Sistema Multi-Agente per il supporto agli MSP

Lorenzo Valgimigli, Riccardo Soro

February 2, 2020

CONTENTS

1 INTRODUZIONE

1.1 IDEA DI BASE

Oggi giorno sempre più aziende medio piccole si affidano a terzi per la gestione del IT attraverso un meccanismo ben conosciuto chiamato **Outsourcing**. Tra i vari servizi di questo tipo troviamo il **Managed Service Provider** (o MSP) che è una tipologia di outsourcing di tutto l'apparato informatico in base alle esigenze e alle possibilità del cliente. Le aziende che forniscono il servizio spesso si trovano a lavorare con moltissime aziende e, di conseguenza, con reti formate da molti dispositivi, con vari compiti e controlli da svolgere. Per queste aziende è necessario avere del personale competente in grado di gestire e risolvere eventuali problemi da remoto qualora si verificassero dei malfunzionamenti o in locale in caso la rete dovesse cadere. Spesso per queste aziende avere un sistema di controllo autonomo che anche in caso di isolamento continui a svolgere il suo lavoro e provi a ristabilire in autonomia una connessione può rappresentare un guadagno enorme in termini di tempo ma anche economici. In più queste tecnologie spesso sono scritte attraverso l'uso di linguaggi e framework complicati e potenti che, da un lato garantiscono affidabilità e potenza computazionale, dall'altro fanno si che il tecnico informatico medio fatichi a metterci mano per risolvere anche i problemi più semplici. Con questo progetto vogliamo inserirci in questo spazio e creare quindi un sistema che provi a gestire queste problematiche. Questo progetto sarà per aziende medio piccole che necessitano di un sistema di **Remote Management**. Quindi per avere un sistema di controllo dei vari servizi offerti facile e veloce. Il fornitore quindi potrà monitorare i servizi IT forniti utilizzando un sistema distribuito che garantirà una certa autonomia dei singoli servizi, il controllo su di essi per garantire un corretto funzionamento e una serie di funzionalità per gestire runtime problematiche comuni senza l'intervento diretto di un operatore.

1.2 SCENARI

In questo paragrafo, per spiegare con maggiore precisione quello che vorremmo realizzare, andremo a mostrare e discutere alcuni casi applicativi del sistema.

1.2.1 IL PROGETTO

Ora proveremo a descrivere l'idea del progetto con maggior dettaglio. Si immagini ci sia un azienda medio piccola che magari ha più sedi e per ogni sede ha un dominio autonomo. Tale azienda necessita di svolgere in maniera autonoma determinate funzioni su alcuni o tutti i domini e raccogliere delle informazioni. Per non appesantire il carico di lavoro dei tecnici informatici ha l'esigenza che tali funzionalità siano completamente autonome e che possano reagire a possibili problematiche o cambiamenti dell'environment. Il nostro progetto nasce per rispondere a questo tipo di esigenza che non è stata immaginata da noi ma ci è stata sottolineata dall'azienda in cui Riccardo Soro lavora. Quindi questa problematica è reale e questo progetto potrebbe portare a risvolti concreti. Un altra prerogativa

del sistema che l'azienda ci ha fatto presente è quella di non essere scritta con linguaggi di programmazione complessi poichè spesso aziende medio piccole non hanno operatori informatici formati ma autodidatti che faticherebbero a mettere mano al codice. Per far fronte a questo problema l'azienda ha sottolineato positivamente l'utilizzo di un linguaggio di scripting come Python o come Power Shell molto più semplici e facili da capire.

1.2.2 Registrazione di un dominio.

Si immagini che un'azienda necessiti di servizi informatici e scelga come soluzione di affidarsi a terzi che le forniscano moduli software per svolgere tali funzionalità. Tale azienda potrà quindi sottoscrive un contratto con gli amministratori di questo sistema. In questo caso l'azienda deve avere o creare un Active Domain sotto Windows e registrarlo nel sistema. Dopo di che sul Domain Controller verrà lanciato un elemento di Coordinamento che si occuperà di mappare la topologia della rete e, in base al servizio richiesto, installare sui vari host determinati elementi che svolgano tale servizio. Il fornitore sarà in grado di monitorare ed eventualmente correggere il funzionamento del servizio fornito da remoto.

1.2.3 Monitoring di più domini.

Nel caso ci siano più domini registrati alla rete allora l'elemento di coordinamento dei vari domini si occuperà di collezionare i messaggi dei vari moduli installati nel dominio di competenza e inviarli a un Database cloud distribuito al quale gli amministratori avranno accesso. Dopo di che il sistema offre agli amministratori la possibilità di visualizzare tali informazioni attraverso una interfaccia web.

1.3 Politiche di autovalutazione

Il progetto mira alla realizzazione di un prototipo che svolga le varie funzionalità base. Essendo questo progetto vicino più ad uno studio di fattibilità con tanto di prototipo più che ad una vera realizzazione di un sistema è molto complesso formalizzare degli obbiettivi a priori in modo che essi siano verificabili una volta concluso il progetto. Vogliamo però darci delle metriche di valutazione che siano ragionevoli per quello che andremo a sviluppare:

- **Scalabile**. Essendo questo un progetto di un sistema distribuito è ovvio che tra le caratteristiche ci deve essere la scalabilità. Proveremo a fare particolare attenzione a questo aspetto perché vogliamo che il nostro sistema lavori bene sia con pochi domini sia con molti. Ovviamente ci saranno delle limitazioni per quanto riguarda le risorse disponibili ma cercheremo di sfruttare al massimo ciò che ci viene offerto gratuitamente.
- **Autonomo**. E anche qui ricadiamo all'interno di una caratteristica fondamentale dei sistemi distribuiti ad agenti. Per questa condizione immagineremo alcuni casi in cui la rete si isola e adotteremo tutte le misure perché gli elementi nella rete possano continuare a lavorare anche in condizioni non favorevoli. In più vorremmo che il sistema

fosse autonomo anche nella distribuzione delle funzionalità, nelle scelte, ... magari addottando meccanismi di **Code Mobility**.

- **Funzionante**. Può sembrare una metrica molto astratta ma in realtà è forse la più concreta. Ci preme che il prototipo del sistema svolga correttamente tutte le funzionalità base. Per funzionalità base si intendono tutti quei task che sono fondamentali per il corretto funzionamento del sistema.

2 REQUISITI DEL SISTEMA

Dopo aver spiegato velocemente il dominio applicativo bisogna capire bene quelle che dovranno essere le funzionalità del sistema. Avendo come obbiettivo quello di mitigare alcuni problemi sopracitati il sistema dovrà quindi avere le seguenti caratteristiche:

- **Autonomia**: Il sistema si baserà su elementi che dovranno essere autonomi. Potranno interagire tra loro ma questa interazione non è condizione necessaria per il loro funzionamento così che se dovessero isolarsi potranno comunque continuare a svolgere il loro lavoro nei limiti del possibile. Per garantire questa caratteristica ogni elemento avrà i propri obbiettivi e in più dovrà gestire il **Versioning** e le **dipendenze**
- **Context-Awerness**: Gli elementi del sistema e soprattutto quelli con ruoli di coordinamento dovranno studiare il contesto in cui sono inseriti per trovare in autonomia risorse, host, Server, ...
- **Fail-Reactness**: Gli elementi del sistema dovranno essere pronti a reagire ad eventuali problemi che si presentano e flessibili nel modificare i propri comportamenti per poter aggirare tali errori. Primo di tutti il sistema avrà un meccanismo che, in caso di errore e caduta della connessione, si preoccuperà di ripristinare quest'ultima.

2.1 REQUISITI NON FUNZIONALI

Il sistema in oggetto dovrà in oltre avere le seguenti caratteristiche:

- Scritto in un linguaggio di programmazione accessibile. Vogliamo che il tecnico informatico di una piccola azienda, che spesso non ha nemmeno una laurea in questo campo, possa comunque provare a mettere mano al codice o al sistema in caso ci siano problemi. Per fare questo serve un linguaggio che sia semplice e conosciuto, ma allo stesso tempo potente e ben supportato. Si è scelto quindi di realizzare l'intera infrastruttura utilizzando **Power Shell** che è nativamente supportato da **Windows 10**, essendo un linguaggio di scripting, quindi più semplice da utilizzare o da imparare rispetto a linguaggi di programmazione da compilare, ma comunque solido e performante.
- Interfacciamento web al sistema. Ovvero gli amministratori potranno visualizzare il sistema, gli output dei vari elementi, i possibili errori, ... dal portale web che deve essere semplice e intuitivo.

2.2 TECNOLOGIE

Per la realizzazione di questo progetto sono state selezionate varie tecnologie da utilizzare mentre altre sono state scartate. Analizziamo velocemente quali scelte sono state fatte e perché:

- **Linguaggio per la scrittura dei moduli**. Come abbiamo già ampiamente dette e motivato nei primi paragrafi abbiamo scelto di utilizzare un linguaggio di scripting per la

realizzazione del sistema. Tra i possibili linguaggi ne abbiamo identificati 3 eleggibili: Python, PowerShell e VBS perché tutti funzionano su Windows.

  - `Python`: sicuramente uno dei linguaggi di scripting più utilizzato a livello professionale e nel campo della ricerca. E' un linguaggio completo e ben supportato dalla comunity. Note negative sono che il tecnico informatiche che abbiamo identificato come target probabilmente non ha mai utilizzato questo linguaggio. In più python non è nativamente installato sui sistemi operativi Windows.
  - `VBS`: Linguaggio di programmazione largamente utilizzato da chi lavora con Windows, completo e rodato è sicuramente un candidato migliore di python.
  - `Power Shell`: Linguaggio recente che sta diventando fondamentale per chi lavora su Windows. E' supportato nativamente da Windows (che arriva con la versione 4 preinstallata) ed è più potente di VBS.

  La nostra scelta è quindi ricaduta su quest'ultimo.

- **Database**. Ci serviva un database gratuito che però offrisse Resource on Demand in maniera dinamica. Caratteristica che noi non avremmo sfruttato poiché a pagamento ma che, ai fini della scalabilità del sistema, sarebbe stato molto utile avere. Abbiamo quindi scelto di utilizzare **MondoDB Cloud** poiché si può utilizzare la versione **Shard** quindi distribuita e ridondante. Caratteristica molto interessante che calza a pennello con quelli che sono i requisiti del notro progetto. Altro fattore che ci ha spinto in questa direzione è l'averlo già utilizzato in altri progetti e quindi avere una certa dimestichezza.

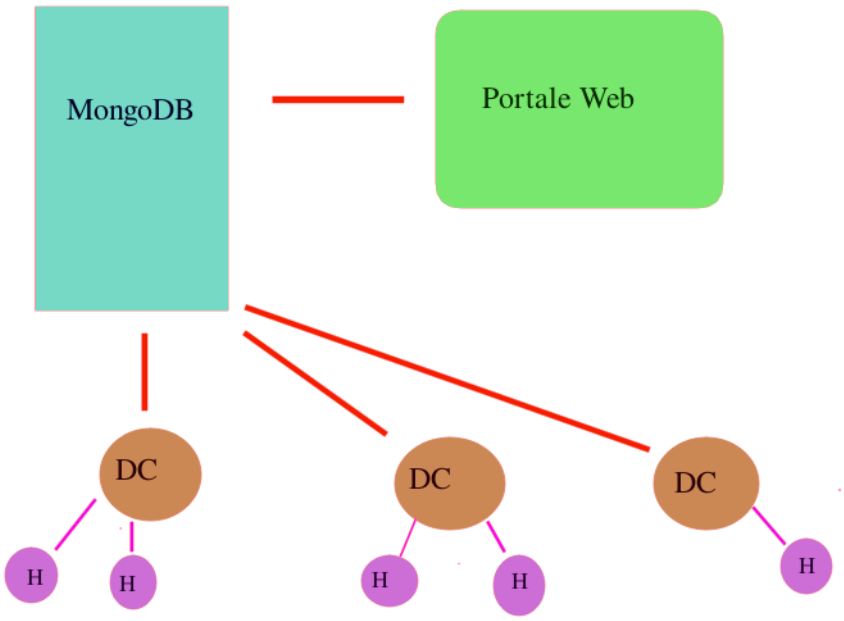

Figure 3.1: Schematizzazione del dominio e dei vari elementi del sistema

3 Design

Nel seguente capitolo andremo a parlare e spiegare tutte le scelte progettuali fatte per realizzare i sistema, le modalità di lavoro e le problematiche riscontrate.

3.1 Modellazione del Dominio

Il sistema deve modellare le entità presenti in un Dominio. I principali elementi sono quindi:

- Domain Controller
- Hosts

In questo sistema il cliente è rappresentato da un dominio, quindi da una serie di computer e/o dispositivi connessi a una rete e gestiti in quelche modo da un Domain Controller. Su ogni dominio si vogliono utilizzare programmi che aggiungano certe funzionalità, certe caratteristiche e che siano monitorabili dagli amministratori. E' stato scelto di utilizzare un modello con **Coordinazione** per gestire i singoli domini. Tale modello prevede l'uso di un elemento che si occupi di coordinare gli altri elementi presenti negli host del dominio. In oltre bisogna considerare che le dimensioni dei vari Domini può variare anche nel tempo. Per gestire questa caratteristica è necessario sviluppare un sistema **scalabile** e soprattutto **decentralizzato** poiché è prerogativa del sistema avere le varie funzionalità distribuite sia nello spazio sia nelle unità logiche presenti. Queste caratteristiche spingono verso un sis-

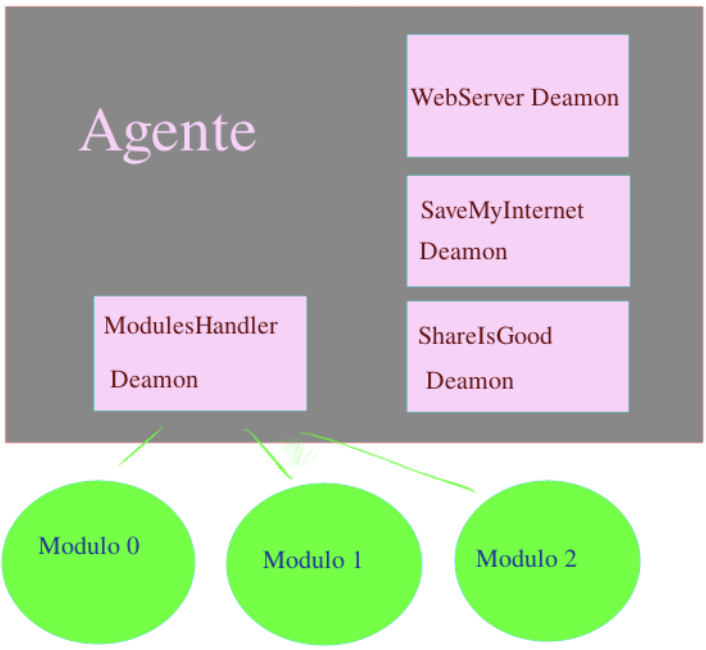

Figure 3.2: *Nell'immagine si vedono i vari elementi che compongono l'agente quali: i demoni e i moduli*

tema distribuito ma la forte autonomia che i vari elementi devono avere ci fa optare per una specifica tipologia: **sistema distribuito ad agenti**. E' stato scelto di implementare un modello **multi-agente** dove sul Domain Controller viene installato un agente di Coordinazione detto **Coordinator Agent** che si occuperà di studiare la rete e di installare e lanciare vari agenti sui vari hosts. Sarà quindi l'agente sul Domain Controller per primo a interagire con il Database leggendo quali agenti e quali funzionalità eseguire sui vari elementi della rete. Sugli host, invece, vengono eseguiti degli agenti che avranno vari compiti. Di seguito andremo ad analizzare le varie funzionalità degli agenti presenti nel **Multi-Agent System**

3.1.1 AGENTI

Il sistema impiega vari agenti per eseguire varie funzionalità. Essendo degli agenti hanno delle caratteristiche comuni che li caratterizzano ovvero un agente è una entità **autonoma** che incapsula il proprio controllo. Essi si realzionano tra di loro e, se necessario, studiano il contesto in cui si trovano per poterci interagire. Gli agenti nel sistema svolgono varie funzionalità e operazioni ognuna delle quali è affidata a un componente software chiamato demone. I demoni impiegati in questa prima versione del sistema sono:

- **SaveMyInternet Demon**. Questo demone serve a garantire all'agente autonomia e reattività infatti, questo componente software, si occupa di controllare lo stato della rete e reagire nel caso il collegamento tra l'agente e il coordinator dovesse cadere per qualche motivo. In caso si verifichi tale evento questo componente si occuperà di

tentare di ripristinare la connessione con Internet utilizzando delle impostazioni di rete salvate in precedenza o cercando un nuovo DHCP server nella rete. Inoltre, in caso non si riesca a contattare il Coordinator Agent questo demone instaurerà delle connessioni **Peer2Peer** con gli altri agenti in modo da poter continuare a svolgere le varie funzionalità anche senza una centralizzazione.

- **WebServer Deamon**. Come abbiamo già detto ogni agente è un entità autonoma in grado di operare senza l'intervento di fattori esterni ben che meno di un operatore umano. Comunque casi in cui ad esempio un tecnico debba metterci mano sono ancora possibili. Si pensi ad esempio ad un guasto o un errore del software che necessiti di un intervento esterno. Per questi casì ogni agente è dotato di una funzionalità **WebServer** che propone un interfaccia human-friendly con le funzionalità e le impostazioni dell'agente.
- **ShareIsGood Deamon**. Questo demone rappresenta un aggiunta di funzionalità al dominio. Come già detto, un obbiettivo di questo sistema, è quello di supportare aziende che fanno outsourcing di soluzioni informatiche quindi è importante avere degli strumenti per aggiungere funzionalità ai vari host qualora il cliente lo richieda. Per fare questo chiunque può sviluppare un demone o un modulo (Vedi prossimo paragrafo) che svolga determinate operazioni. Abbiamo quindi sviluppato un demone che permette agli agenti che lo utilizzano di fornire un interfacciamento tramite **API Call** alle stampanti. Esso espone le proprie stampanti e le condivide a richiesta. Gestendo quindi le code per le richieste e le varie funzioni delle stampanti.
- **ModulesHandler Deamon**. Quest'ultimo demone si occupa di controllare quali moduli l'agente deve eseguire. I moduli come verranno spiegati ampiamente nel paragrafo seguente sono delle funzionalità aggiuntive che determinano il **Carattere** dell'agente. Ad ogni agente vengono assegnati dei moduli da eseguire e da gestire. Ovvero ogni modulo potrebbe necessitare di varie dipendenze da scaricare, in più essi restituiscono dei valori e dei risultati che devono essere raccolti e salvati sul Database.

Bisogna inoltre specificare che il Controller Agent allo stato attuale del sistema è composto solo dal **ModulesHandler Deamon** che si occupa di contattare il database dove è salvata una mappa degli host e delle varie funzionalità del dominio, scarica da qui i vari agenti e le informazioni ad essi relative e si occupa di lanciare sui vari host gli agenti assegnandoli un deteminato compito.

3.1.2 MODULI

Come detto sopra ad ogni agente vengono specificati dei moduli che sono dei task aggiuntivi che l'agente deve fare. Ad esempio possono essere dei task di controllo a livello dell'host o a livello della rete, possono essere delle funzionalità aggiuntive come backup, controllo file di log .... Tutte queste funzioni vengono modellate all'interno dell'oggetto Modulo e gestite dei vari agenti tramite il demone **ModulesHandler Deamon** (Vedi paragrafo superiore). I moduli possono essere di varie tipologie a seconda di quale è il loro target. Per target si intende quale elemento o gruppi di elementi nella rete lo devono eseguire.

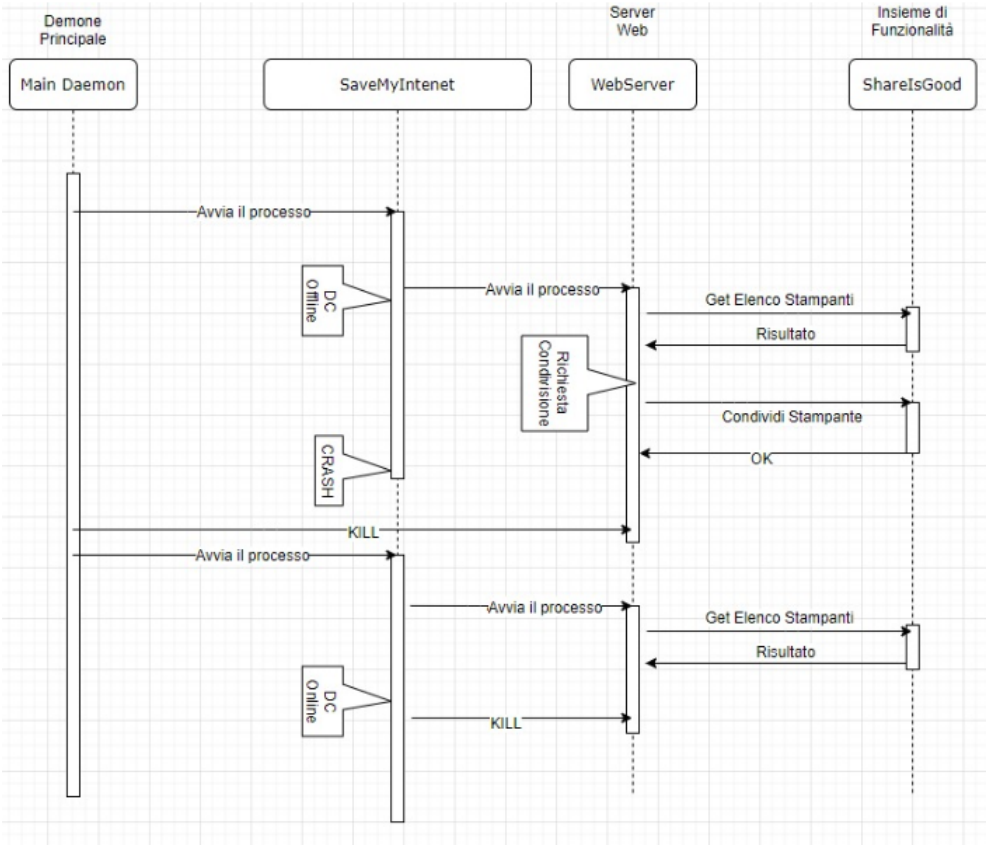

Figure 3.3: *Nell'immagine vengono mostrati i vari demoni che compongono l'agente e come interagiscono tra loro*

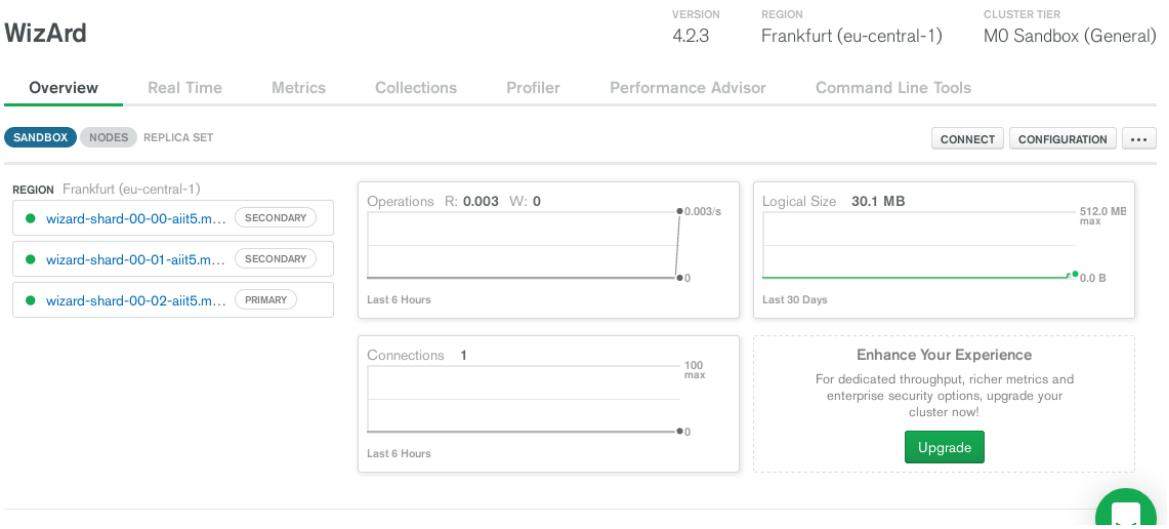

Figure 3.4: *Overview del database. Dalla immagine si vedono i 3 nodi su cui il Database è distribuito e varie informazioni come la memoria disponibile e il numero di connessioni per ora*

- * o **All** che indica che tutti gli agenti all'interno di un dominio devono eseguire quel task.
- **%** o **One** che indica che almeno un agente nella rete deve eseguire quel task
- **Nome Host** indica che l'agente su quello specifico host deve eseguire quel task

Ogni modulo ha inoltre delle caratteristiche importanti:

- **Versioning**. Ogni modulo ha una versione di rilascio che nel tempo può essere modificata. E' compito del gestore moduli occuparsi di scaricare la versione corretta del modulo.
- **Dipendenze**. Ogni modulo per funzionare deve specificare tutto quello di cui ha bisogno per funzionare come ad esempio: data, DLL, altri moduli .... E' compito del demone di gestione moduli controllare le dipendenze e, nel caso sull'host mancassero uno o più di questi elementi, scaricarli dal database.
- **Output**. Ogni modulo restituisce degli output con un formato comune a tutti i moduli in modo che il gestore moduli possa riconoscere le informazioni necessarie e collezionarle. Gli output possono essere di tre tipologie:
  - `INFO`: valori utili da leggere in fase di debug o per monitorare il funzionamento del modulo. Anche i risultati del modulo sono mappati in questa tipologia.
  - `WARINING`: messaggi che identificano un problema esterno che impossibilita il modulo a funzionare.
  - `ERROR`: messaggi che vengono prodotti quando si verifica un errore interno al modulo e contengono delle informazioni utili al debug.

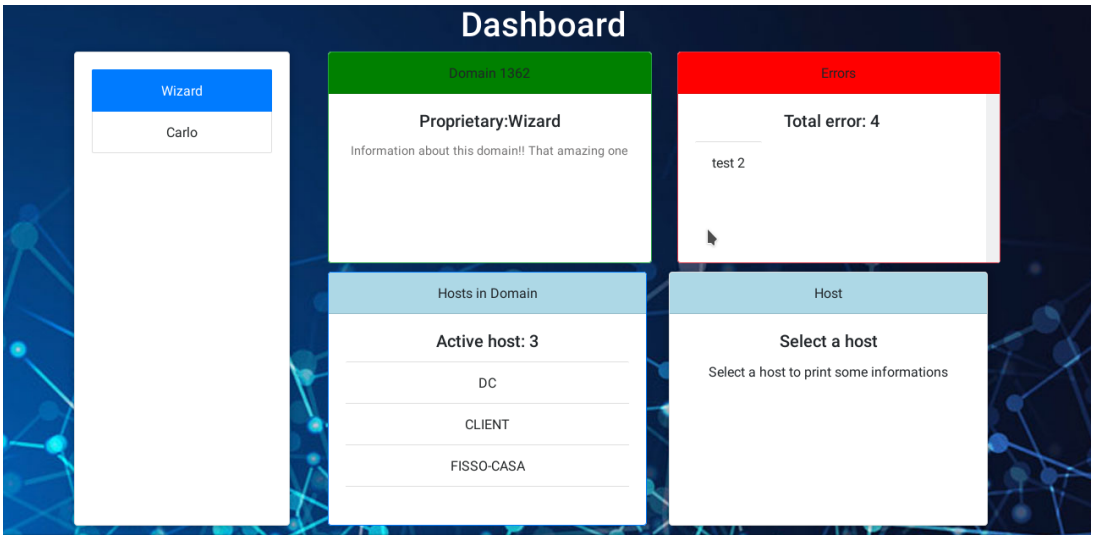

Figure 3.5: Dash Board presente sul portale web

3.2 Database

Questo sistema necessita di un punto di centralizzazione che permetta di mantenere le informazioni e i dati ed avere un interfaccia di accesso uniforme. Abbiamo quindi scelto di utilizzare un Database unico per il sistema. La scelta è ricaduta su **MongoDB Cloud** per vari motivi elencati nel paragrafo `'Tecnologie'`, in modo da poter utilizzare un database documentale, distribuito e gratuito. La versione gratuita mette a disposizione uno spazio di archiviazione limitato: 500Mbyte, un numero di accesso per ora massimo limitato a 100 ma comunque si appoggia su un database distribuito e ridondante su 3 server garantendo quindi una serie di vantaggi tra cui scalabilità e sicurezza. Il database è formato da più database interni, uno per ogni modulo in modo che questi ultimi possano sfruttare dello spazio di archiviazione per salvare quello di cui hanno bisogno come file di configurazione o altro. Così facendo si mantiene un buon grado di separazione di concetti e indipendenza tra i vari moduli e quindi tra gli agenti che li utilizzano.

3.3 Lato Web

Come si legge da i requisiti non funzionali il sistema necessita di un interfacciamento web per permettere agli amministratori del sistema di poter interagire e controllare le varie entità in gioco attraverso uno strumento semplice e intuitivo. Essendo l'aspetto legato al web non oggetto di valutazione di questo esame e non strettamente pertinente ai fini del corso si è optato per una scelta semplice, senza troppe finezze o frivolezze estetiche che non richiedesse troppo tempo ma che fosse estremamente funzionale. Il sito web ha quindi una serie di funzionalità importanti:

- Accesso a tutti i domini registrati dei clienti
- Visualizzazione dei vari host nei vari domini

- Possibilità di visualizzare informazioni sugli host e sui domini
- Visualizzazione immediata di errori o problemi legati al dominio selezionato
- Possibilità di visualizzare i messaggi dei vari agenti sui vari host ed, eventualmente, eliminarli.

3.4 Interazione tra i vari elementi

In questo paragrafo vediamo le interazioni tra i vari elementi del sistema nei vari momenti del suo **Life-Cycle**.

3.4.1 Installazione sul Domain Controller

Come prima cosa, dopo la registrazione di un Dominio nel sistema bisogna installare l'agente di Coordinazione sul Domain Controller che provvederà poi a creare le condizioni favorevoli alla creazione di tutti gli altri agenti. Ecco quali sono i passaggi che vengono eseguiti:

- **Target**: Domain Controller di un Dominio Windows, si presuppone si abbia accesso come amministratore di dominio e come amministratore locale
- **GPO**: L'agente, una volta lanciato sul DC crea una GPO (Policy di gruppo) e seleziona i vari client nel dominio che sono i vari host su cui gireranno gli altri agenti.
- **Cartella di condivisione**. Dopo di che genera una cartella e la condivide con tutti gli host in sola lettura.
- **Modifica lo start-up**. Aggiunge allo Start-Up dei vari PC nella rete lo script per lanciare i vari agenti e i loro demoni.

3.5 Funzionamento

Adesso verranno riportati i passaggi che compongono il funzionamento classico del sistema.

- L'agente di controllo si collega al database per scaricare le configurazioni di quel dominio autenticandosi tramite un codice. Dal database scarica inoltre i vari moduli e le varie dipendenze da eseguire sui vari computer del dominio
- A questo punto, leggendo il file di configurazione del dominio decide a quali agenti e su quali host far eseguire determinate funzionalità. Per distribuire il carico di lavoro controlla l'utilizzo della CPU dei vari host collegati alla rete e assegna i compiti a quello con un valore più basso.
- I moduli eseguiti sui vari host restituiscono un insieme di output (Vedi paragrafo Moduli). Per ogni modulo viene lanciato un timer e se dopo un certo tempo variabile da modulo a modulo esso non ha finito scatta un time-out e viene segnalato un errore. In questo caso l'Agente fa eseguire lo stesso modulo su un altro host.

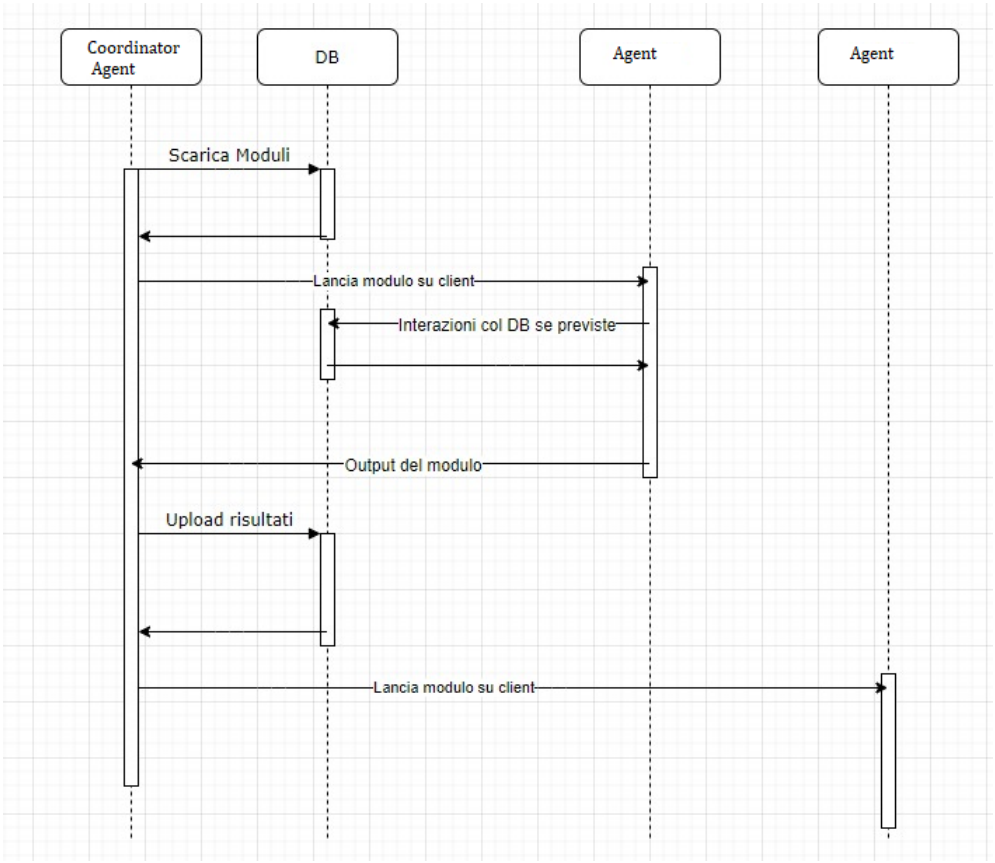

Figure 3.6: *Nell'immagine si può osservare il diagramma delle attività del sistema nel caso di funzionamento normale*

- I vari valori generati dal funzionamento dei moduli vengono raccolti e salvati sul server.

3.5.1 Disconnessione del Coordinator Agent

Come abbiamo già anticipato, in caso di disconnessione o fallimento nel dialogo con l'agente di controllo il sistema attua una serie di meccanismi per poter continuare a lavorare.

- Se i vari agenti falliscono nel comunicare con il Domain Controller allora tenteranno di ripristinare la connessione con internet impostando una configurazione di rete temporanea. I vari demoni passano quindi in modalità autonomia e cercano di rendere l'agente autonomo.
- Il demone **SaveMyInternet** cerca ora di trovare un nuovo DNS nella rete per poter ripristinare le varie funzionalità. In più inizierà a dialogare con gli altri agenti nella rete in modalità Peer2Peer.
- Il demone di gestione dei moduli dei vari agenti si attiva e rende l'agente autonomo nello scaricare e lanciare i propri moduli dal database.
- Il demone **ShareIsGood** cerca nella sotto rete le varie stampanti collegate in modo da poter continuare a utilizzarle e espone le proprie stampanti e le condivide su richiesta.

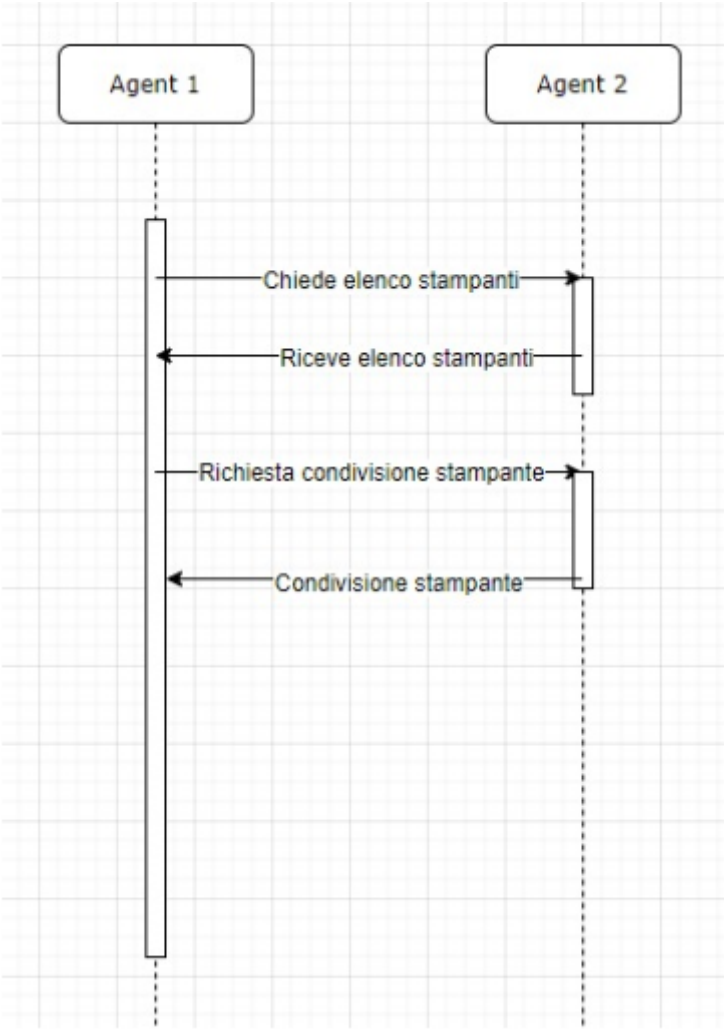

Figure 3.7: *Nell'immagine si vede come funziona il discovering delle stampanti e le interazioni tra i vari agenti*

4 Dettagli implementativi

4.1 Moduli creati

- Scan dispositivi tramite protocollo SNMP con lista di OID da richiedere e valori attesi, utile per valutare lo stato dei vari elementi nel Domain Controller di cui si conosce l'OID come ad esempio il livello del toner delle stampanti o lo stato di salute del raid di un NAS o di un SAN.
- Controllo dello stato di Backup del software Veeam, utile per ricevere un errore in caso in cui il backup non sia stato effettuato correttamente.

4.2 Realizzazione e tecnologie web

Il sito web è stato utilizzando utilizzando il linguaggio di programmazione web **React** per il lato client e **Node** per il lato server. Il Server web a sua volta si collega al database che si trova in cloud su **MongoDB** attraverso la tecnologia **Mongoose** per evere un accesso semplice ai campo dei documenti e un binding dinamico agli oggetti che definiscono tali documenti. Il sito, una volta realizzato, è stato pubblicato su **Heroku** al seguente link https://portaleweb.herokuapp.com

5 ISTRUZIONE PER IL DEPLOYMENT

5.1 REQUISITI

Il sistema è stato testato utilizzando un dominio 'tradizionale' seguendo le best pratice suggerite da Microsoft e utilizzando come sistema operativo Windows Server 2019 con l'installazione degli aggiornamenti fino a Novembre 2019. Come client sono stati utilizzate 2 macchine con Windows 10 versione 1909, entrambe partecipanti al programma Insider, una virtualizzata in lingua inglese e con OS ottimizzato per essere eseguita su Hyper-v, mentre l'altra in lingua italiana e installata su macchina fisica. Per il join delle macchine in dominio seguire la procedura 'standard', è consigliato aggiungere l'utente di dominio come amministratore locale della macchina (cosa che viene proposta in fase di join). Il programma necessita di PowerShell versione 4.0 o superiore (installata di default su Windows 10). Si consiglia, nel caso di sviluppo di propri moduli in Powershell, di evitare di introdurre comandi che necessitino di versioni di Powershell maggiori della 4 per evitare problemi di compatibilità. Si consiglia, nel caso di sviluppo di propri moduli in VBscript, di controllare la versione di DotNet installata sul dispositivo e di evitare di lanciare comandi non supportati. Si consiglia inoltre di iniziare un modulo in VBScipt con il comando On Error Resume Next e di ritornare l'errore come output al fine di evitare il blocco dello script e il conseguente timeout del server (il quale ritornerebbe un errore generico e poco utile per il debug del modulo). Il demone SaveMyInternet e il modulo Allarmi potrebbero avere malfunzionamenti in subnet con netmask diverso dallo standard 255.255.255.0

5.2 PROCEDURA INSTALLAZIONE

All'interno della cartella con il programma è presente uno script di installazione che permette di automatizzare alcune operazioni tra cui la generazione, condivisione e impostazione dei permessi di alcune cartelle, provvede inoltre al download e all'installazione di alcune dipendenze di cui il sistema necessita, tra cui il modulo mdbc necessario per la comunicazione col database. Lo script di installazione chiede un codice di dominio, questo deve essere precedentemente configurato sul database ed è considerato segreto (usare 1362 per test). Lo script di installazione NON provvede alla generazione del dominio, al join dei client, alla configurazione delle GPO e alla schedulazione dello script. Se lo script di installazione segnala degli errori interrompere l'installazione per evitare danni al sistema operativo, inoltre è sconsigliata l'installazione in ambienti di produzione in quanto il programma è ancora in fase di test.

La cartella contiene 2 file ps1, uno per il domain controller e uno per i client. Questi possono essere schedulati tramite lo scheduler di Windows o inseriti come script di startup dentro alle group policy di dominio. Entrambi gli script necessitano di permessi di amministratore locali nella macchina, per il server è richiesto l'esecuzione tramite utente appartenente al gruppo di amministratori di dominio. I due programmi sono indipendenti e l'installazione nel client è opzionale, fornisce infatti una serie di demoni che permettono di gestire alcune situazioni anomale causate da un problema nella rete o nel domain con-

troller. Per la distribuzione del programma per i client è consigliato utilizzare le GPO di Windows in modo da automatizzare la distribuzione su tutto il dominio (o eventualmente solo ad alcuni gruppi).

Attualmente sono disponibili solo due moduli, uno per il controllo del corretto avvenimento del backup attraverso il software Veeam e uno per gestire allarmi personalizzabili di device che supportano il protocollo SNMP. Il modulo per la gestione degli allarmi attualmente supporta il controllo del livello di toner nero delle stampanti Epson all'interno della rete locale, utilizza la versione 1 e 2 del protocollo SNMP con credenziali di default per la lettura(password "public"), ma è disponibile nel db un documento nel quale è possibile creare nuove regole per altri dispositivi come ad esempio NAS,SAN o server con OID pubblici (Fujitsu).

5.2.1 Esempio di regola

- nome:"EPSON Built-in 11b/g/n 10/100/1000 Print Server" (Valore restituito dall'oid .1.3.6.1.2.1.1.1.0 e che indica il nome del dispositivo)
- oid:"1.3.6.1.2.1.43.11.1.1.9.1.1" (Oid da andare a leggere) operazione:">" (come deve essere il valore letto rispetto alla soglia)
- soglia:"10" (soglia del valore)
- oggetto:"toner nero" (cosa si sta andando ad interrogare, verrà poi restituito nell'errore)

Sono attualmente supportate operazioni di tipo <,>,<> e = su valori interi, mentre per i valori di tipo stringa sono disponibili solo le operazioni = e <>

5.3 Procedura creazione nuovo modulo

Ogni modulo ha la sua tabella nel DB che deve contenere obbligatoriamente alcuni documenti:

- **ConfigurazionePrivata**: indica i siti e i device che devono eseguire il modulo.
- **Dipendenze**: eventuali dipendenze del modulo (opzionali), codificate in base64. Possono essere inseriti qualsiasi file, compresi eseguibili,foto e dll.
- **Programma**: il corpo del programma in Powershell o in VBScript, codificato il base64.

6 CONCLUSIONI

Ci riteniamo soddisfatti di questo progetto poiché il prototipo realizzato seppur con forti limitazioni cerca di rispettare tutti i requisiti che erano stati stabiliti nelle fasi iniziali di lavoro. Per poter stabilire il successo o meno di questo lavoro non possiamo affidarci a framework di test o meccanismi avanzati infatti il progetto mette insieme vari aspetti che non riescono a essere testati tutti insieme. Ci basiamo quindi sulle funzionalità e sulle caratteristiche che abbiamo dato al sistema. Tra i punti fondamentali troviamo la **scalabilità**. Per questa specifica caratteristica il nostro sistema si basa sulla autonomia dei vari agenti e questo permette di poterne schierare grandi numeri senza rallentare il sistema poiché non esiste un punto di centralizzazione per quanto concerne il controllo degli agenti. L'unico collo di bottiglia è rappresentato dal Database che è unico per tutti ma come già detto nell'ipotesi di una realizzazione di questo progetto consigliamo di utilizzare un database distribuito cloud. Noi utilizziamo una versione gratuita che comunque garantisce ottime prestazioni in termini di performance ma pagando si ottengono ottime prestazioni anche in termini di scalabilità. L'**autonomia** dei vari componenti è stato il punto centrale nello sviluppo di questo sistema. Ogni elemento, testato in varie situazioni è in grado di continuare a svolgere il suo lavoro. Tra queste situazioni abbiamo testato il casi di isolamento del dominio: nel caso cadesse la rete i vari agenti nel giro di qualche secondo si accorgono del problema e passano in modalità peer2peer per continuare a dialogare tra loro e tentano di ristabilitre la connessione in vari modi.

6.1 LAVORI FUTURI

In futuro sarebbe comodo fornire una pagina web per la creazione guidata di un nuovo modulo e di automatizzare alcune procedure che sono attualmente necessaria per il suo deployment, questo incoraggerebbe l'utilizzatore a sviluppare dei propri moduli personali. Sarebbe inoltre opportuno firmare i moduli attraverso una criptazione asimmetrica al fine di migliorare la sicurezza del sistema ed evitare l'esecuzione di codice arbitrario sulle macchine avendo il semplice accesso al database. Un altro aspetto migliorabile è la gestione dei demoni, attualmente sono monitorati e ripristinati attraverso un Job in Powershell, ma in futuro sarebbe sensato sfruttare il gestore di servizi nativo di Windows per delegarne il controllo e il ripristino in caso di errori. Ci sono molti margini di miglioramento per quanto riguarda lo script di installazione, questo infatti non fa un check profondo del sistema e un suo scorretto utilizzo potrebbe causare danni al sistema. Inoltre sarebbe opportuno averlo in una GUI invece che da linea di comando (anche se questo potrebbe limitarne l'utilizzo sui sistemi Core). Un altro miglioramento di questo sarebbe la creazione di un gruppo all'interno del dominio con la schedulazione dello script di installazione per i client. Un altro aspetto interessante ma molto ambizioso sarebbe il porting su Linux, infatti da poco tempo Powershell è diventato multi-piattaforma (dalla versione Core 6). Questo darebbe la possibilità al sistema di poter gestire reti eterogenee.

6.2 RIFLESSIONI PERSONALI

Il progetto, per quanto ambizioso, ci ha spinto a studiare i vari funzionamenti dei sistemi distribuiti odierni per capire quali meccanismi potevano essere utili come implementarli. Ovviamente questa cosa ci ha portato a confrontarci con varie soluzioni spesso irrealizzabili poiché necessitavano di risorse di cui non disponevamo. Ma malgrado questa cosa sia sicuramente un problema da un altro punto di vista è stato uno stimolo per ragionare su come implementare funzionalità simili con risorse e strumenti limitati. Ci riteniamo soddisfatti di quanto ottenuto, sicuramente il prototipo non rappresenta una soluzione concreta ma da spazio per ulteriori lavori e studi in tale direzione ponendosi come base di partenza.