicato un paragrafo ai test di resilienza condotti in questo lavoro di ricerca.

7.1 Setup

Per avviare le comunicazioni tra PMU/PDC e abilitare quindi l'invio dei sincrofasori da parte delle PMU verso i PDC si è utilizzata la funzione "*add source device"* della GUI propria dell'applicativo PDC.
Questa permette, tramite il pop-up che appare a seguito dell'interazione, di inserire gli estremi (id-code, indirizzo IPv4, porta TCP) della PMU o del PDC che si vuole contattare abilitando l'invio del CFG frame che permette di iniziare la comunicazione.

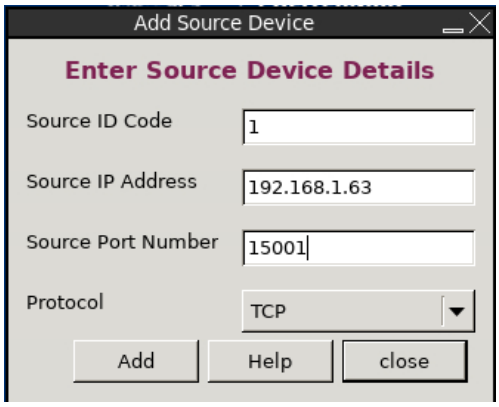


Figura 12: Add Source Device

In figura è rappresentata l'aggiunta della sorgente PMU_1 raggiungibile dal PDC tramite l'indirizzo IPv4 dell'host (sotto dicitura source IP Address) e la porta sul

quale è stato creato il port-mapping come visibile in Figura 11: Test-bed rappresentante la Smart-Grid implementata.

Una volta ultimata l'operazione, il PDC riceverà il CFG-FRAME (si veda 5.1 Standard IEEEc37.118) proveniente dalla sorgente PMU precedentemente aggiunta.

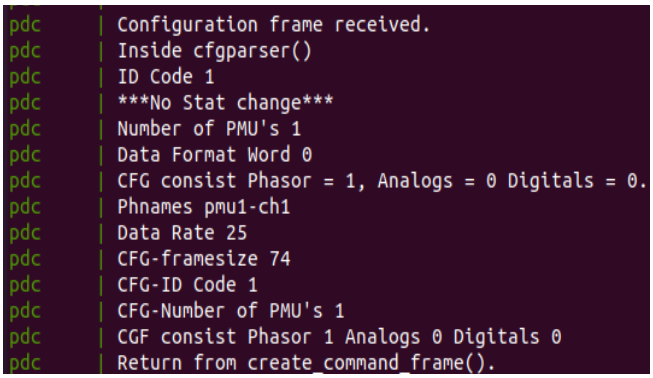


Figura 13: Ricezione e stampa del CFG-frame dal PDC

Come è possibile vedere in figura il frame di configurazione mostra alcune informazione di interesse come l'id-code della PMU_1, il numero dei fasori, dei canali analogici e digitali e lo stato degli input di ogni canale che possiede.

A questo punto il PDC inoltra i dati appena ricevuti sulla porta di default 9000 UDP dove sta in ascolto il DB logger, il quale stampa a video il frame di configurazione appena ricevuto e si incarica di memorizzare i frame dati in un MySQL database dispiegato nella stessa rete.

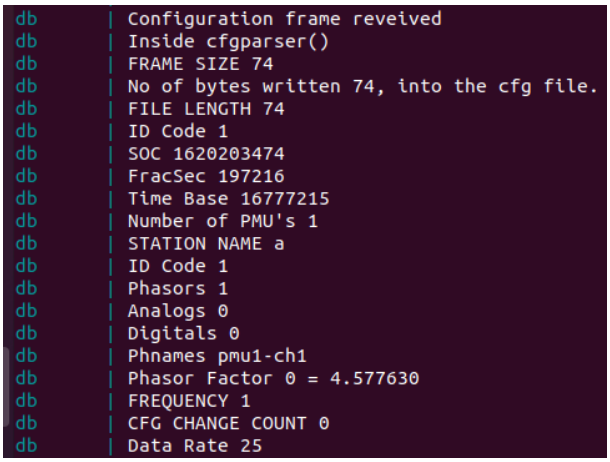


Figura 14: Ricezione e stampa del CFG-frame dal DB logger

La figura mostra quasi le stesse informazioni mostrate in Figura 13: Ricezione e stampa del CFG-frame dal PDC ma in maniera meno compatta stampando altre informazioni d'interesse come:

- SOC timestamp:
- FracSec: frazione di secondi e time-quality;
- Time Base: risoluzione del timestamp FRACSEC;
- Data Rate: rate della trasmissione dati.

Seguono infine le inserzioni fatte dal DB logger verso il DBMS MySQL per la memorizzazione persistente dei DATA-frame ricevuti successivamente all'inoltro del CFG-frame.

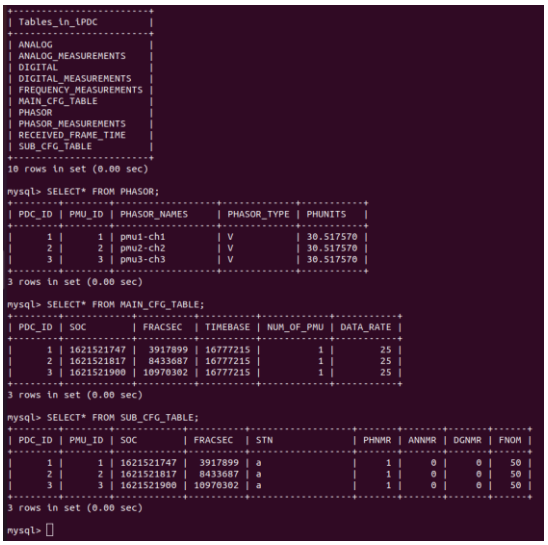


Figura 15: Contenuto del MySQL

In figura è possibile notificare quanto avvenuto andando ad ispezionare il contenuto delle tabelle interessate del DBMS MySQL container e verificando che le inserzioni verso di esso sono andate a buon fine.
Nel dettaglio, le tabelle interessate, dipendenti dalle configurazioni che assumono le PMU, sono le seguenti mostrate in figura.
Infatti, a seguito della non presenza di canali analogici e digitali in ciascuna PMU poichè non inseriti in fase di configurazione, le rispettive tabelle ove vengono memorizzate le informazioni risultano essere vuote a differenza della tabella

PHASOR che invece contiene le entry proveniente dalle tre PMU.

Si può notare infine che in tabella PHASOR, le colonne PDC_ID e PMU_ID riportano lo stesso valore poiché si tratta di una comunicazione PMU-PDC; questo non accade invece nella comunicazione tra PDC-PDC dove i dati aggregati che arrivano a destinazione sono aggregati mostrano un contenuto differente tra le colonne PDC_ID e PMU_ID come è visibile in seguito.

Completate le operazioni descritte sopra, il DB logger stampa in loop i frame dati ricevuti dalla PMU_1 per notificare la sua operatività.

A runtime è possibile interagire con la GUI dell'applicativo PDC per interrompere/riprendere la comunicazione con gli end-point tramite il comando turn off/on data transmission o anche inserire nuove entry PMU/PDC per la comunicazione.

Come visualizzato in Figura 10: Test-bed, in questa smart-grid sono stati aggiunti tre nodi PMU: dunque il PDC riceve i CFG-frame dalle rispettive PMU che poi inoltra al DB logger il quale compie le operazione già descritte in precedenza e quindi stampa a video i frame di configurazione/dati dei 3 nodi e si interfaccia con il MySQL container per memorizzare le informazioni contenute in tali frame.

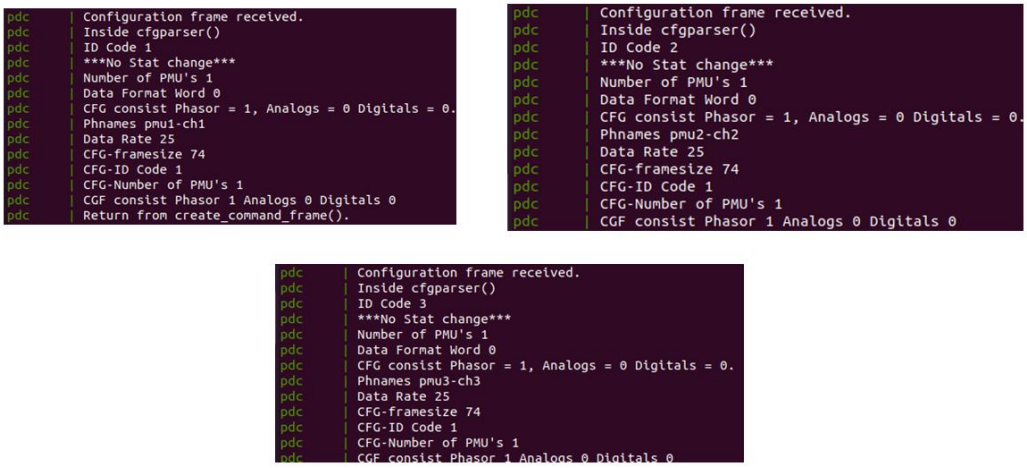


Figura 16: Sommario di tutti i CFG-frames ricevuti dal PDC

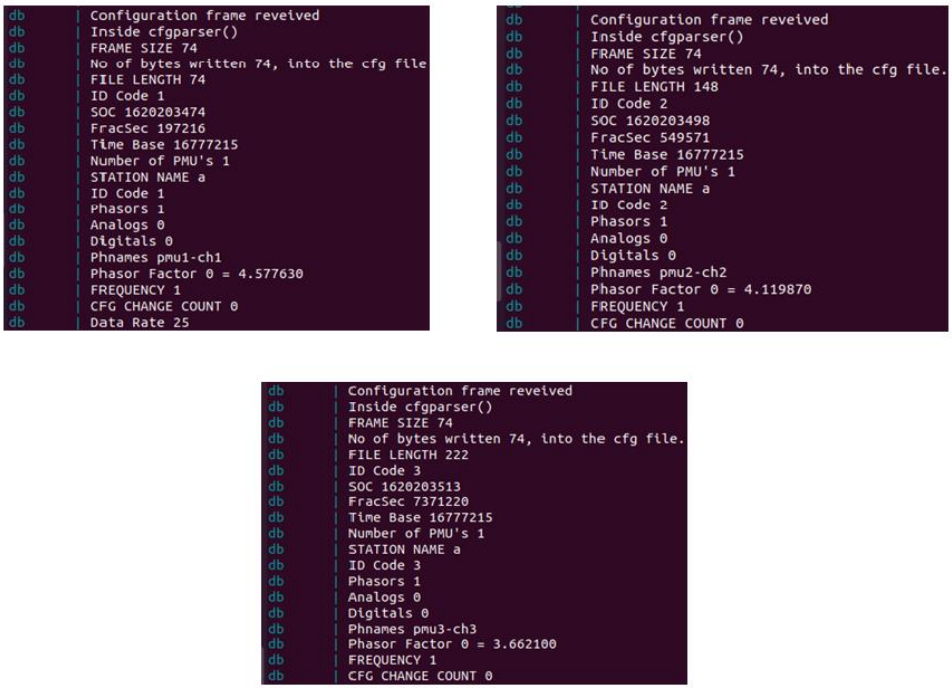


Figura 17: Sommario di tutti i CFG-frames ricevvuti dal DB logger

Quanto illustrato in precedenza riguarda la comunicazione tra le tre PMU dispiegate e il PDC_1. Come risulta visibile in figura 11-12 e come avviene in uno scenario reale, un PDC_1 (e.g. locale) interagisce con un PDC_2 (e.g. corporate) per l'inoltro di dati aggregati sotto un unico data-stream per svolgere funzioni di controllo qualitativo dati o per l'inoltro degli stessi verso applicazioni di monitoraggio e controllo.

Per abilitare la comunicazione tra i due PDC i passi da seguire sono quasi del tutti analoghi: dal PDC_2 si ricorre alla funzione "*add source device"* aggiungendo, in questa sede, non gli estremi per raggiungere le PMU come è stato fatto per PDC_1 ma id-code e porta TCP di quest'ultimo, oltre all'IPv4 dell'host sul quale è creato il port-mapping. L'unica differenza in confronto alla comunicazione PMU-PDC_1 consiste nell'abilitare il PDC_1 a riconoscere la sua controparte come trusted permettendo cosi la ricezione sulla sua porta 14000 TCP di default.

Figura 18: Add Destination Device

Il suo trusted end-point dal quale riceverà la richiesta di CFG-frame è rappresentato dal suo default gateway (sotto dicitura destination IP Address) che impersonifica e inoltra il traffico mandato dal PDC_2 poiché dispiegati in reti differenti.

Ad interazione finita, la connection-table del PDC risulta essere composta da tre sorgenti rappresentati le tre PMU terminali e un device destinazione identificato dal PDC_2 con il quale interagisce.

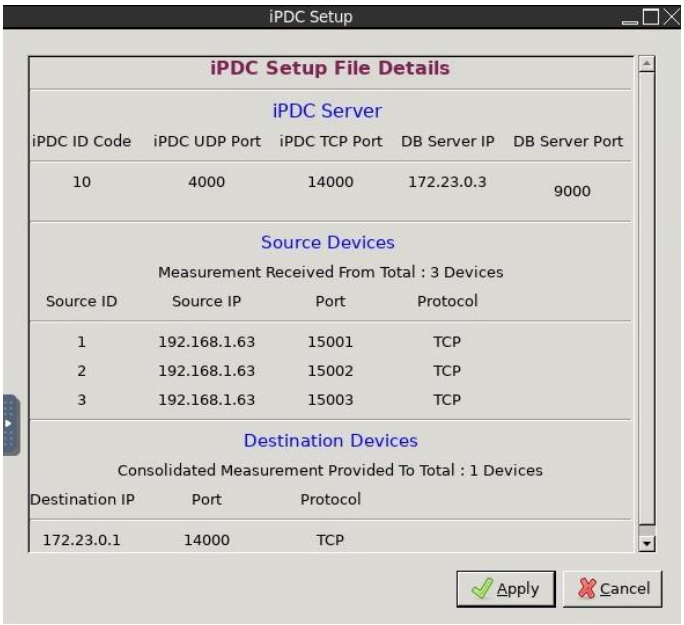


Figura 19: Connection-table del PDC1

Il PDC_1 riceve la richiesta di CFG-frame da parte del PDC di ordine superiore abilitando così l'inizio della trasmissione dei frame dati secondo i passi spiegati in precedenza.

Nelle figure successive sono visibili i frame scambiati tra i due PDC e le stampe condotte dal DB logger (chiamato dbH per differenziarlo dal logger del PDC_1) oltre alle insert fatte verso il DBMS MySQL (con container-name MySQLH per distinguerlo dal MySQL del PDC_1).

```
dbH     | Configuration frame reveived
dbH     | Inside cfgparser()
dbH     | FRAME SIZE 174
dbH     | No of bytes written 174, into the cfg file.
dbH     | FILE LENGTH 174
dbH     | ID Code 10
dbH     | SOC 1620891228
dbH     | FracSec 0
dbH     | Time Base 16777215
dbH     | Number of PMU's 3
dbH     | STATION NAME a
dbH     | ID Code 1
dbH     | Phasors 1
dbH     | Analogs 0
dbH     | Digitals 0
dbH     | Phnames pmu1-ch1
dbH     | Phasor Factor 0 = 4.577630
dbH     | FREQUENCY 1
dbH     | CFG CHANGE COUNT 0
dbH     | STATION NAME a
dbH     | ID Code 2
dbH     | Phasors 1
dbH     | Analogs 0
dbH     | Digitals 0
dbH     | Phnames pmu2-ch2
dbH     | Phasor Factor 0 = 4.119870
dbH     | FREQUENCY 1
dbH     | CFG CHANGE COUNT 0
dbH     | STATION NAME a
dbH     | ID Code 3
dbH     | Phasors 1
dbH     | Analogs 0
dbH     | Digitals 0
dbH     | Phnames pmu3-ch3
dbH     | Phasor Factor 0 = 3.662100
dbH     | FREQUENCY 1
dbH     | CFG CHANGE COUNT 0
dbH     | Data Rate 25
```

Figura 20: Ricezione e stampa del CFG-frame dal dbH logger

Il frame di configurazione mandato dal PDC_1 al PDC_2 e inoltrato al DBH, come visibile in figura, è più grande rispetto ai frame di configurazione visti in precedenza nella comunicazione PMU-PDC.

Infatti, nell'interazione PDC-PDC, il frame di configurazione inoltrato è un sommario composto dall'unione dei frame di configurazione dei nodi terminali PMU poiché i dati tendono ad essere aggregati quando si sale nella comunicazione verso l'alto (ordini gerarchici). E' quindi visibile l'id-code del PDC_1 (in questo è il 10) e oltre ai dati sempre presenti come SOC, FracSec, risulta subito visibile che il numero delle PMU presenti nel CFG-frame sono 3 poiché tale è il numero delle PMU presenti nella Smart-Grid.
Seguono successivamente le configurazioni specifiche di ogni PMU presente come è visibile in Figura 14: Ricezione e stampa del CFG-frame dal DB logger.
Sono compiute infine le insert verso il DBMS del PDC_2 (chiamato MySQLH) dei dati contenuti all'interno di tali frame.

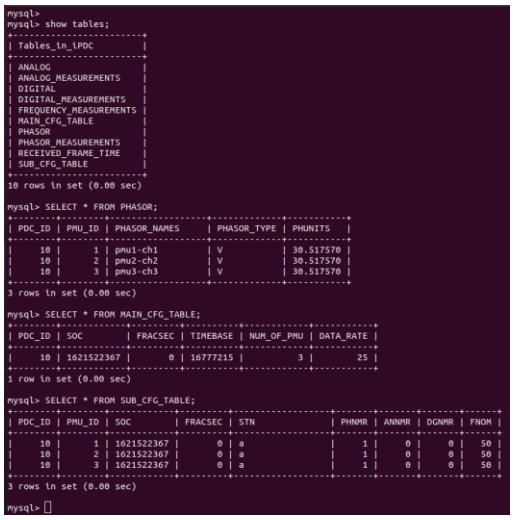


Figura 21: Contenuto del MySQLH

A differenza di quanto visualizzato in figura Figura 15: Contenuto del MySQL, poiché si tratta di una comunicazione PDC-PDC i dati viaggiano aggregati e il contenuto delle colonne PDC_ID e PMU_ID risulta essere differente.

Dunque, a seguito di quanto osservato in precedenza le comunicazioni presenti tra gli applicativi in versione containerizzata risultano funzionare in maniera corretta ed adeguata rispecchiando le interazioni e dando gli stessi risultati che si avrebbero se fossero lanciati come normali processi in ambiente POSIX-compatible.

Nel paragrafo successivo si affronta il tema della resilienza ovvero la capacità di un sistema di adattarsi alle condizioni d'uso e di resistere all'usura-malfunzionamento in modo da garantire la disponibilità dei servizi erogati.
La resilienza rappresenta uno dei maggiori vantaggi acquisiti con l'uso della virtualizzazione, che rispetto al mondo fisico, sembra essere ormai predominante per i vantaggi che essa offre.

7.2 Resilienza

Facendo riferimento al paragrafo 3.5 Analisi di eventi ICT critici, i vantaggi dei servizi derivanti dall'uso della virtualizzazione e dell'orchestrazione di essi sembrano essere acquisiti in maniera quasi gratis; tuttavia l'agevolazione nel rendere smart i servizi dipende fortemente da come questi sono sviluppati e configurati.
La latenza per eseguire determinati task di configurazione o anche semplicemente per rimettere i servizi up a fronte di un guasto sembrano essere trascurabili se paragonati ad uno stesso deployment nel mondo fisico.
Se si considera la sola virtualizzazione dei servizi, come fatto in questo lavoro di ricerca, i vantaggi derivanti sono pochi e non significativi ma il tutto diventa fortemente differente se la virtualizzazione fosse affiancata da un arbitro (chiamato orchestratore) che ne coordina le operazioni in maniera automatizzata e intelligente.
In questo lavoro, a fronte della mancata operatività di un servizio sul quale esegue la PMU o anche il PDC in versione containerizzata, sono stati condotti due test di resilienza.
Seguendo la terminologia usata nel paragrafo sopra indicato, i due resilience-test riguardano la gestione dei seguenti eventi critici:

- Crash CFC (componente funzionale critico) con successivo riavvio in un docker-container con stesso indirizzo IPv4 poiché dispiegato sullo stesso nodo di computing;
- Cambiamento indirizzo IPv4 del CFC per simulare il riavvio su un nodo di computing differente.

Prima di scendere nel dettaglio è opportuno parlare dell'importanza del file di configurazione. Quando una nuova sorgente PMU/PDC eseguita come usuale processo viene creata e configurata, un file di configurazione viene associato riportando le caratteristiche d'interesse. Nel caso di una PMU, il suo rispettivo file contiene parametri di configurazione come id-code, numero di fasori, canali digitali e analogici e la porta TCP usata per comunicare con essa; nel caso di un PDC vengono invece salvati l'id-code, la porta TCP impiegata nel dialogo verso gli upper PDC, l'indirizzo del DB logger e la connection-table che possiede riportando i nodi (PMU terminali e PDC di più alto livello) con il quale esso comunica.

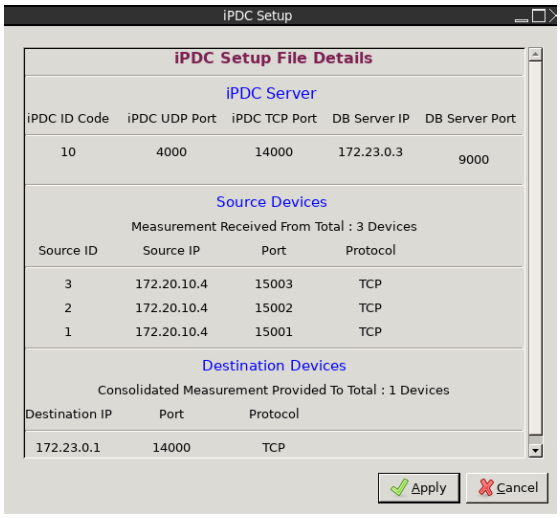


Figura 22: File di configurazione di un PDC

Tale file di configurazione contiene quindi tutte le informazioni necessarie che servono per ripristinare il servizio di una PMU/PDC a seguito di un periodo di inattività quale, per esempio, un guasto. Esso è uguale al contenuto delle informazioni della connection-table (si veda Figura 19: Connection-table del PDC1) perché essenzialmente deriva da essa.

Cliccando sul tasto apply, il nuovo servizio carica quindi la configurazione mostrata in figura e ritorna ad interagire con i vari end-point come in condizione normale.

Tornando all'esecuzione dei servizi in ambiente virtualizzato, data l'importanza

del file di configurazione, è stato specificato un volume alla creazione del docker-container sul quale esegue il CFC per rendere persistenti i dati di configurazione generati anche a seguito di un crash improvviso dello stesso docker-container.
Infatti, a seguito di un crash del CFC PDC, il nuovo servizio viene ripristinato caricando su questo il file del 'vecchio' CFC PDC soggetto a crash.
Si può notare che il nuovo PDC inizia la sua comunicazione con i vari end-point presenti nel file (appena caricato) in maniera trasparente e del tutto corretta.
È stato riportato, nelle prime righe di questo paragrafo, che i vantaggi derivanti dall'uso della sola virtualizzazione non rappresentano un valore aggiunto per abilitare, in maniera intensiva, il passaggio dal mondo fisico al mondo virtuale.
La presenza di un orchestratore renderebbe invece lo scenario differente introducendo una serie di vantaggi importanti poiché permetterebbe di automatizzare i task a seguito di un evento improvviso/schedulato nel tempo; la presenza di un orchestratore potrebbe però introdurre innumerevoli svantaggi se non eseguito e configurato in maniera adeguata.

8. Conclusioni e lavori futuri

In questo lavoro di ricerca è stato sviluppato un prototipo di Smart-Grid composto dai principali dispositivi coinvolti in ambito elettro-energetico operanti come funzione software virtualizzate in docker-container ed eseguite su hardware general-purpose.
In questo capitolo conclusivo sono stati riassunti i punti critici di questo sistema e quali miglioramenti potranno essere fatti in futuro.

8.1 Configurazione

Uno dei fattori che ha impedito di rendere automatizzato il sistema è stata la necessità di configurazione di ogni singola entità presente e il relativo setup della comunicazione.
È evidente che una soluzione di questo tipo, in ambiente wide, non risulta essere scalabile e la sua complessità e necessità di tempo di configurazione/setup cresce all'aumentare del numero dei dispositivi coinvolti.
In uno scenario altamente variabile come quello elettro-energetico dove il tempo (dell'ordine dei ms) risulta essere il fattore primario, l'obbligo di configurazione ne rappresenta un limite forte.
Tramite la creazione a priori dei file di configurazione da caricare su ogni singolo dispositivo è possibile ridurre notevolmente tale tempo ma comunque il sistema creato manca di automazione nel caricamento di essi.
In questa sede le comunicazioni sono quindi etichettate human-driven e questo rappresenta il problema principale derivante dalla realizzazione di una Smart-Grid tramite gli applicativi messi a disposizione dal pacchetto iPDC software.

8.2 Possibili miglioramenti

Nel lavoro di tesi condotto, a seguito dell'utilizzo della sola virtualizzazione nell'abilitare il passaggio dei servizi operanti su hardware dedicato special-purpose verso nodi di computing general-purpose, numerose ottimizzazioni possono essere ancora attuate per rendere il sistema maggiormente resiliente e automatizzato.

Una Smart-Grid può essere resa ancora più smart tramite la presenza di intelligenza aggiuntiva quale un orchestratore kubernetes per rendere il sistema reattivo a fronte di eventi che determinano un cambiamento di stato (e.g. guasto).

Il Cloud Computing o anche il Fog/Edgecomputing (si veda il capitolo 1. Cloud e Fog/Edge Computing) non possono essere considerate soluzioni efficaci e scalabili se non è presente un'entità superiore che ne dirige le operazioni in maniera smart.

A maggior ragione, in una Smart-Grid dove lo stato della rete elettrica e energetica cambia costantemente nel tempo, pensare di coordinare le operazioni a mano rende lo scenario non scalabile e pronto ad errori.

Seguendo le tabelle elencate in "3.5 Analisi di eventi ICT critici"si può notare che la presenza di un orchestratore sembra essere la soluzione più efficace nel rendere il sistema automatizzato, resiliente, veloce e adattabile ai cambiamenti di stato.

Nei test di resilienza condotti e spiegati nel paragrafo precedente, se il servizio di un PDC inteso come funzione software virtualizzata operante in un docker-container si interrompesse per un motivo qualsiasi, l'operatore dovrebbe riconoscere l'inattività di tale servizio ed eseguire un docker-container duale sul quale far ripartire il servizio.

Risulta subito evidente, dunque, che il down-time del servizio dipende fortemente dalla latenza che ha un operatore umano nel riconoscere il problema e rieseguirlo e, normalmente, questo periodo di tempo può essere anche abbastanza lungo.

Con un orchestratore invece, la latenza dipende solamente dal tempo che impiega il servizio nel tornare up e running quando dispiegato su un nodo piuttosto che un altro evitando quindi il tempo di risoluzione dell'operatore poiché riconosce, idealmente a tempo nullo, che il servizio ha bisogno di essere rieseguito poiché andato down.

Con la presenza di un orchestratore kubernetes, il sistema diventa più complesso acquisendo una maggiora complessità che, se configurato in maniera non adeguata, introdurrebbe più svantaggi che vantaggi.

Per finire, le due strade verso cui tendere per migliorare le perfomance del sistema sono:

- Introduzione di un'orchestratore kubernetes per introdurre i vantaggi elencati e anche per rendere attuabile una soluzione di Fog/Edgecomputing;
- Introduzione delle Software Defined Network (SDN) per garantire una rete infrastrutturale che si adatta maggiormente allo stato della rete elettrica/energetica soggetta a numerosi cambiamenti di stato oltre che per mitigare problemi di sicurezza informatica.

Bibliografia

[1] NIST, «The NIST Definition of Cloud Computing,» [Online]. Available: https://www.nist.gov/publications/nist-definition-cloud-computing.

[2] OpenFog, «OpenFog Reference Architecture,» [Online]. Available: https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf .

[3] «FreeBSD,» [Online]. Available: https://www.freebsd.org/doc/it_IT.ISO8859-15/books/handbook/jails-intro.html.

[4] «LXC - Linux Containers,» [Online]. Available: https://linuxcontainers.org/downloads/.

[5] Thurrott, Paul, «Windows Server Virtualization Preview,» 2007. [Online]. Available: https://web.archive.org/web/20071011034732/http://www.winsupersite.com/showcase/viridian_preview.asp .

[6] Barbier, Julien, «It's Here: Docker 1.0,» 2014. [Online]. Available: https://blog.docker.com/2014/06/its-here-docker-1-0/ .

[7] DockerDocs2017a, «Overview of Docker Compose,» [Online]. Available: https://docs.docker.com/compose/.

[8] «Docker Swarm,» 2017.

[9] «IEEE Power and Energy Society, C37.118.1-2011 - IEEE Standard for Synchrophasor Measurements for Power Systems, IEEE Power and Energy Society.».

[10] «[PES2018] IEEE Power and Energy Society, 60255-118-1-2018 - IEEE/IEC International Standard - Measuring relays and protection equipment - Part 118-1: Synchrophasor for power systems - Measurements, IEEE Power and Energy Society.».

[11] «[PES2013] IEEE Power and Energy Society, IEEE C37.244-2013 - IEEE Guide for Phasor Data Concentrator Requirements for Power System Protection, Control, and Monitoring, IEEE Power and Energy Society,

2013.».

[12] Docker Inc, «Docker container networking,» [Online]. Available: https://docs.docker.com/ engine/userguide/networking/.

[13] Docker Inc, «Docker container volume,» [Online]. Available: https://docs.docker.com/storage/volumes/.

[14] Docker Inc, «Docker container volume,» https://docs.docker.com/compose/.

[15] «iPDC software,»https://sourceforge.net/projects/iitbpdc/files/) .