**ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA**

**SCUOLA DI INGEGNERIA E ARCHITETTURA**

DIPARTIMENTO DI INFORMATICA SCIENZA E INGEGNERIA (DISI)

*Corso di Laurea Magistrale in Ingegneria Informatica*

**TESI DI LAUREA**

in

SISTEMI DISTRIBUITI M

# Integrazione di Framework Open-source per Monitoraggio Distribuito di Sistemi basati su Redhat Enterprise Linux

CANDIDATO:
Monteferrante Francesco

RELATORE:
Chiar.mo Prof. Ing. Paolo Bellavista

CORRELATORI:
Fabio Alvisi
Chiar.mo Prof. Ing. Antonio Corradi

Anno Accademico 2015/2016

Sessione III

# Indice generale

# Introduzione

Il lavoro svolto durante questo periodo di tesi, presso l'azienda NoemaLife di Bologna, presenta come concetto chiave il monitoraggio. In particolar modo, è stato effettuato uno studio intenso sul sistema di monitoraggio utilizzato dall'azienda in questione (chiamato Skynet), cercando di trovare una soluzione per rendere più efficiente il lavoro svolto dal personale del customer service, ovvero, soddisfare, nel modo più comodo, semplice e veloce possibile, le richieste dei clienti. La sfida, in questa circostanza, è stata quella di trovare un modo per cercare di automatizzare il salvataggio di dati monitorati, in modo da evitare che questo lavoro venisse svolto manualmente comportando perdite di tempo e rallentamenti di operazioni di intervento nei confronti dei clienti.

I sistemi di monitoraggio sono molto importanti perché permettono di evitare spiacevoli situazioni, come il crash dell'intera rete IT, attraverso degli interventi preventivi. In questo modo, si può intervenire tempestivamente per risolvere un determinato problema garantendo la continuità del servizio e la soddisfazione da parte dei clienti. L'obiettivo principale del monitoraggio è quello di controllare che tutto funzioni in modo corretto per evitare perdite aziendali in termini economici, di affidabilità e di credibilità.

Il progetto consiste nella realizzazione di un server Web che ha il compito di allineare specifici dati, monitorati dall'azienda attraverso il sistema di monitoraggio e non solo, con il SysAid CMDB e nella realizzazione di service che permettono di aggiungere ulteriori informazioni con lo scopo di semplificare le necessità aziendali continuando a soddisfare le richieste dei clienti.

SysAid CMDB (Configuration Management DataBase) è una repository di informazioni relative a tutti i componenti che costituiscono un sistema; infatti, lo potremmo definire come un database che mantiene memorizzate tutte le informazioni relative agli assert di rete, prodotti software e catalog item e le relazione che possono esserci fra di loro.

Tra i diversi service realizzati, ci sono quelli che vengono utilizzati per arricchire le informazioni iniziali che sono state memorizzate dal server Web all'interno del SysAid CMDB e altri che servono per il monitoraggio e controllo di parametri di interesse per l'azienda.

Il server ha il compito di controllare i dati monitorati e creare una CI (Configuration Item) all'interno del SysAid CMDB inserendo le informazioni iniziali. Una volta che il server Web ha creato il CI corrispondente con tali informazioni, si possono aggiungere dettagli attraverso i diversi service. In particolar modo, quello che bisogna fare è inserire il service di interesse su Skynet e farlo girare sulla macchina di interesse. Quando il service viene inserito, questo, ad intervalli di tempo regolari, eseguirà sulla macchina fornendo i risultati, i quali verranno inseriti nel corrispondente campo all'interno del SysAid CMDB.

Infine, l'intero progetto è stato integrato in un altro sistema di monitoraggio, ovvero Zabbix, per confrontare i due sistemi in termini di prestazioni e qualità e verificare la fattibilità del lavoro, cioè constatare che il server Web e i service hanno una base solida per poter essere utilizzati anche in altri sistemi.

Per quanto riguarda la struttura della tesi, questa è costituita da 5 capitoli. Nel capitolo 1 vengono presentati gli obiettivi di alto livello del lavoro, il perché sono rilevanti in termini generali e aziendali, un po' di storia dell'azienda e il motivo della scelta di utilizzare un sistema di monitoraggio realizzato internamente piuttosto che adottare una soluzione commerciale. Nel capitolo 2 viene descritto il sistema di monitoraggio Skynet, in particolar modo quali sono le parti che lo compongono e la descrizione di ciascuna di esse. Nel capitolo 3 viene descritto, invece, un altro sistema di monitoraggio che prende il nome di Zabbix, quindi, quali sono le caratteristiche, i concetti chiave, configurazione degli elementi principali e requisiti di sistema ed installazione. Nel capitolo 4, viene discusso il progetto in termini teorici, ovvero qual è l'architettura generale e quali sono le sue funzionalità. Nel capitolo 5, infine, vengono descritte le scelte implementative che sono state fatte per la realizzazione dell'intero progetto, gli strumenti utilizzati, la realizzazione dello stesso ed, infine, i risultati che si sono ottenuti.

# 1 – Automatizzare per maggiore qualità

Il lavoro svolto consiste nella realizzazione di un server Web che ha il compito di allineare specifici dati monitorati con il SysAid CMDB dell'azienda e nella realizzazione di servizi che permettono di dettagliare le informazioni allineate con lo scopo di semplificare le necessità aziendali continuando a soddisfare i clienti.

SysAid CMDB (Configuration Management DataBase) è una repository di informazioni relative a tutti i componenti che costituiscono un sistema; infatti, lo potremmo definire come un database che mantiene memorizzate tutte le informazioni relative agli assert di rete, prodotti software e catalog item e le relazione che possono esserci fra di loro.

SysAid CMDB aiuta a comprendere la relazione fra i componenti e tracciare le loro configurazioni. Tutto questo può essere utile per mantenere in memoria le informazioni relative a diversi tipi di item.

Per comprendere meglio, facciamo un esempio "di uso quotidiano". SysAid CMDB può essere utilizzato da una stazione taxi per tener traccia dei diversi veicoli, dei conducenti, destinazioni, garage e altri componenti ed informazioni rilevanti.

Alcuni esempi di item di configurazione che possono essere settati e gestiti con SysAid CMDB:

- hardware (inclusi componenti di rete dove rilevante);
- sistemi software, inclusi i sistemi operativi;
- business system – custom built applications;
- pacchetti software;
- prodotti database;
- database fisici;
- environments;
- documentazione di configurazione, ad esempio specifiche di sistema e di interfaccia, accordi di manutenzione, ecc…;
- altre risorse come utenti, contratti, fornitori, ecc…;
- altre documentazioni come IT business process, procedure, ecc….

Tramite questo server, si è in grado di creare una entry iniziale all'interno del SysAid CMDB relativa al dispositivo monitorato, che successivamente verrà maggiormente dettagliata da specifici servizi.

Tali servizi hanno il compito di controllare ciò che è di interesse per un determinato dispositivo, ad esempio la versione del database, parametri relativi alla memoria e altre informazioni necessarie.

Tutti questi valori, in seguito, andranno a riempire la corrispondente entry che è stata creata dal server Web.

In altre parole, quello che possiamo dire è che il server ha il compito di creare le entry del SysAid CMDB inserendo le informazioni iniziali per ciascun dispositivo, mentre i servizi hanno lo scopo di arricchire le informazioni relative a ciascuna entry.

Facendo un esempio, supponiamo che venga monitorata una nuova macchina; il server Web si occupa di individuare la nuova macchina monitorata, di verificare se questa risulta essere già presente nel SysAid CMDB e se così non fosse allora la memorizza inserendo solo il suo nome, il suo indirizzo IP e il suo alias. A questo punto, ci sono solo le informazioni iniziali relative al dispositivo monitorato. In base a ciò che è di interesse per quel dispositivo, si va ad aggiungere il corrispondente servizio. Supponendo di essere interessati alla versione del database posseduto da quella macchina, allora si va ad inserire, nel sistema di monitoraggio, il corrispondente servizio che esegue sulla macchina di interesse. Tale servizio controllerà costantemente la versione del database posseduta dalla macchina e la memorizzerà nella entry del SysAid CMDB corrispondente alla macchina stessa, in modo che i dati risultano essere sempre aggiornati e pronti all'uso.

Tutto questo viene fatto per rendere più semplice l'assistenza dei clienti; infatti, ogni volta che un nuovo dispositivo viene monitorato (di conseguenza viene inserito nel sistema di monitoraggio), questo deve essere memorizzato all'interno del SysAid CMDB per consentire al customer service di poter accedere al dispositivo, conoscere le sue caratteristiche, le sue proprietà e capire quale sia il problema che è stato comunicato dal cliente che possiede il corrispondente dispositivo monitorato.

Questo lavoro deve essere fatto a mano, quindi risulta essere particolarmente fastidioso, noioso e soprattutto richiede un'infinità di tempo se, come succede sempre quando si ha a che fare con le aziende, si devono monitorare centinaia o migliaia di dispositivi.

Questo problema viene risolto attraverso il server Web che automaticamente va a controllare se tali dispositivi sono già presenti all'interno del SysAid CMDB e se così non fosse va a creare una entry corrispondente a quel dispositivo.

Il controllo di verifica di presenza o meno di un dispositivo all'interno del SysAid CMDB viene fatto attraverso delle richieste al Web Services di SysAid; di conseguenza, viene fatta una richiesta SOAP in cui viene richiesta la lista dei dispositivi che sono memorizzati nel SysAid CMDB. Successivamente, il server, ricevuta la corrispondente risposta, effettuerà un controllo per verificare se i dispositivi correntemente monitorati sono tra quelli presenti nel SysAid CMDB oppure no. Se non sono presenti, il server effettua un'altra richiesta SOAP che non è altro che la scrittura dei dispositivi all'interno del SysAid CMDB.

Tutto questo lavoro viene fatto in pochissimo tempo (si parla di meno di 10 secondi) a discapito del tempo impiegato da una persona (ore se non giorni) per andare a memorizzare ciascun dispositivo.

Tale server permette di velocizzare la memorizzazione di quelle informazioni che sono ritenute fondamentali da aziende che si occupano di monitoraggio.

Il server, inoltre, non si occupa di allineare solo i dispositivi monitorati con il SysAid CMDB aziendale; infatti, permette di memorizzare anche altre informazioni come specifici servizi presenti sul dispositivo, i tipi di database e specifici prodotti realizzati dall'azienda.

L'aspetto importante è che il server Web può, comunque, essere esteso in base alle esigenze aziendali proprio perché consente di automatizzare il salvataggio di dati monitorati da un sistema di monitoraggio all'interno del SysAid CMDB aziendale. I servizi, invece, permettono di rendere più dettagliata la descrizione dei dispositivi semplificando la vita di coloro che hanno il compito di risolvere i problemi e soddisfare le richieste di un determinato cliente.

Il lavoro è stato svolto presso l'azienda Noemalife con sede a Bologna, quindi, il tutto è incentrato sulle loro esigenze.

NoemaLife è un gruppo internazionale fondato in Italia nel 1996; dal 2006 al 2016 è stato quotato in borsa italiana ed è divenuto il leader europeo nel mercato dell'informatica clinica ospedaliera, grazie allo sviluppo software all'avanguardia, ricerca avanzata ed eccellenza nel servizio al cliente.

I software realizzati da questa azienda sono destinati alle organizzazioni sanitarie sia pubbliche che private contribuendo a rendere il sistema sanitario più efficiente attraverso la riduzione dei costi ed il miglioramento della qualità del processo di cura del paziente e dei servizi.

Le soluzioni proposte da NoemaLife contribuiscono all'ottimizzazione del flusso di lavoro delle strutture sanitarie a livello dipartimentale, ospedaliero e territoriale, introducendo una nuova modalità di gestione integrata del processo clinico in tutti i principali ambiti di applicazione:

- **Area Clinica**: NoemaLife ha introdotto in Italia la Cartella Clinica Elettronica sin dal 2001 sviluppando, in oltre quindici anni di lavoro e a fianco delle strutture sanitarie, una nuova filosofia di prodotto per la gestione integrata dei processi clinici creando un'infrastruttura applicativa focalizzata sul paziente ad ogni livello organizzativo, dipartimentale, ospedaliero e territoriale.

  Le soluzioni software proposte per l'area clinica permettono la gestione sanitaria e amministrativa del paziente lungo l'intero percorso ospedaliero, migliorando il servizio, ottimizzando i costi e riducendo i rischi terapeutici. Tutto questo facilita l'integrazione fra le diverse competenze professionali e fornendo una base informativa consistente e comune.

  La continua ricerca di nuove soluzioni, con l'obiettivo di migliorare il lavoro quotidiano di medici e personale sanitario, vede l'impiego delle tecnologie più avanzate; dalle console interattive touch-screen alle soluzioni mobili che permettono un'efficiente attività clinica direttamente al letto del paziente;

- **Area Diagnostica**: NoemaLife accompagna ed anticipa l'evoluzione e la crescita dei dipartimenti di diagnostica da oltre venti anni; suo è il modello concettuale di architettura informatica chiamato LABORATORIO LOGICO UNICO (LLU), conosciuto anche come Laboratorio di Area Vasta, che consiste in una sorta di laboratorio virtuale con l'obiettivo di integrare, su un'unica piattaforma, più laboratori di analisi distribuiti su una determinata area territoriale per ottimizzare le risorse economiche e i processi di diagnostica. Grazie all'esperienza maturata, le soluzioni proposte non rispondono solamente alle esigenze più consolidate, come quelle dei Laboratori di Analisi e dei Dipartimenti di Anatomia Patologica, ma comprende anche soluzioni software mirate alla gestione di ambiti in piena evoluzione.

La competenza acquisita nell'ottimizzazione dei flussi di lavoro si rivela fondamentale anche nel rispondere all'esigenza di valutare l'appropriatezza delle richieste di analisi, infatti, l'azienda dispone di sistemi esperti in grado di verificare la correttezza delle richieste da qualunque sistema esterno di accettazione e di supportare il processo decisionale attraverso l'immediata segnalazione di eventuali anomalie.

Le soluzioni di diagnostica NoemaLife comprendono anche innovativi software per il Dipartimento di Immagini in grado di gestire l'intero processo di Imaging, rispondendo alle esigenze di interazione, multimedialità e velocità d'esecuzione;

- **Comunità e Territorio**: Il nuovo paradigma verso cui si sta spostando la cura del cittadino è quello di ridurre al minimo la necessità di accedere alle strutture ospedaliere. Cura, assistenza, ritiro referti, riabilitazione e, per quanto possibile, anche la diagnosi, devono essere effettuate sul territorio ottenendo così una riduzione dei costi fissi di struttura ed un sistema globalmente più sostenibile che migliori la qualità di vita della popolazione.

  NoemaLife fornisce in questo ambito una nuova generazione di sistemi informativi di supporto alla cura domiciliare e all'assistenza del paziente, mobili e flessibili, in grado di essere adottati sia nelle corsie di un ospedale, che dal personale di assistenza domiciliare. I programmi avanzati di screening assicurano l'efficacia di una prevenzione a livello territoriale e comprendono pianificazione degli inviti, simulazione degli scenari, gestione delle agende, interfaccia per il front office telefonico e produzione di statistiche. Allo stesso modo, i progetti di telemedicina realizzati da NoemaLife, non si limitano a fornire strumenti di consulenza, ma anche di gestione dell'intero flusso operativo come richiesta di intervento, pianificazione della consulenza, inclusione nel referto delle indicazioni cliniche ricevute e diagnosi da remoto.

  NoemaLife risponde, inoltre, alle nuove normative che prevedono l'obbligatorietà da parte delle aziende del Servizio Sanitario di adottare procedure telematiche per consentire il pagamento delle prestazioni erogate e la consegna dei referti, offrendo anche innovative soluzioni facili da usare e vicine al cittadino per il ritiro referti da qualunque luogo e a qualunque ora. Tutto questo può essere fatto attraverso portali internet, applicazioni per iPhone e postazioni dislocate sul territorio. Strumenti che migliorano il servizio

facilitano l’accesso alle strutture dei cittadini, rendendo possibile ottenere la stampa del referto indipendentemente dal centro prelievi, dal presidio ospedaliero e dalla sede di esecuzione degli esami.

Il Gruppo collabora con le migliori Università e con prestigiosi Centri di Ricerca nel mondo, partecipando attivamente a numerosi Progetti Internazionali e Comunitari.

Nel 2011, NoemaLife ha acquisito la maggioranza relativa del capitale del gruppo francese Medasys, leader del settore in Francia, a sua volta quotato alla borsa di Parigi, consolidando, in tal modo, il proprio ruolo di principale fornitore europeo di soluzioni informatiche di processi clinici.

Nel Giugno 2016, Dedalus S.p.A. ha acquisito la maggioranza di NoemaLife S.p.A. L'unione delle due organizzazioni internazionali ha segnato la nascita del primo operatore in Italia, tra i principali attori sulla scena europea.

L’azienda presenta diversi settori attraverso cui vengono gestiti i servizi offerti:

- IT Performance Monitoring che si occupa della pianificazione dei backup, del monitoraggio pro-attivo, ottimizzazione delle infrastrutture e gestione della sicurezza e della privacy;
- Database si occupa, invece, della gestione dei dati memorizzati nei database, dell’ottimizzazione delle licenze Oracle, Patch Analysis, Monitoring e Trace, analisi statistica delle prestazioni e della Capacity planning;
- IT Assessment consente di effettuare l’aggiornamento del client server, il backup e ripristino del client, IT fleet management, gestire il data center, di effettuare interventi e presidio onsite, di gestire i server room e la sistemistica;
- Progettazione e consulenza tratta, invece, la business continuity, disaster recovery, si occupa della high availability, della gestione dei rischi, data assestment e delle virtualizzazioni;
- Help Desk presenta strumenti di supporto personalizzati per fornire, all’utente o al cliente, assistenza relativa ai prodotti o servizi informatici/elettronici dell’azienda.

Il settore in cui ho operato in questo periodo è quello dell’IT Performance Monitoring, in particolar modo sul monitoraggio. Gli obiettivi dell’azienda, in questo ramo, è quello di monitorare quello che può essere monitorato; infatti, il tutto dipende dalla tipologia del contratto stipulato con uno o più clienti.

Nel momento in cui un cliente entra in contatto con l'azienda, questo può decidere di comprare prodotti o servizi applicativi NoemaLife. Se si limita all'acquisto dei servizi applicativi, il settore si occupa di monitorare solo quelli, ad esempio, stato della Java Virtual Machine di Tomcat, di JBoss o di altre applicazioni Java e di eseguire i check basilari sulla macchina per verificare l'uso della cpu, della ram, lo spazio del disco; questi ultimi vengono fatti per sapere se un servizio applicativo può essere installato e messo in esecuzione sulla macchina. È chiaro che dal momento in cui un cliente richiede uno o più servizi applicativi, l'azienda ottiene l'accesso alle macchine su cui tali servizi devono girare in quanto devono effettuare il controllo e, successivamente, il monitoraggio.

Un aspetto importante, inoltre, è che l'azienda si occupa di monitorare tutto ciò che è a loro carico; in caso contrario non se ne occupano. Per comprendere meglio facciamo un esempio: consideriamo la gestione del database Oracle; se questo risulta a carico del cliente, l'azienda non se ne preoccupa minimamente, ma nel momento in cui il cliente decide di firmare un contratto con l'azienda in cui richiede anche installazione e assistenza del database Oracle, allora verrà effettuato anche il monitoraggio; questo perché l'azienda ha necessità di sapere quello che succede in quanto si occuperà di fornire anche assistenza.

Nel caso del database, quindi, verranno monitorati la table space per valutare lo spazio rimasto libero, il numero di sessioni, numero di processi, uso dei redo log, verifica di lock sulle tabelle, ecc…; in caso di sistemistica, si va a monitorare le temperature, lo status del raid dei dischi, lo stato dell'infrastruttura VMware e così via.

Ricapitolando, quindi, cosa monitorare viene stabilito all'atto della stipulazione del contratto. Solo in casi estremamente eccezionali e rari è possibile intervenire dopo aver sottoscritto il contratto; questo perché richiede una procedura molto complessa e particolarmente lunga in cui bisogna valutare se l'intervento può essere fatto, se è necessario dover ricreare l'infrastruttura da capo, i tempi di lavoro e molto altro ancora.

Altro aspetto importante riguarda il sistema di monitoraggio utilizzato dall'azienda e capirne il motivo del suo utilizzo.

Noemalife nel 2011 decise di ampliare la sua offerta commerciale proponendo al cliente una soluzione all-in-one che comprendesse oltre all'installazione e

manutenzione dei propri software, specifici per l'ambito sanitario, anche una manutenzione di tipo sistemistico specialistico.

Nell'ambito della proposta è stato individuato, come elemento cardine, l'introduzione di un sistema di monitoraggio utile all'intervento tempestivo, tramite allarmi, nonché alla notifica preventiva sull'andamento delle performance generali dell'installato e specifiche per i propri applicativi.

La ricerca del software in un ambito di mercato, piuttosto amplio, è ricaduta su di una soluzione basata su strumenti Open Source, ma pacchettizzata in un' appliance hardware denominata NetEye prodotta e distribuita dall'azienda Würth Phoenix.

Come detto NetEye è una piattaforma di IT System Management centralizzata dove convergono tecnologie Open Source volte a soddisfare le esigenze per il monitoraggio informatico. NetEye comprende una suite di software Open Source di cui offre l'immediata operatività senza richiedere, a chi lo utilizza, il know-how necessario per l'installazione, l'integrazione e la manutenzione.

Tra le funzionalità chiave del prodotto scelto emersero:

- **Incident Management**: prevede un monitoraggio amplio sulle più svariate piattaforme software e hardware;
- **Problem Management**: gestione ottimizzata dei problemi tramite un'interfaccia web intuitiva, nonché dinamiche di dipendenze tra i diversi host;
- **Configuration Management**: utile alla semplificazione nella gestione delle configurazioni, a volte anche molto complesse, di strumenti come Nagios e gestione di controlli temporali, nonché la configurazione semplificata di SMS e log viewer;
- **Dashboard**: una comoda interfaccia Web per la gestione del livello di servizio, dati storici alert e SLA.

Per ragioni di costi fissi e di scarsa flessibilità nel gestire controlli specifici legati soprattutto al mondo applicativo Noemalife, fu creato, dal gruppo sistemistico di Noemalife, uno spin-off del prodotto, che venne inseguito denominato Skynet, prendendo spunto dall'as-is di NetEye ed implementando nuove features nonché introducendo controlli a largo spettro su componenti proprietari dell'azienda. Un altro aspetto legato a questa scelta fu la maggior presenza, presso clienti, di infrastrutture basate su Hypervisor come VMware vsphere, che permettevano, creando appliance virtuali, di svincolarsi dal verificarsi di guasti hardware sull'appliance NetEye.

Questo ha facilitato anche la fase di deploy del prodotto che poteva essere preconfigurato e successivamente rilasciato sull'infrastruttura cliente con procedure standard.

Il primo rilascio, da parte di NoemaLife, del nuovo sistema di monitoraggio Skynet 1.0, che andrà nel tempo a sostituire la presenza di NetEye presso l'installato clienti, avviene nel 2014.

Tutt'ora Skynet viene gestito e mantenuto dal reparto ITOPS Noemalife, con continue implementazioni sulle esigenze che nascono su specificità dei clienti, piuttosto che con integrazioni da e verso il sistema di ticketing di Noemalife (SysAid).

Ad oggi ci si sta interrogando internamente sull'effort quotidiano investito internamente per la manutenzione, upgrade e la facilità/templating legati ai controlli che interessano l'installo clienti.

Noemalife, ancora in una fase embrionale, sta valutando soluzioni alternative che possano sostituire nel tempo e migliorare l'attuale sistema di monitoraggio. Il maggior punto d'attenzione è avere una soluzione con facile deploy, che possa ricoprire qualsiasi necessità di monitoring che nel tempo potrebbe emergere e soprattutto che sia mantenibile con un minor effort interno.

Ad oggi, la fase di PoC (Proof of Concept) che si sta portando avanti è basata su Zabbix, oramai leader in ambito Enterprise, ricordando, infatti, che tale sistema risulta essere il secondo prodotto nel quadrante di Gartner subito dopo HP Openview.

Quest'ultimo è stato scartato per ragioni di costo, mentre Zabbix, ad oggi, si pone l'obiettivo di ricoprire qualsiasi esigenza di controlli utilizzando i propri agent server e ove necessario richiamare script esterni. Questo è uno dei motivi principali per cui si sta valutando l'utilizzo di Zabbix oltre a considerare l'aspetto più importante di tutti che è quello di avere continuità con l'attuale Skynet. Inoltre una delle caratteristiche interessanti sviscerate durante le fasi del PoC è stata la scoperta dei template che permetterebbero una maggior velocità e flessibilità nell'applicazione dei controlli sull'infrastruttura.

Altro aspetto importante è che bisogna fare i conti anche con i vantaggi e svantaggi che derivano dall'adottare una soluzione interna o adottarne una commerciale.

Partendo con la prima; è chiaro che il beneficio principale che deriva nell'adottare una soluzione built-in è l'elevatissimo livello di personalizzazione. Anche se una

soluzione commerciale potrebbe soddisfare molte necessità aziendali, una soluzione interna potrebbe risultare più efficiente in quanto potrebbe soddisfare esattamente le specifiche di business senza dover aggiungere componenti extra. Questo permette di aver un maggior controllo, un'interfaccia più familiare e di facile utilizzo. Infine, poiché il software è stato sviluppato da un team interno, risulta essere più facile avere supporto; infatti, tecnici che hanno creato il sistema saranno in grado di capire qualsiasi situazione e di capire anche piccole sfumature a differenza di tecnici esterni.

Di contro, però, una soluzione interna potrebbe essere meno sofisticata in quanto i tecnici interni potrebbero non avere determinate conoscenze; questo potrebbe richiedere il supporto di consulenti esterni che non conoscono la soluzione adottata e ciò potrebbe essere dannoso per il software. Inoltre, bisogna tener presente che scelte interne potrebbero non essere scalabili ed essere difficili e fastidiosi nell'aggiornamento; inoltre, non sempre si riesce ad andare a passo con i tempi e questo potrebbe portare ad ulteriori spese per l'installazione di un nuovo software.

Per quanto riguarda una soluzione commerciale, invece, un aspetto particolarmente importante è rappresentato dal fatto che un software del genere sarà ampiamente testato e utilizzato da altre aziende; questo potrebbe consentire un'integrazione veloce e molto semplice. Di contro, però, il software commerciale potrebbe accumulare elevati costi di supporto e manutenzione. Al fine di risolvere eventuali problemi tecnici, ci potrebbero anche essere tempi di attesa e supporto particolarmente lunghi che potrebbero rendere le operazioni molto lente ed aumentare i costi.

Tutte queste informazioni devono essere prese in considerazione e valutate molto attentamente quando si vuole realizzare o comprare un software e nel caso specifico un sistema di monitoraggio. Anche per questi motivi, l'azienda sta valutando se effettuare il passaggio da una soluzione interna come Skynet ad una commerciale come Zabbix.

Ritornando al lavoro svolto, il server Web è stato realizzato per memorizzare quei dati che loro ritengono fondamentali per i loro scopi, pertanto il tutto potrebbe cambiare da azienda ad azienda.

Una cosa importante da dire, però, è che la base del server Web è abbastanza solida così come la realizzazione dei servizi. Infatti, in seguito alla loro realizzazione è stato effettuato un processo di integrazione degli stessi su un altro sistema di monitoraggio, Zabbix, ottenendo importanti risultati; infatti, è stato possibile passare da un sistema

di monitoraggio all'altro senza eccessive modifiche di codice sorgente, dimostrando così come tale lavoro potrebbe diventare di utilizzo comune per altre aziende che adoperano un sistema di monitoraggio differente.

# 2 - Skynet

Skynet è il sistema di monitoraggio utilizzato dall'azienda Noemalife S.p.a. con sede Bologna presso cui è stato effettuato il tirocinio/tesi.

Tale sistema consente di monitorare gli host presso i diversi clienti, in particolar modo i servizi di rete, le risorse di ciascuna macchina e molto altro.

Skynet è una soluzione interna, nel senso che è stata realizzata dal personale che lavorava presso l'azienda (e ora non più; infatti il sistema di monitoraggio risulta essere piuttosto vecchio) e si basa su tre elementi fondamentali:

1. Nagios: il sistema di monitoraggio vero e proprio;
2. Thruk: interfaccia grafica per interagire con Nagios;
3. Monarch: interfaccia di configurazione attraverso cui configurare i servizi, le macchine e molto altro.

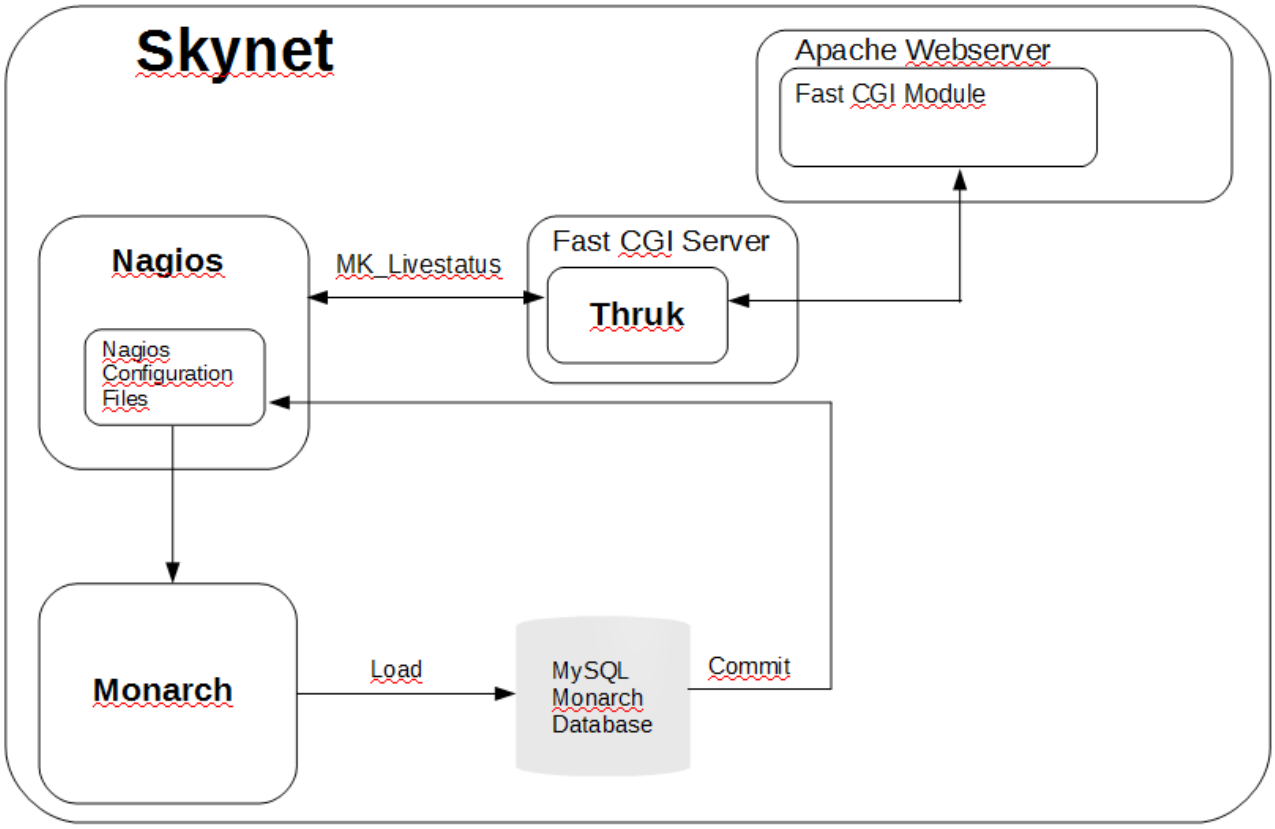



*Figura 1: Architettura Skynet*

## 2.1 – Nagios

Nagios è un programma Open Source per il monitoraggio di sistemi e della rete. Consente di monitorare host e service che vengono specificati, di lanciare allarmi nel caso di malfunzionamenti e, di conseguenza, intervenire tempestivamente. Diverse sono le caratteristiche che contraddistinguono tale sistema:

- monitoraggio di service di rete (SMTP, POP3, HTTP, NNTP, PING, ecc…);
- monitoraggio delle risorse di rete (carico del processore, memoria utilizzata, ecc…);
- possibilità di realizzare dei plugin personalizzati che consentono agli utenti di sviluppare dei propri service/host check;
- service/host check paralleli;
- possibilità di definire gerarchie fra gli host in modo da individuare e distinguere macchine che sono "down" da quelle che sono irraggiungibili;
- definizioni di contatti verso cui notificare un malfunzionamento quando si verificano dei problemi o quando gli stessi vengono risolti (attraverso email, sms o metodi user-defined);
- possibilità di definire gestori di eventi che si attivano durante eventi di host o service e che consentono di risolvere il problema in modo pro-attivo;
- creazione automatica di file di log;
- supporto per implementare il monitoraggio di host ridondanti;
- interfaccia web opzionale che permette di visualizzare lo stato delle macchine, le notifiche, problemi che si sono verificati e molto altro.

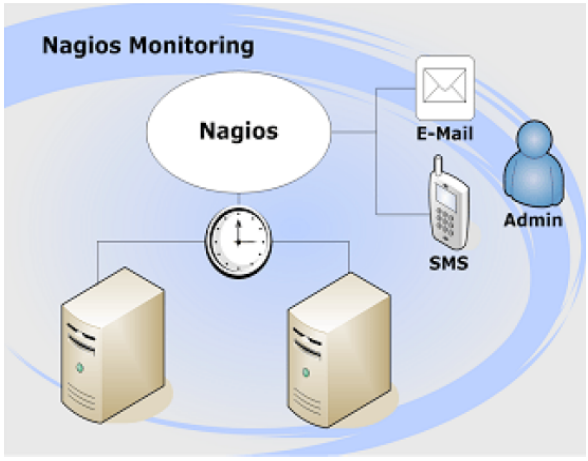



*Figura 2: Schema di funzionamento*

I componenti principali di Nagios sono:

- *Monitoring daemon*: è un processo, lato server, che implementa le funzionalità di monitoraggio e di notifica di allarmi;
- *Web Interface*: è un'applicazione Web, installabile su un qualsiasi server Web che supporti lo standard CGI (tipicamente viene utilizzato il server Apache), che consente di visualizzare lo stato degli host/service monitorati anche da postazioni diverse dal server Nagios.

### 2.1.1 Concetti chiave

In questo paragrafo andiamo a vedere quali sono i concetti chiave presenti in Nagios e come configurarli. Una precisazione: i concetti spiegati qui di seguito sono considerati importanti dal punto di vista del sottoscritto, in quanto il lavoro svolto durante questo periodo si è incentrato su questi elementi. Il mondo di Nagios risulta essere vastissimo pertanto per informazioni relativi ad altri concetti è possibile consultare la documentazione ufficiale su cui ho effettuato i miei studi [1].

#### 2.1.1.1 – Plugin

A differenza di molti altri tool di monitoraggio, Nagios non include nessun meccanismo interno per il controllo dello stato di host e service sulla rete; infatti, Nagios si basa su "programmi" esterni (chiamati plugin) che fanno tutto il lavoro "sporco".

I plugin sono dei compilati, eseguibili o script (Perl, bash, ecc…) che possono essere messi in esecuzione attraverso una linea di comando per controllare lo stato di un host o di un service. Nagios sfrutta il risultato di tali plugin per determinare lo stato corrente di ciò che si sta monitorando.

Nagios eseguirà un plugin ogni volta che c'è bisogno di controllare lo stato di una macchina o di un servizio; tale plugin eseguirà "qualcosa" e fornirà il risultato al sistema di monitoraggio, il quale processerà l'output in base alle necessità. Ad esempio, se lo stato risulta essere OK non verrà fatto nulla; in caso contrario Nagios metterà in esecuzione il gestore degli eventi, invierà le notifiche, ecc….

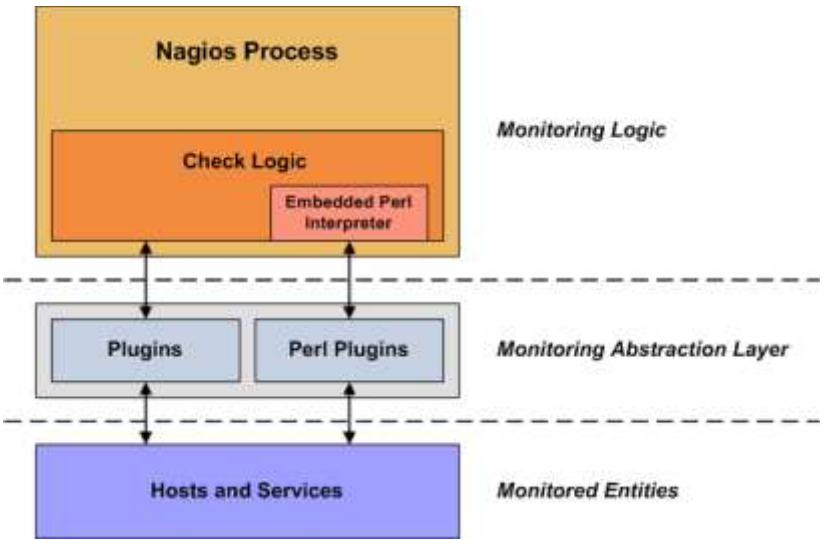



*Figura 3: Architettura Nagios con livello astratto di plugins*

I plugin agiscono come un livello astratto fra la logica di monitoraggio presente nel demone di Nagios e le macchine e servizi attualmente monitorati. Il vantaggio di avere un'architettura di questo tipo è che si può monitorare tutto ciò che passa per la testa. Se si riuscisse ad automatizzare il processo di controllo di un qualcosa, allora si è in grado di monitorarlo con Nagios.

Nagios fornisce molti plugin standard che permettono di monitorare:

- servizi di rete come HTTP, POP3, IMAP, FTP, SSH, DHCP;
- risorse come carico di CPU, utilizzo del disco, utilizzo della memoria, numero di utenti correntemente connessi;
- server Unix/Linux, Windows e Netware;
- Routers e Switches;
- molto altro ancora.

Lo svantaggio, invece, di avere questo tipo di architettura è che Nagios non ha assolutamente idea di ciò che si sta monitorando; infatti, tale sistema di monitoraggio non comprende le specifiche di ciò che si sta controllando, ma traccia solamente i cambiamenti di stato delle risorse. Di conseguenza, si può affermare che solo i plugin sono in grado di conoscere esattamente cosa si sta monitorando e come vengono eseguiti i controlli.

#### 2.1.1.2 – Macros

Una delle caratteristiche principali che rende Nagios così flessibile è la possibilità di utilizzare le macro nella definizione dei comandi; infatti, prima dell'esecuzione di questi, Nagios rimpiazza ogni macro che trova all'interno del comando con il

corrispondente valore. La "sostituzione" delle macro viene fatta per tutti i tipi di controlli che Nagios può fare: host/service check, notifiche, gestione degli eventi, ecc….

Facciamo un esempio per capire meglio come funzionano le macro. Supponendo di utilizzare il seguente host:

**define host{**

**host_name linuxbox**

**address 192.168.1.2**

**check_command check_ping**

**...**

**}**

su cui vogliamo eseguire il seguente comando:

**define command{**

**command_name check_ping**

**command_line /usr/local/nagios/libexec/check_ping -H $HOSTADDRESS$ -w 100.0,90% -c 200.0,60%**

**}**

l'esecuzione del command sul nostro host sarà uguale a:

**command_line /usr/local/nagios/libexec/check_ping -H 192.168.1.2 -w 100.0,90% -c 200.0,60%**.

Come si può notare, il valore della macro verrà sostituito con il corrispondente valore assunto nell'host su cui si sta effettuando il controllo. Questo fa si che si può utilizzare la definizione di un singolo comando per controllare un numero illimitato di host e di service; infatti, il comando può rimanere uguale per ogni macchina o servizio in quanto Nagios effettuerà la sostituzione, con il corrispondente valore, prima dell'esecuzione.

In Nagios, inoltre, c'è la possibilità di creare delle macro personalizzate che devono essere denominate nel seguente modo:

- $_HOST*varname*$: per la definizione di macro personali relative agli host;

- $_SERVICE*varname*$: per la definizione di macro personali relative ai service;
- $_CONTACT*varname*$: per la definizione di macro personali relative ai contact.

Ad esempio, andiamo a definire una variabile, chiamata _MACADDRESS, all'interno della definizione di un host:

**define host{**

**host_name linuxbox**

**address 192.168.1.1**

**_MACADDRESS 00:01:02:03:04:05**

**...**

**}**

La variabile appena creata potrà essere acceduta, per quanto detto prima, attraverso la macro $_HOSTMACADDRESS$.

#### 2.1.1.3 – Host e Service check

Gli host vengono controllati dal demone Nagios:

- ad intervalli di tempo regolari, come riportato nel campo *check_interval* e *retry_interval* nella configurazione degli host;
- on-demand quando il servizio associato all'host cambia stato;
- on-demand, se necessario, come parte della logica di raggiungibilità dell'host;
- on-demand, se necessario, per controlli predittivi sulla dipendenza dell'host.

I service, invece, vengono controllati:

- ad intervalli di tempo regolari, come riportato nel campo *check_interval* e *retry_interval* nella configurazione dei service;
- on-demand, se necessario per controlli predittivi sulla dipendenza del servizio.

La schedulazione regolare dei check fatti sulla macchina sono opzionali, infatti, se si setta il *check_interval* a zero, Nagios non effettuerà controlli sulla macchina in modo regolare. Tuttavia, verranno effettuati controlli on-demand della macchina a causa della logica di monitoraggio. Tali check, infatti, vengono fatti quando il service associato all'host cambia stato, questo perché Nagios necessita di sapere se lo stato

dell'host è cambiato oppure no. In altre parole, se cambia lo stato del servizio associato alla macchina, Nagios automaticamente effettua un controllo anche dell'host.

Nagios è stato progettato per individuare interruzioni di rete il più velocemente possibile e distinguere lo stato DOWN dallo stato UNREACHABLE dell'host. Anche se potrebbero sembrare simili, questi stati sono molto differenti fra loro e consentono all'utente di individuare, velocemente, la causa dell'interruzione della rete.

La performance della logica di monitoraggio di Nagios può essere ulteriormente migliorata implementando l'utilizzo dei cosiddetti *cached check*. Questi particolari controlli consentono a Nagios di rinunciare ad eseguire un check su servizi o macchine, a patto che tale controllo determina un risultato relativamente recente, perché se così non fosse allora il check deve essere per forza eseguito.

*Cached check* forniranno un incremento di prestazioni solo se vengono utilizzate le dipendenze degli host o dei service.

Altro aspetto importante di Nagios è che i check possono essere eseguiti in parallelo. Quando il sistema di monitoraggio necessita di effettuare un controllo sull'host o sul servizio, esso eseguirà il check senza bloccarsi, ma ritornerà per eseguire altro (ad esempio un check di un service). Questo avviene perché il demone Nagios, all'esecuzione di un qualsiasi check, genera un processo figlio, attraverso la fork(), al quale attribuisce il controllo correntemente eseguito. Quando l'operazione viene completata, il processo figlio comunicherà il risultato al demone Nagios (processo padre), il quale lo mostrerà all'utente e deciderà quale azione eseguire.

Ogni host può assumere tre stati differenti:

- UP;
- DOWN;
- UNREACHABLE;

mentre per il service si ha:

- OK;
- WARNING;
- UNKNOWN;
- CRITICAL.

I controlli fatti sugli host e su service vengono eseguiti dai plugin, i quali restituiscono, come risultato, uno di questi valori: OK, WARNING, UNKNOWN o CRITICAL.

Per i service non c'è alcun tipo di problema, nel senso che ciò che viene restituito dal plugin, verrà utilizzato immediatamente per indicare lo stato del servizio; ad esempio se l'output del plugin sarà WARNING, allora lo stato del servizio sarà WARNING.

Per gli host la situazione è differente. A questo punto la domanda sorge spontanea: come si fa a collegare l'output di un plugin con lo stato che un host può assumere?

La risposta è semplice e la si può capire dalla seguente tabella:

| Risultato Plugin | Stato preliminare dell'host |
|---|---|
| OK | UP |
| WARNING | UP o DOWN |
| UNKNOWN | DOWN |
| CRITICAL | DOWN |

Come si può notare, in seguito al risultato ottenuto dal plugin, verrà effettuato una seconda operazione che consente di assegnare all'output il corrispondente stato che deve essere attribuito all'host.

Dalla tabella, inoltre, si evince che il risultato di WARNING può rappresentare sia lo stato di UP che quello di DOWN. Il primo caso è quello che si verifica di solito, ma nel caso in cui viene abilitato lo *use_aggressive_host_checking* (controllo della macchina più approfondito), allora c'è la possibilità che si verifica il secondo caso.

Nagios, quando lo stato preliminare della macchina risulta essere DOWN, effettua un ulteriore controllo per capire se l'host è effettivamente DOWN oppure è UNREACHABLE. La differenza fra i due stati è molto importante perché ciò consente di individuare la causa dell'interruzione della rete in modo molto più veloce. Il controllo che permette al sistema di scegliere lo stato finale è il seguente:

| Stato preliminare dell'host | Stato dell'host parent | Stato finale dell'host |
|---|---|---|
| DOWN | Se lo stato di almeno un parent è UP | DOWN |
| DOWN | Se lo stato di tutti i parents è DOWN oppure UNREACHABLE | UNREACHABLE |

Se lo stato degli host o dei service tende a cambiare molto spesso, in questo caso si dice che la macchina o il servizio è in "flapping". Un buon esempio di questo potrebbe essere rappresentato da un server che continua spontaneamente a riavviarsi non appena viene caricato il sistema operativo. In questa circostanza, Nagios si rende conto che la macchina è in flapping, di conseguenza sospende l'invio delle notifiche finché tale problema non viene risolto e lo stato viene re-stabilizzato.

### 2.1.1.4 – Active e Passive check

Nagios ha la capacità di monitorare gli host e i service in due modi: attivamente e passivamente. In tal caso, si parla di check attivi e check passivi.

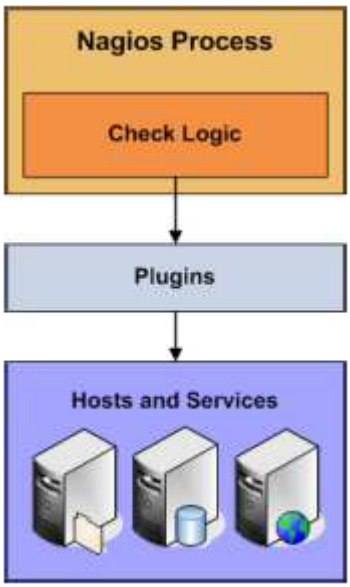



*Figura 4: Active Checks*

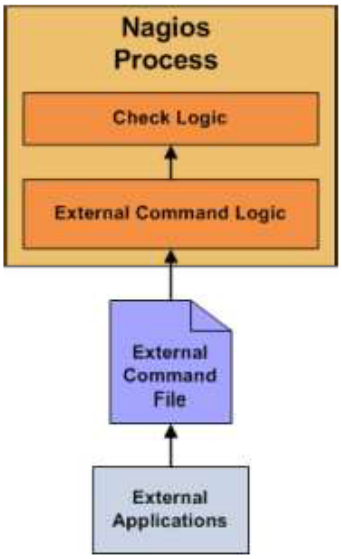



*Figura 5: Passive Checks*

I primi sono i metodi più comuni e più utilizzati per il monitoraggio. Le caratteristiche principali dei check attivi sono:

- vengono inizializzati dal demone Nagios;
- vengono eseguiti regolarmente.

Quando Nagios necessita di controllare lo stato di una macchina o di un servizio, esegue un plugin passandogli le informazioni relative a ciò che desidera controllare. Il plugin, successivamente, controllerà lo stato e restituirà il risultato al demone Nagios, il quale, infine, processerà le informazioni ricevute ed si comporterà di conseguenza, mostrando il risultato, inviando notifiche, ecc.…

I check attivi vengono eseguiti:

- ad intervalli di tempo regolari, definiti da *check_interval* e *retry_interval* definiti nella configurazione dell'host o del service;
- on demand, quando necessario.

L'esecuzione di tali check dipende dal tipo di stato che caratterizza un host o un service. Se lo stato risulta essere HARD, allora il check verrà eseguito ad intervalli di tempo specificati dal *check_interval*; se lo stato risulta essere SOFT, allora il check verrà eseguito ad intervalli di tempo specificati dal *retry_interval*.

Come detto in precedenza, Nagios solitamente si basa su check attivi per il monitoraggio di ciò che è di interesse, in quanto tali check possono essere utilizzati per fare polling su una macchina o servizio per ottenere informazioni di stato. Sappiamo, però, che esiste anche la possibilità di operare in modo passivo attraverso check passivi, le cui caratteristiche principali sono:

- inizializzazione ed esecuzione da parte di applicazioni/processi esterni;
- i risultati dei controlli vengono presentati a Nagios, affinché li possa processare.

I check passivi risultano essere molto utili ed importanti per monitorare quei servizi che sono asincroni in natura e non possono essere controllati attraverso polling per monitorare il loro stato e per quei servizi che sono localizzati dietro un firewall e non possono essere controllati in modo attivo dal sistema di monitoraggio. Esempi di servizi asincroni che si prestano ad essere monitorati attraverso controlli passivi sono SNMP trap e avvisi di sicurezza; infatti, non si saprà mai quanti trap o avvisi verranno ricevuti in un dato intervallo di tempo, quindi non risulta essere flessibile monitorarli ogni tot minuti.

Vediamo adesso il modo in cui lavorano i check passivi.

Un'applicazione esterna controlla lo stato di una macchina o di un servizio e scrive i risultati all'interno del file di comando esterno (Figura 5). A questo punto, Nagios legge tale file ed inserisce i risultati dei controlli passivi all'interno di una coda che successivamente verrà processata. Tale coda viene utilizzata non solo per memorizzare i risultati di check passivi, ma anche quelli dei check attivi. Periodicamente la coda viene scansionata e i risultati vengono processati da Nagios. Ogni risultato trovato viene processato allo stesso modo, indipendentemente che si tratti di un check attivo o passivo. Fatto questo, il sistema provvederà ad eseguire delle azioni in base alle necessità.

Il processo di esecuzione di check attivi e passivi è essenzialmente lo stesso. Questo consente una perfetta integrazione di informazioni sullo stato di applicazioni esterne con Nagios.

#### 2.1.1.5 – State type

Lo stato corrente di un host o servizio monitorato è costituito da due parti:

- lo stato vero e proprio (OK, WARNING, UP, DOWN, ecc…);
- il tipo di stato (SOFT, HARD).

I tipi di stato rappresentano una parte cruciale della logica di monitoraggio perché permettono di capire quando gestori di eventi devono essere eseguiti e quando le notifiche devono essere inviate.

Lo stato di tipo SOFT si verifica:

- quando un service/host check comporta uno stato non-OK o non-UP e il check non è stato eseguito il numero di volte specificato dal *max_check_attempts* presente nella configurazione del servizio o della macchina. Questo prende il nome di soft error;
- quando un servizio o un host ristabilisce il suo stato dal soft error. In questo caso si parla di soft recovery.

La sola cosa importante che accade durante uno stato SOFT è l'esecuzione di gestori di eventi. L'utilizzo di questi può essere particolarmente utile per cercare e risolvere, in modo pro-attivo, il problema prima che si passi allo stato HARD.

Le macro $HOSTSTATETYPE$ o $SERVICESTATETYPE$ avranno il valore "SOFT" quando gestori di eventi sono in esecuzione e consentiranno al gestore stesso di sapere quando eseguire l'azione correttiva.

Lo stato di tipo HARD si verifica:

- quando un service/host check comporta uno stato non-OK o non-UP e il check è stato eseguito il numero di volte specificato dal *max_check_attempts* presente nella configurazione del servizio o della macchina. Questo prende il nome di hard error;
- quando un host o un service passa dallo stato hard error ad un altro stato di errore (ad esempio dal WARNING al CRITICAL);

- quando un service check comporta uno stato non-OK e il suo corrispondente host presenta uno stato DOWN o UNREACHABLE;
- quando un servizio o un host ristabilisce il suo stato dall'hard error. In questo caso si parla di hard recovery;
- quando viene ricevuto un host check passivo.

A differenza di uno stato SOFT, quando si verifica uno stato HARD quello che succede è che tale situazione viene riportata in un file di log, vengono eseguiti i gestori di eventi per poter gestire la situazione ed, infine, vengono inviate notifiche ai contatti di interesse per informarli del problema o del recovery avvenuto su un servizio o su un determinato host.

Anche in questo caso, le macro $HOSTSTATETYPE$ o $SERVICESTATETYPE$ avranno il valore "HARD" quando gestori di eventi sono in esecuzione e consentiranno al gestore stesso di sapere quando eseguire l'azione correttiva.

#### 2.1.1.6 – Time Period

I time period consentono di controllare quando i vari aspetti del monitoraggio e la logica di allerta possono operare. Ad esempio, si può definire:

- quando eseguire host/service check regolarmente schedulati;
- quando inviare le notifiche;
- quando utilizzare le escalation di notifiche;
- quando le dipendenze sono valide.

I time period possono contenere diverse direttive, come giorni della settimana, giorni del mese e date di calendario. Ciascuna di queste presenta un livello di precedenza che può consentire ad una direttiva di sovrascrivere l'altra. Vediamo come queste sono posizionate in ordine decrescente (tra parentesi un esempio):

1. data del calendario (01-01-2008);
2. specifica data del mese (1 Gennaio);
3. generica data del mese (Giorno 15);
4. giorno offset della settimana di un mese specifico (secondo Martedì di Dicembre);
5. giorno offset della settimana (terzo Lunedì);

6. normale giorno della settimana (Venerdì).

Un esempio di time period potrebbe essere:

**define timeperiod{**

**timeperiod_name nonworkhours**

**alias Non-Work Hours**

**sunday 00:00-24:00;**

**monday 00:00-09:00, 17:00-24:00;**

**tuesday 00:00-09:00, 17:00-24:00;**

**wednesday 00:00-09:00, 17:00-24:00;**

**thursday 00:00-09:00, 17:00-24:00;**

**friday 00:00-09:00, 17:00-24:00;**

**saturday 00:00-24:00;**

**}**

In questo esempio, i controlli verranno effettuati tutti i giorni della settimana, ma dal lunedì al venerdì dalle 00:00 alle 09:00 e dalle 17:00 alle 24:00, mentre il sabato e la domenica dalle 00:00 alle 24:00.

Le definizioni di host e di service hanno un campo opzionale, chiamato *check_period*, che consente di specificare il periodo in cui effettuare i check attivi, limitando così la loro regolare schedulazione. Ad esempio, se lo scheduling regolare prevede di effettuare un check ogni 10 minuti, impostando un periodo, tipo ogni primo Martedì del mese, allora solo in quel determinato giorno verranno effettuati controlli secondo lo scheduling previsto. Se il *check_period* non viene considerato, allora Nagios effettuerà i check ogni volta che lo riterrà necessario. Tipicamente, questa situazione rispecchia uno scenario di monitoraggio 24x7, ovvero, i controllo vengono fatti tutti i giorni, ventiquattro ore su ventiquattro.

Quando Nagios tenta di rischedulare un controllo, si assicura che il prossimo check ricada nell'intervallo specificato in *check_period*. Se così non fosse, il sistema di monitoraggio "aggiusterà" il prossimo tempo di check in modo che questo coincida con il prossimo tempo di check valido all'interno del time period specificato.

Un aspetto importante è rappresentato dal fatto che il time period può limitare la schedulazione regolare solo dei check attivi e non quella dei check passivi o di quelli on-demand.

A meno che non ci sia una buona ragione per non farlo, si raccomanda di monitorare gli host e/o i service tutti i giorni ventiquattro ore su ventiquattro; questo per evitare di avere problemi durante i blackout, cioè quei tempi che non sono validi nella definizione del time period. Alcune problematiche potrebbero essere:

- lo stato della macchina o del servizio potrebbe apparire immutato durante il blackout;
- i contatti potrebbero non essere rinotificati dei problemi verificatosi durante il blackout;
- se un host e/o un service recupera lo stato durante il blackout, i contatti potrebbero non essere immediatamente notificati della recovery avvenuta.

Una caratteristica importante dei time period è che possono essere utilizzati sia per la notificazione dei contatti, ma anche per la definizione delle escalation.

Per quanto riguarda le notifiche, nella definizione di host o service, è possibile settare il campo *notification_period* che consente a Nagios di notificare i contatti, per eventuali problemi o per recovery avvenute, limitando l'intervallo di tempo in cui questa operazione deve essere effettuata; infatti, quando deve essere inviata una notifica, Nagios si assicurerà che il tempo corrente faccia parte dell'intervallo di tempo definito in *notification_period*.

Per quanto riguarda le escalation, invece, nella definizione di host e/o di service bisogna settare il campo *escalation_period*, per specificare l'intervallo di tempo in cui considerare l'escalation valida e poterla utilizzare. Nel caso in cui il periodo non viene definito, allora Nagios considererà l'escalation sempre valida.

#### 2.1.1.7 – Notification

La decisione di inviare le notifiche viene presa nella logica dei service/host check. Le notifiche si presentano:

- quando si verifica un cambiamento dello stato HARD;
- quando un host o service rimane in uno stato HARD non-OK e il tempo specificato nel campo *notification_interval* è stato superato rispetto all'invio dell'ultima notifica.

Per quanto riguarda i destinatari delle notifiche, ogni definizione di host o service presenta un campo, chiamato *contact_groups*, che specifica il gruppo di contatti a cui devono essere inviate le notifiche. Tale gruppo può essere costituito da uno o più contatti.

Quando Nagios deve inviare delle notifiche relative ad un host o un service, le manda a tutti i componenti del gruppo specificato in *contact_groups*. Inoltre, poiché un contatto può far parte di più gruppi, Nagios, prima di fare qualsiasi cosa, elimina le notifiche di contatto duplicate.

È chiaro che solo perché c'è la necessità di inviare notifiche questo non significa che ogni contatto debba essere notificato. Esistono, infatti, molti filtri che potenziali notifiche devono superare per essere considerate valide e per poter essere inviate. Di conseguenza, specifici contatti potrebbero non essere notificati se i loro filtri non consentono la ricezione della notifica.

Andiamo ad analizzare, adesso, i diversi filtri che una notifica dovrà affrontare per poter essere inviata e raggiungere, così, la destinazione.

La prima cosa da verificare è vedere se il sistema di monitoraggio ha l'opzione, relativa alle notifiche, abilitata. Questo rappresenta il primo grande ostacolo che bisogna superare, in quanto se le notifiche non sono abilitate, queste non verranno mai inviate. L'abilitazione può essere fatta attraverso il campo *enable_notification* presente nel file di configurazione, oppure durante l'esecuzione attraverso l'interfaccia web. Se le notifiche sono abilitate, andiamo a vedere quali sono gli altri filtri da superare per far si che la notifica giunga a destinazione.

Il primo filtro è un controllo per vedere se l'host o il service si trova in un periodo di downtime schedulato. Se così fosse, nessuna notifica viene inviata, in caso contrario si passa al filtro successivo. In questo caso, si a verificare se il servizio o la macchina si trovano in "flapping" (sempre se la flapping detection risulta essere abilitata). Se ci si trova in questa situazione, le notifiche non vengono inviate, in caso contrario si prosegue nel superamento dei filtri che si incontrano. In questa circostanza, invece, il filtro è dato dalle impostazioni di notifica presenti nella definizione di host o di service. In base a questi, si stabilisce se la notifica deve proseguire oppure non deve essere inoltrata. Se le opzioni consentono il passaggio, si passa al filtro successivo, che consiste nel rispettare l'intervallo di tempo indicato nel campo *notification_interval*; infatti, se la notifica viene inviata in un tempo al di fuori del

range specificato, allora questa non verrà portata avanti, in caso contrario possiamo considerare terminati i filtri relativi all'host e/o al service.

Arrivati a questo punto, è chiaro che ci saranno filtri relativi ai contatti che potranno e vorranno ricevere le notifiche. Il primo filtro è rappresentato dalle opzioni di notifica presenti nella definizione di un contatto. In base a tali criteri, si stabilisce se una notifica deve essere consegnata oppure no. Se si supera questo tipo di filtro, si passa all'ultimo ostacolo, rappresentato dal campo *notification_period* che specifica l'intervallo di tempo considerato valido per la ricezione delle notifiche. Se tale periodo viene rispettato, allora la notifica, finalmente, viene consegnata a destinazione e ricevuta dal contatto, in caso contrario, non viene inviata.

### 2.1.2 – Requisiti di sistema ed installazione

Gli unici requisiti che vengono richiesti affinché sia possibile installare ed eseguire Nagios sono:

- utilizzo di una macchina Linux (o variante di UNIX);
- installazione del compilatore C.

Inoltre, poiché la maggior parte di service check vengono eseguiti sulla rete, sarebbe opportuno configurare TCP/IP, ma solitamente questo risulta essere già configurato.

Per quanto riguarda l'installazione di Nagios, questo viene fatto attraverso source code. La prima cosa da fare è soddisfare i prerequisiti, ovvero, installarsi ciò che è richiesto per l'utilizzo di Nagios. Tali elementi sono:

- Apache: può essere installato con *yum install httpd*;
- PHP: può essere installato con *yum install php*;
- GCC compiler: può essere installato con *yum install gcc glibc glibc-common*;
- librerie di sviluppo GD: può essere installato con *yum install gd gd-devel*.

Per poter eseguire questi comandi, è necessario avere l'accesso di *root* alla macchina.

Una volta installati i prerequisiti, bisogna creare un utente nagios ed attribuirgli una password:

- */usr/sbin/useradd -m nagios*: per la creazione dell'utente;
- *passwd nagios*: per la creazione della password da attribuire all'utente nagios.

A questo punto, bisogna crearsi un nuovo gruppo per consentire che comandi esterni possano essere presentati attraverso l'interfaccia web; successivamente bisogna aggiungere al gruppo creato sia l'utente nagios che l'utente apache:

- */usr/sbin/groupadd nagcmd*: per la creazione di un gruppo chiamato nagcmd;
- */usr/sbin/usermod -a -G nagcmd nagios*: per aggiungere l'utente nagios al gruppo nagcmd;
- */usr/sbin/usermod -a -G nagcmd apache*: per aggiungere l'utente apache al gruppo nagcmd.

Fatto questo, finalmente possiamo scaricarci Nagios e i corrispondenti plugin.

Per prima cosa, creiamo una directory:

- *mkdir ~/downloads*: per la creazione della directory;

successivamente, entriamo nella directory creata e scarichiamo il nostro sistema di monitoraggio con i plugin in formato .tar.gz:

- *cd ~/downloads*: per entrare nella directory;
- *wget http://prdownloads.sourceforge.net/sourceforge/nagios/nagios-3.5.1.tar.gz*: per scaricare Nagios;
- *wget http://prdownloads.sourceforge.net/sourceforge/nagiosplug/nagios-plugins-1.4.11.tar.gz*: per scaricare i plugins di Nagios.

Scaricati tali pacchetti, questi devono essere compilati ed installati. Partiamo con Nagios:

- *tar xzf nagios-3.5.1.tar.gz*: per l'estrazione dei file contenuti nel pacchetto nagios-3.5.1.tar.gz in una semplice directory nagios-3.5.1;
- *cd  nagios-3.5.1*: entriamo nella directory;
- *./configure –with-command-group=nagcmd*: eseguiamo lo script di configurazione di Nagios con il nome del gruppo creato;
- *make all*: compiliamo il codice sorgente di Nagios;

ed, infine, installiamo tutto il resto:

- *make install*: per l'installazione dei file binari;
- *make install-init*: per l'installazione degli init-script;

- *make install-config*: per l'installazione dei file di configurazione;
- *make install-commandmode*: per l'installazione e settaggio dei permessi sulla directory dei comandi esterni.

Arrivati a questo punto, bisogna andare ad installare i file di configurazione dell'interfaccia web di Nagios nella directory *conf.d* di Apache:

- *make install-webconf*: per l'installazione di file di configurazione dell'interfaccia web.

Bisogna crearsi un account *nagiosadmin* per poter accedere all'interfaccia web con con corrispondente password:

- *htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin*: per la creazione della password relativa all'admin che permetterà di accedere all'interfaccia web.

Per rendere effettive le modifiche, riavviamo Apache:

- *service httpd restart*: per riavviare il server Apache.

Una volta fatto tutto questo, non rimane altro che installare i plugin di Nagios e avviare il nostro sistema di monitoraggio.

Installiamo i plugin:

- *tar xzf nagios-plugins-1.4.11.tar.gz*: per l'estrazione dei file contenuti nel pacchetto nagios-plugins-1.4.11.tar.gz in una semplice directory nagios-plugins-1.4.11;
- *cd  nagios-plugins-1.4.11*: entriamo nella directory;
- *./configure --with-nagios-user=nagios --with-nagios-group=nagios* : eseguiamo lo script di configurazione dei plugins;
- *make && make install*: per l'installazione dei plugins.

Abbiamo installato tutto quello che serve per l'utilizzo di Nagios, non è rimasto altro che verificare il suo funzionamento. Per prima cosa aggiungiamo Nagios nella lista dei servizi che vengono avviati automaticamente al boot del sistema:

- *chkconfig --add nagios*: per aggiungere Nagios ai servizi di avvio al boot;
- *chkconfig nagios on*: per abilitare l'avvio di Nagios.

Verifichiamo che i file di configurazione non presentano degli errori:

- */usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg*

se il risultato non presenta errori, possiamo avviare finalmente il nostro sistema di monitoraggio:

- *service nagios start*: per l'avvio del server Nagios.

Per accedere all'interfaccia web, bisogna aprire un qualunque browser, inserire il seguente URL:

*http://localhost/nagios*

inserire come username nagiosadmin e come password quella che abbiamo creato quando è stata installata l'interfaccia web.

## 2.2 – Thruk

Thruk è un'interfaccia web di monitoraggio multi-backend che supporta correntemente Naemon, Nagios, Icinga e Shinken utilizzando Livestatus API. Thruk presenta diverse caratteristiche importanti:

- minor utilizzo di CPU;
- mostra dati live in quanto non c'è nessun ritardo fra core e GUI;
- indipendenza da core di monitoraggio, in quanto questo può essere installato su host remoto;
- può essere clusterizzato sugli host;
- facilmente estendibile con plugin.

### 2.2.1 – Come lavora

Thruk è stato scritto in Perl utilizzando PSGI/Plack Framework. PSGI è un'interfaccia fra applicazioni web in Perl e server web, mentre Plack è un modulo Perl che contiene il PSGI middleware, helper e adapter per il server web stesso.

I sistemi di monitoraggio saranno connessi con il modulo Perl Monitoring::Livestatus che consente di accedere, a runtime, ai dati forniti dai sistemi stessi.

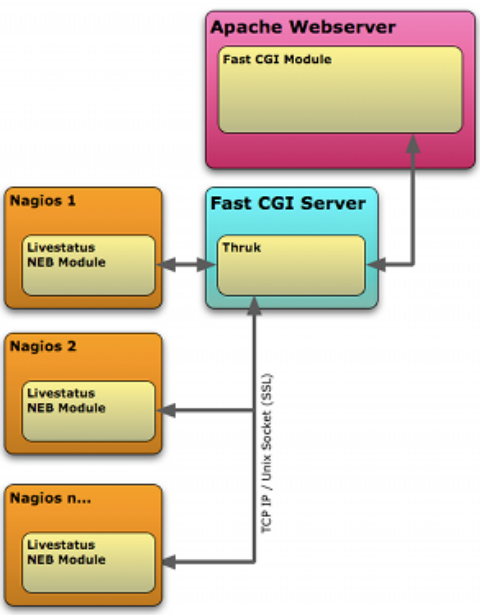



*Figura 6: Architettura Thruk*

Thruk esegue come un processo FastCGI, vale a dire che si basa su un protocollo che permette di interfacciare programmi interattivi CGI con un server web. FastCGI rappresenta una variazione della Common Gateway Interface (CGI), ma con lo scopo

di ottimizzare le risorse di sistema, permettendo al server di gestire più richieste di pagina web assieme.

Thruk utilizza il modulo Perl Monitoring::Availability per il calcolo della disponibilità del sistema a cui è collegato, mentre l'autenticazione viene fornita dal web server Apache.

### 2.2.2 – Perché scegliere Thruk?

Ci sono diverse ragioni che potrebbero portare un utente a scegliere di utilizzare Thruk, motivi che hanno spinto anche l'azienda ad utilizzarlo.

In primo luogo, Thruk prevede il supporto multi-site, cioè ha la possibilità di connettersi a tanti core di monitoraggio quanti se ne vogliono. Ad esempio, si potrebbero combinare le istanza di Nagios, Naemon, Icinga e Shinken in un'unica interfaccia compatta; in questo modo l'utente ha il vantaggio di avere locazioni indipendenti, ma nello stesso tempo monitorare tutte le sue installazioni. Per una migliore sistemazione, c'è anche la possibilità di raggruppare i diversi site in un unico gruppo.

Viene fornita la possibilità di inviare una molteplicità di comandi in una sola volta, infatti, è possibile farlo sia per gli host che per i service. Quando viene effettuato il re-scheduling, Thruk attenderà finché i check, impostati dall'utente, non saranno terminati in modo da poter mostrare i risultati di ritorno il più velocemente possibile.

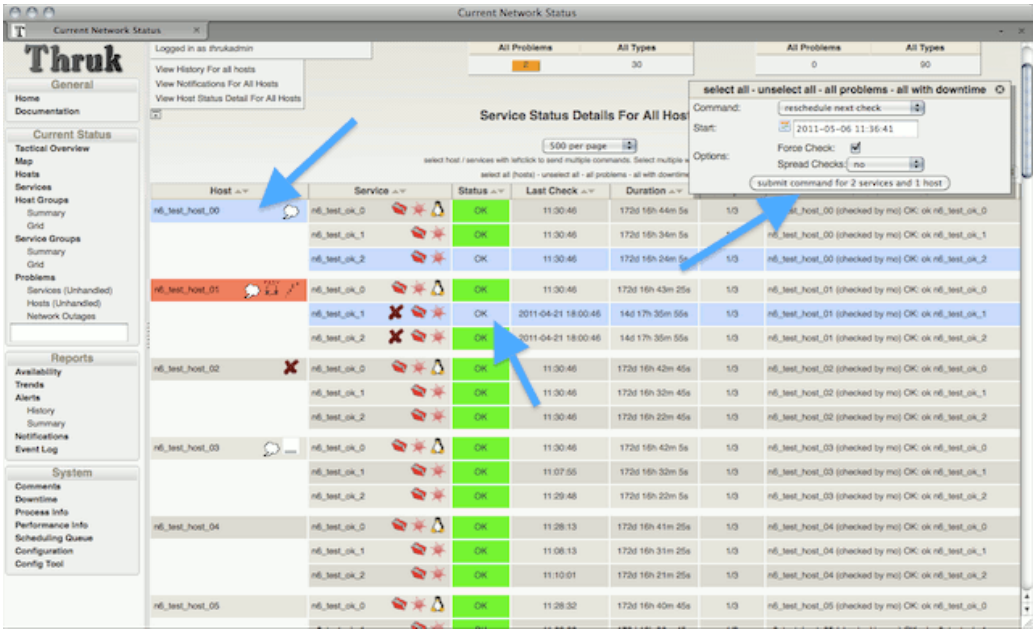



*Figura 7: Esempio di invio di comandi multipli*

Il panorama Thruk fornisce una soluzione dashboard super flessibile. Ogni utente, infatti, può realizzare delle dashboard personalizzate in modo molto semplice e condividerle con i propri colleghi. Più dashboard possono essere collegate fra di loro in modo da poter agire all'unisono; questo consente di creare un proprio centro di controllo.

Thruk presenta dei plugin che permettono di creare completi resoconti sulla SLA (Service Level Agreement) in formato HTML o anche PDF, che possono essere inviati, tramite email, ad intervalli di tempo regolari.

La caratteristica dei bookmark consente all'utente di salvare le sue ricerche e di aggiungerle nel proprio menù personale. Oltre all'utilizzo di bookmark, l'utente può personalizzare il menù, modificando il corrispondente file menu_local.conf.

C'è la possibilità di salvare i dati in un file excel che può essere inviato per email o utilizzato per estrarre le informazioni che sono di interesse all'utente.

Quando la action_url di un host o di un service contiene /pnp4nagios/, automaticamente viene mostrato un grafo a loro corrispondente. Questo consente all'utente di avere una rapida visione delle performance degli host e dei service.


*Figura 8: PNP4Nagios*

Il configuration tool rende molto semplice il cambiamento della configurazione della soluzione di monitoraggio. Consente anche di cambiare velocemente le configurazioni di Thruk, così da poter gestire gli accessi degli utenti attraverso htpasswd o attraverso il file cgi.cfg.

Con Thruk è possibile cambiare rapidamente i filtri di visualizzazione, infatti, c'è la possibilità di combinare una molteplicità di filtri per creare la vista che più piace. Una ricerca in Ajax supporta l'utente, in modo da aiutare gli utenti nella ricerca dei nomi degli host o dei service.

Quando l'output del plugin è costituito da più righe, Thruk mostra un risultato parziale "blu", su cui, cliccando, è possibile ottenere l'output completo. In aggiunta, i commenti e i downtime hanno anche una piccola finestra di popup con i loro dati, così che se l'utente non ha aperto la pagina relativa al service/host, può vedere lo stesso chi ha modificato il commento e quando c'è stato un downtime.

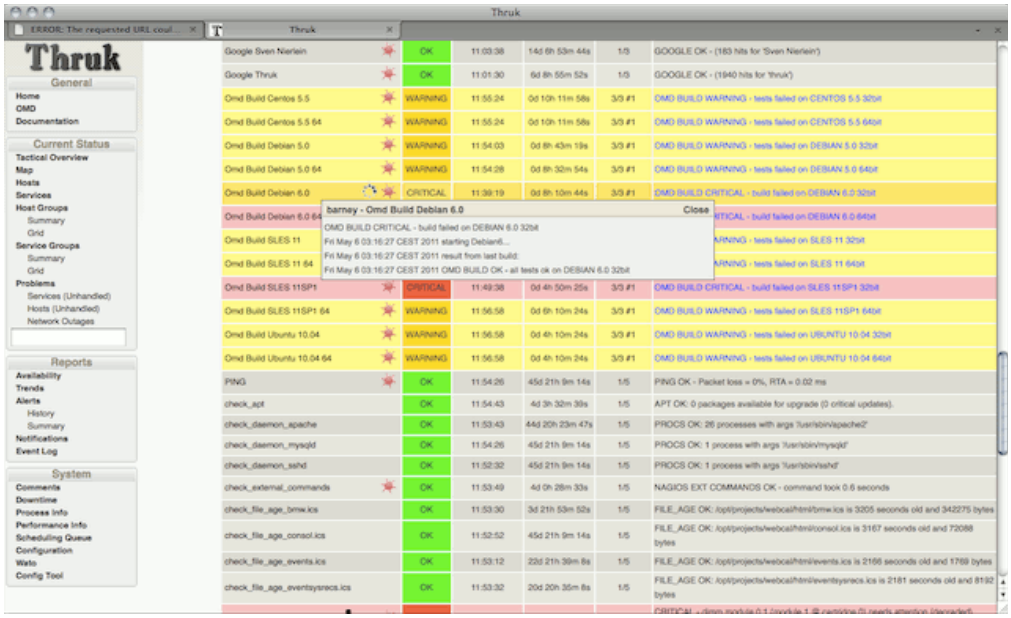



*Figura 9: Output del plugin costituito la linee multiple*

Ultima caratteristica importante di Thruk è la mine map, che rappresenta un tool perfetto per ottenere una rapida panoramica di tutto. È particolarmente utile se si hanno molti service comuni tra gli host. In alternativa, c'è la possibilità di utilizzare hostgroup o servicegroup per ottenere buoni risultati in termini di panoramica.

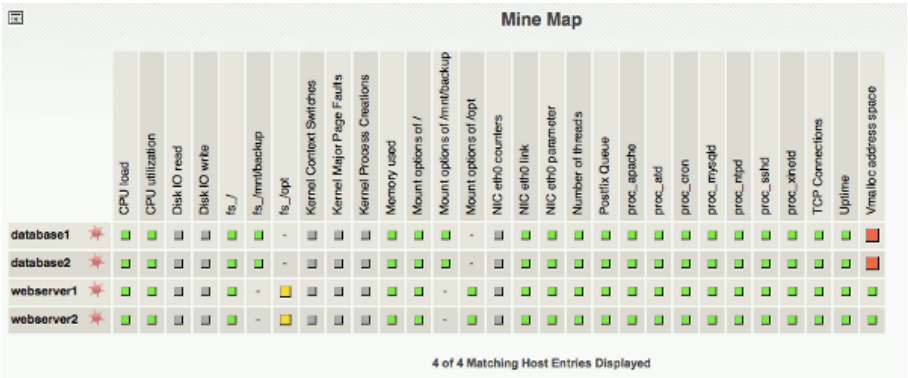



*Figura 10: Esempio di Mine Map*

## 2.3 – Monarch

Groundwork Monitor Architect, comunemente conosciuto come Monarch, è un'applicazione Open Source che si basa su Nagios e consente di effettuare, in modo semplice e veloce, operazioni di configurazione e mantenimento di un sistema IT.

Monarch ha un grandissimo vantaggio, ovvero permette all'utente di configurare il sistema di monitoraggio attraverso una semplice interfaccia web-based piuttosto che andare a modificare manualmente ciascun file di configurazione. Tale modo di lavorare rende la configurazione molto rapida e di facile comprensione, così che chiunque possa cimentarsi in questa operazione senza alcun tipo di difficoltà.

Monarch, quindi, è un sistema web-based completo e di facile utilizzo che viene associato principalmente a Nagios. Tale sistema consiste in un insieme di tool che consentono, ad un utente, di effettuare la configurazione e il mantenimento di Nagios; di conseguenza, Monarch scrive e legge i file di configurazione di Nagios per abilitarli ed integrarli facilmente in un'installazione esistente.

Fatta l'installazione di Monarch, possiamo affermare che tale sistema semplifica il mantenimento del sistema di monitoraggio quando vengono fatte aggiunte, cambiamenti ed eliminazioni sull'infrastruttura IT.

### 2.3.1 – Vista architetturale

Scendendo nel dettaglio, Monarch è un insieme di file/script .cgi scritti in Perl.

Nel momento in cui viene fatta l'installazione di Monarch, viene eseguita un'operazione di caricamento con cui tutti i file di configurazione di Nagios vengono inseriti all'interno del database MySQL di Monarch, rispecchiando al suo interno l'intera configurazione. A questo punto, Monarch non farà altro che manipolare le entry all'interno del database.

In aggiunta alla configurazione di Nagios, in MySQL, Monarch può inserire informazioni aggiuntive come host, service, contact, template, escalation tree e molto altro. In altre parole, una volta installato Monarch e aver caricato i file di configurazione di Nagios all'interno del database, per realizzare tutto ciò che serve non bisogna fare altro che accedere all'interfaccia web di Monarch e crearli.

Una volta effettuate le modifiche, bisogna eseguire il Pre Flight Test, per controllare se queste risultano essere fatte in modo corretto. In caso di esito negativo, vengono riportati, nella schermata, errori che sono stati riscontrati nella configurazione e che devono essere corretti affinché si possa proseguire. In caso di esito positivo, tale

processo effettuerà la scrittura dell'aggiornamento dei file di configurazione di Nagios all'interno del Workspace Directory. È chiaro che l'aggiornamento vero e proprio non è stato ancora effettuato, infatti, Pre Flight Test è solo un modo di controllare che le modifiche siano state effettuate in modo corretto e fornisce l'opportunità di visualizzare e manipolare manualmente i file di configurazione.

Arrivati a questo punto, non rimane altro che eseguire un'altra operazione che prende il nome di commit, la quale creerà un backup della vecchia configurazione all'interno della backup directory e copierà gli aggiornamenti, scritti durante la Pre Flight Test, nella directory /nagios/etc ed eseguirà un update del database MySQL. La commit, inoltre, comporta anche il riavvio di Nagios, attivando e rendendo effettive le modifiche che sono state fatte.

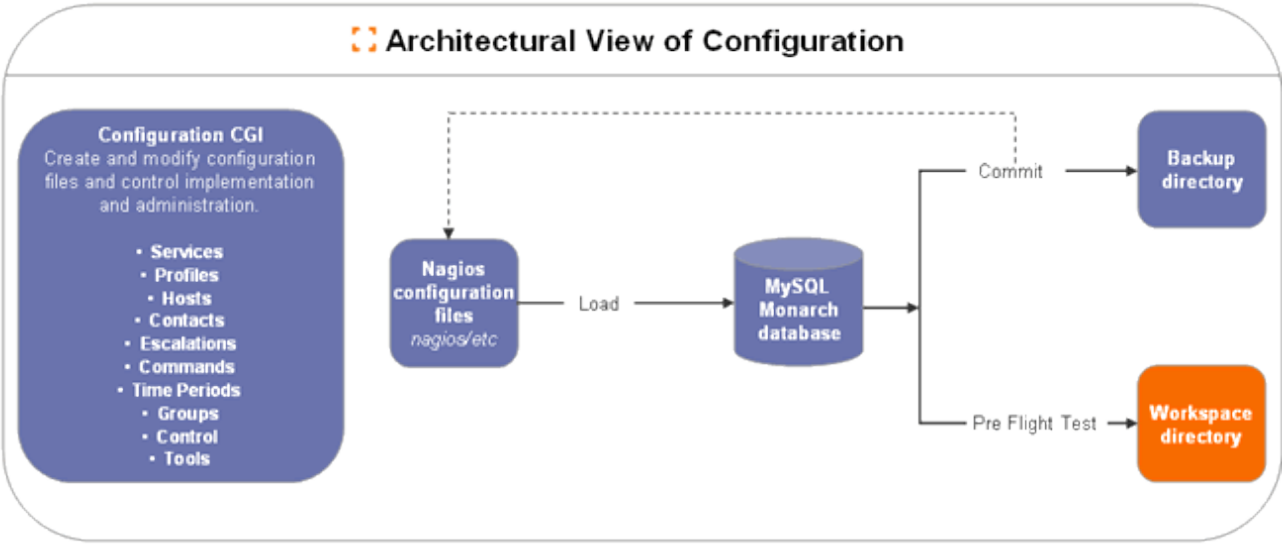



*Figura 11: Vista architetturale*

## 2.3.2 – Concetti chiave

In questo paragrafo andiamo a vedere quali sono i concetti chiave presenti in Monarch e come configurarli. Una precisazione: i concetti spiegati qui di seguito sono considerati importanti dal punto di vista del sottoscritto, in quanto il lavoro svolto durante questo periodo si è incentrato su questi elementi. Il mondo di Monarch risulta essere vastissimo, dal punto di vista della configurazione, pertanto è possibile consultare la documentazione ufficiale su cui ho effettuato i miei studi [2].

### 2.3.2.1 – Host

Un host viene utilizzato per definire server fisici, workstation, device, ecc… localizzati sulla rete e configurare un host consiste nel settare quei parametri che permettono di monitorare la macchina appena creata.

Per poter creare e configurare un host, la prima cosa da fare è entrare nel menù di configurazione di Monarch. A questo punto non rimane altro che andare su *Hosts* e selezionare *Host Wizard*. Fatto questo ci ritroveremo la seguente schermata:

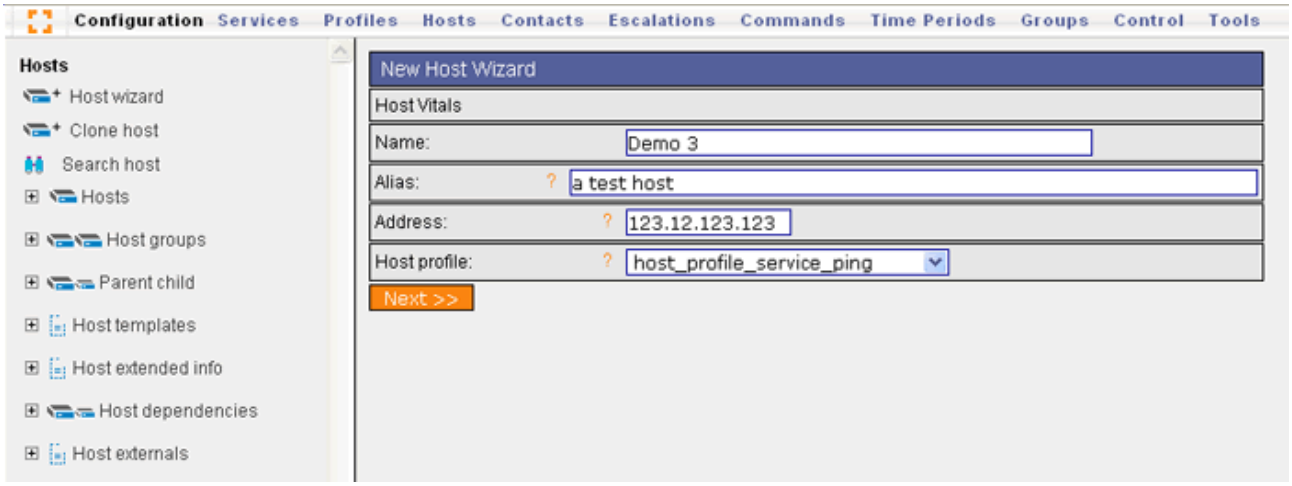



*Figura 12: Creazione di un Host*

In questo caso, andremo ad inserire:

- *nome* della macchina;
- un'*alias* che indica un nome più lungo o una breve descrizione utilizzata per identificare l'host;
- l'*indirizzo IP* della macchina*;*
- un *profilo host* che viene utilizzato per aiutare a progettare e gestire tutti gli host che si vogliono monitorare. Creare un profilo host consente di definire genericamente ruoli differenti per diversi device che si stanno monitorando ed applicarli a loro.

Una volta settati questi parametri, andiamo avanti con il tasto *Next* e ci ritroveremo una schermata in cui è possibile settare le proprietà di livello 1:

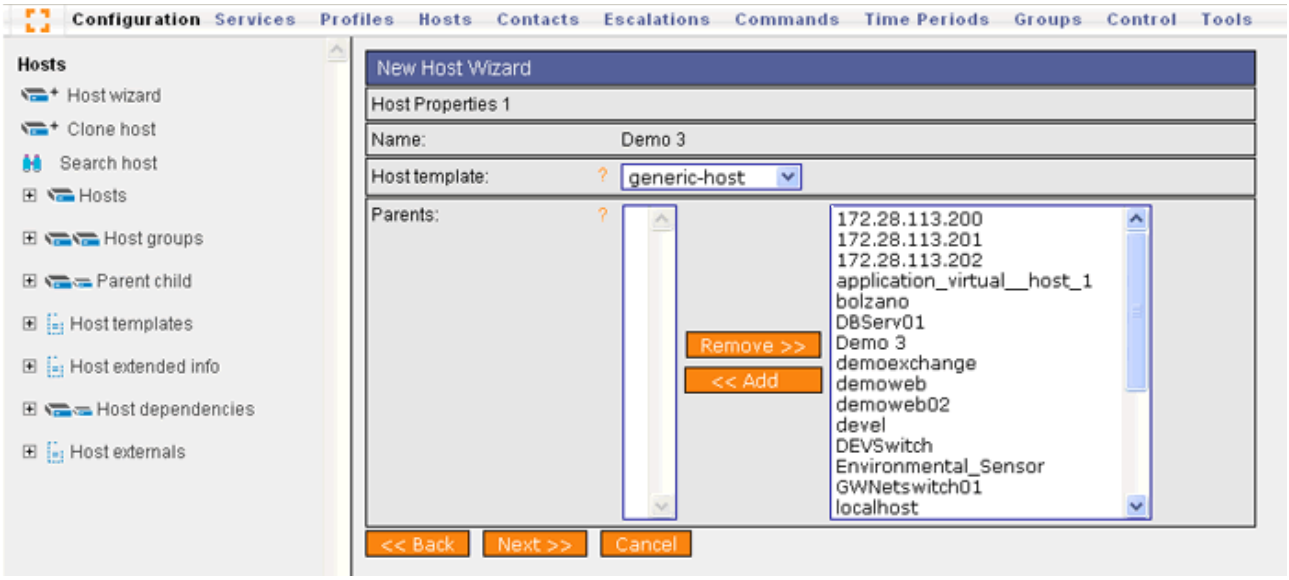



*Figura 13: Proprietà di livello 1*

In questo caso, bisogna inserire:

- un *host template* che deve risultare essere il più adatto alla macchina che stiamo creando. L' host template serve per definire un insieme di proprietà che vanno a caratterizzare uno o più host, in questo modo si riduce il numero di entry ripetute quando si definiscono gli oggetti. Ad esempio, abbiamo N host che hanno caratteristiche uguali; invece di andarli a settare N volte, creiamo un template che presenta ciò che è di interesse e, per ogni macchina che andiamo a creare, impostiamo quel template in modo da settare i parametri automaticamente;
- uno o più *parents*. I parent, invece, tipicamente sono dei router, switch, firewall, ecc… che giacciono fra l'host di monitoraggio e quello remoto. Quello che risulta essere il più vicino all'host remoto viene considerato parent. Se i due host non presentano intermediari, si dice che l'host remoto si trova sulla rete locale di quello di monitoraggio e quindi non avrà parent.

Andando ancora avanti, ci si porta nella schermata che permette di settare le proprietà di livello 2:

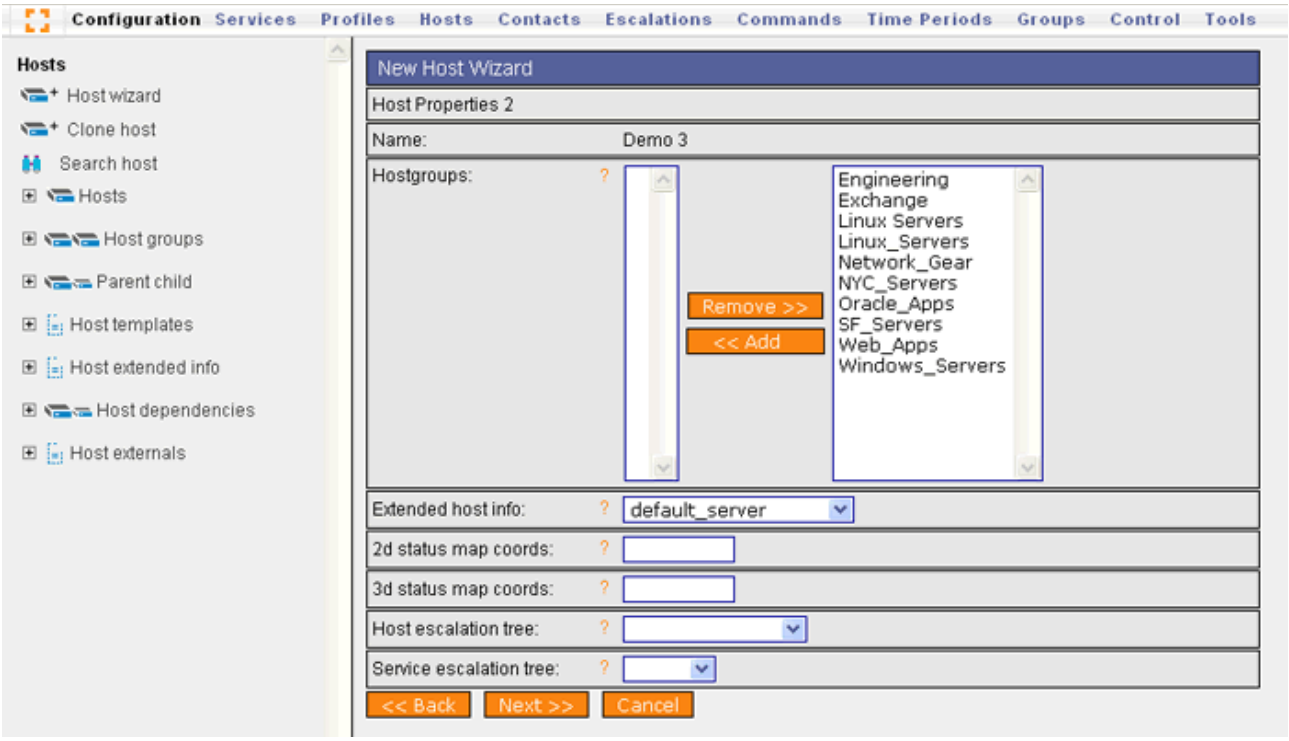



*Figura 14: Proprietà di livello 2*

Dalla Figura 14 possiamo osservare quali sono le proprietà:

- *hostgroup* che consente all'host che stiamo creando di far parte di un particolare gruppo di host;
- *extended host info* che definisce ulteriori informazioni relative all'host come action_url, notes, icone_image, ecc…;
- *2d-3d status map coords* definiscono le coordinate per determinare la posizione dell'host nella mappa di stato 2d-3d di Nagios;
- *host escalation tree* che sono raggruppamenti di host escalation multipli che vengono assegnati all'host;
- *service escalation tree* che seleziona l'appropriato escalation tree per i servizi sull'host.

Le notifiche e le escalation rappresentano il modo con cui Monarch avvisa i suoi utenti che lo stato di una macchina o di un servizio è cambiato (si passa dallo stato OK al WARNING, CRITICAL, UNKNOWN e viceversa). Le escalation vengono combinate con specifici contact group che devono essere notificati quando si presenta una notifica. Quando un host o un service ha un problema o un cambiamento di stato, Nagios invierà le notifiche allo specifico contact group, di conseguenza la notifica sarà ricevuta da tutti i contatti che fanno parte di quel determinato gruppo.

Definite le proprietà di secondo livello, si può passare alla selezione dei servizi che faranno parte dell'host:

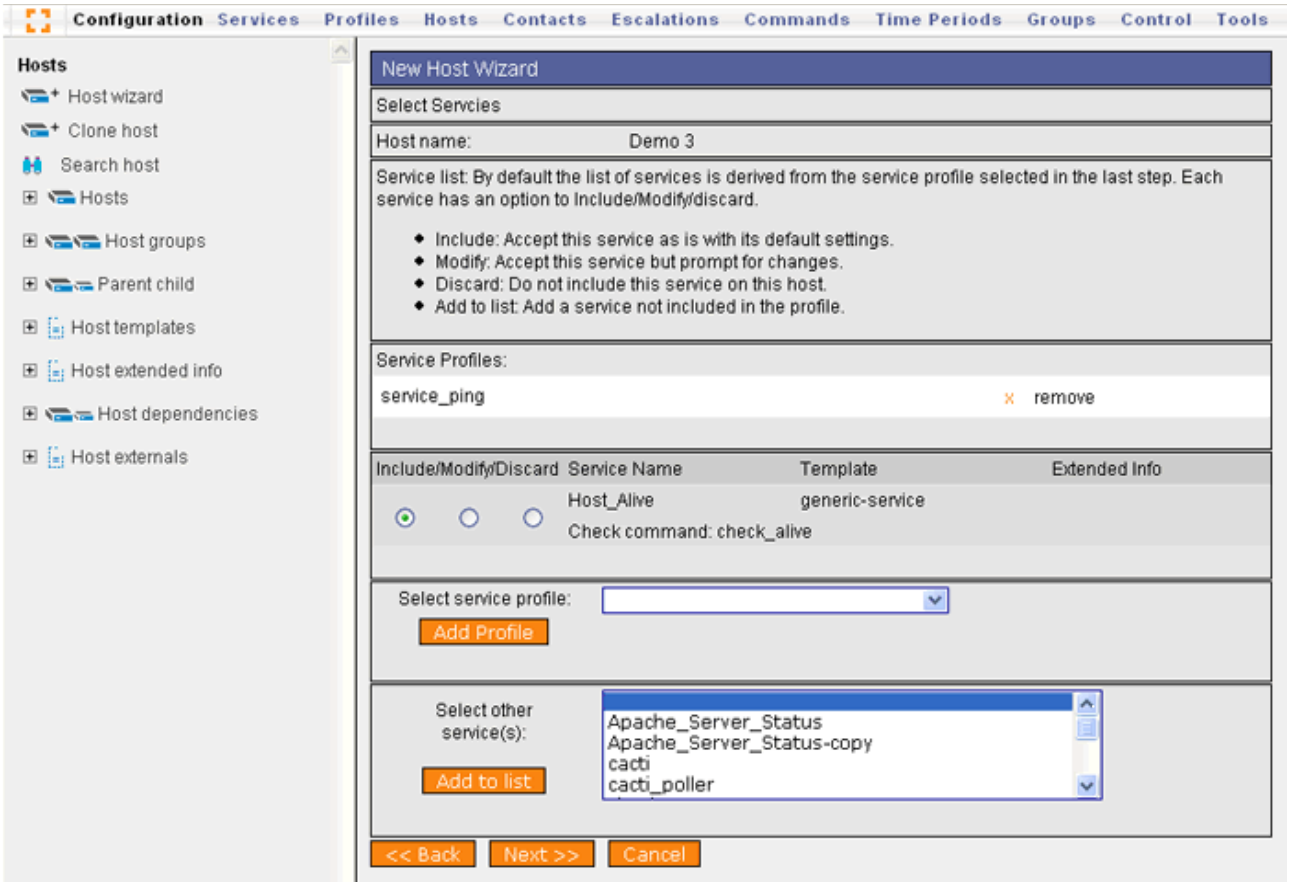



*Figura 15: Selezione dei servizi*

In questo caso, bisogna selezionare:

- il *service_profile* cioè un profilo costituito da una lista di servizi. Un service_profile è molto simile ad un template, nel senso che una volta creato un profilo con una serie di servizi, quando andiamo ad impostare tale profilo alla macchina, questa automaticamente avrà tutti i servizi presenti nel profilo. Ricordiamo che nel template, invece, si avevano non servizi, ma caratteristiche e proprietà;
- *Include/Modify/Discard* consentono all'utente di decidere se accettare i servizi che derivano dal service_profile così come sono (ovvero con i settaggi standard), se modificarli oppure se non includerli;
- *select service profile* permette di aggiungere un profilo ulteriore a quello definito in precedenza;
- *select other services* permette di aggiungere altri servizi che non sono presenti nel service profile.

Cliccando ancora il tasto *Next*, abbiamo terminato la creazione e configurazione dell'host:

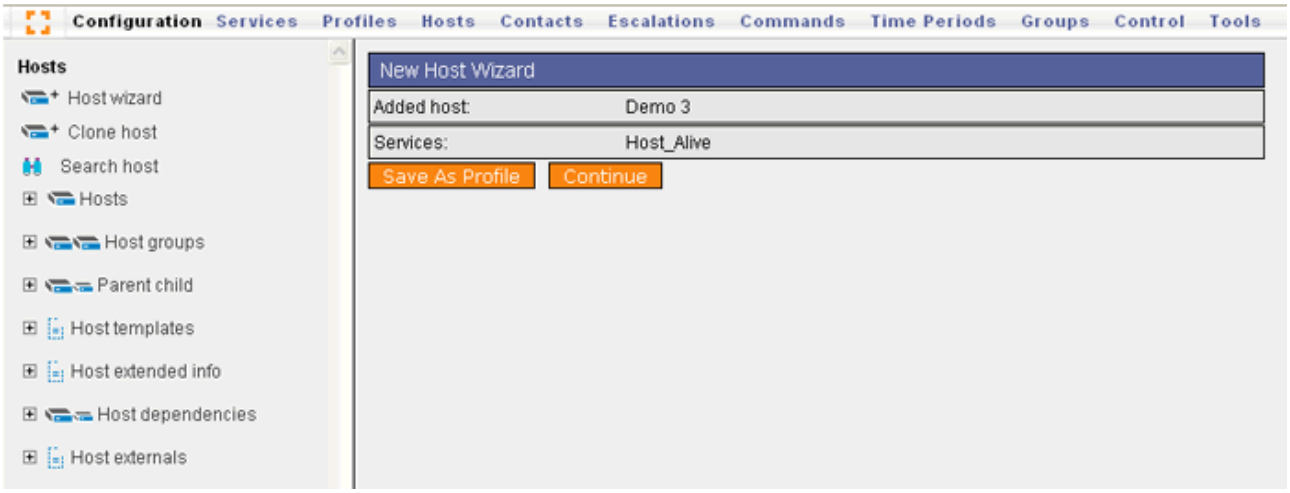



*Figura 16: Terminazione della creazione dell'host*

Una cosa importante: una volta terminata questa procedura, bisogna andare in *Control* (che possiamo vedere in alto a destra nella Figura 16) ed effettuare il Pre Flight Test e successivamente la commit. Senza queste due operazioni l'host non viene creato, in quanto il database non viene aggiornato.

Se tutto va per il verso giusto, nell'interfaccia grafica troveremo il nostro host funzionante.

È chiaro che una volta creato l'host è possibile andare a modificare i dati anche in un secondo momento; infatti, cliccando su *Hosts*, invece di andare su *Host wizard*, bisogna andare su *Hosts* in cui si potrà accedere alle impostazioni di ogni singola macchina e a quelle dei servizi che compongono la macchina stessa. Anche quando vengono effettuate le modifiche bisogna sempre ricordare di effettuare la commit affinché le variazioni possano essere applicate.

Un altro aspetto importante è che se vogliamo creare una macchina simile ad una già esistente, ma con qualche proprietà differente, c'è la possibilità di clonare l'host esistente (in quel caso si cliccherà su *Clone host*) e successivamente accedere ai suoi dettagli per poter effettuare le modifiche desiderate.

### 2.3.2.2 – Service

Un service lo possiamo definire come un servizio che esegue su un determinato host, ad esempio HTTP, SMTP, POP, ecc… oppure come un qualunque tipo di metrica associata alla macchina, ad esempio risposta al ping, numero di utenti loggati, spazio di disco libero, ecc….

Per poter creare e configurare un service, la prima cosa da fare è entrare nel menù di configurazione di Monarch. Fatto questo andiamo a cliccare su *Services* e successivamente su *New Services*:

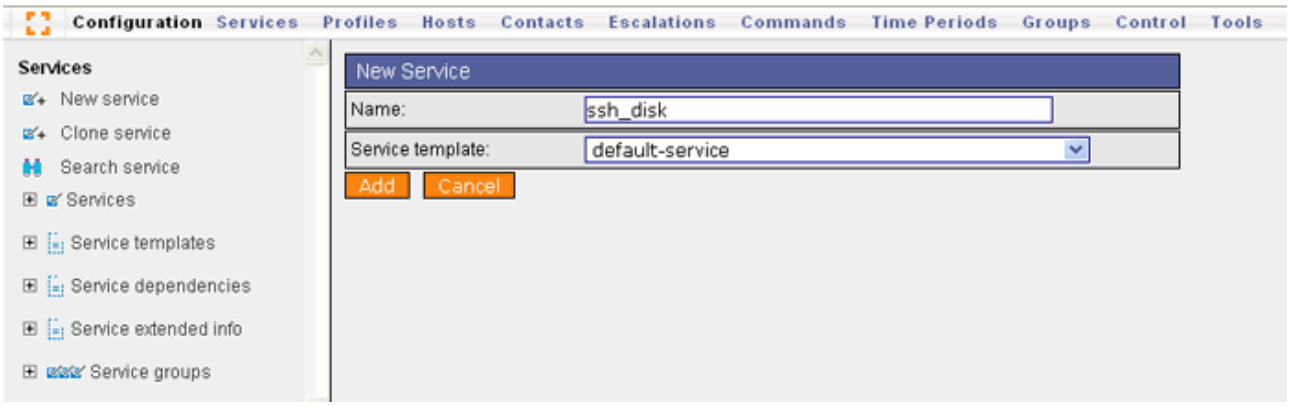



*Figura 17: Creazione di un nuovo service*

In questo caso, come si può osservare dalla Figura 17, bisogna inserire:

- il *nome* del servizio;
- il *template* che si ritiene più opportuno.

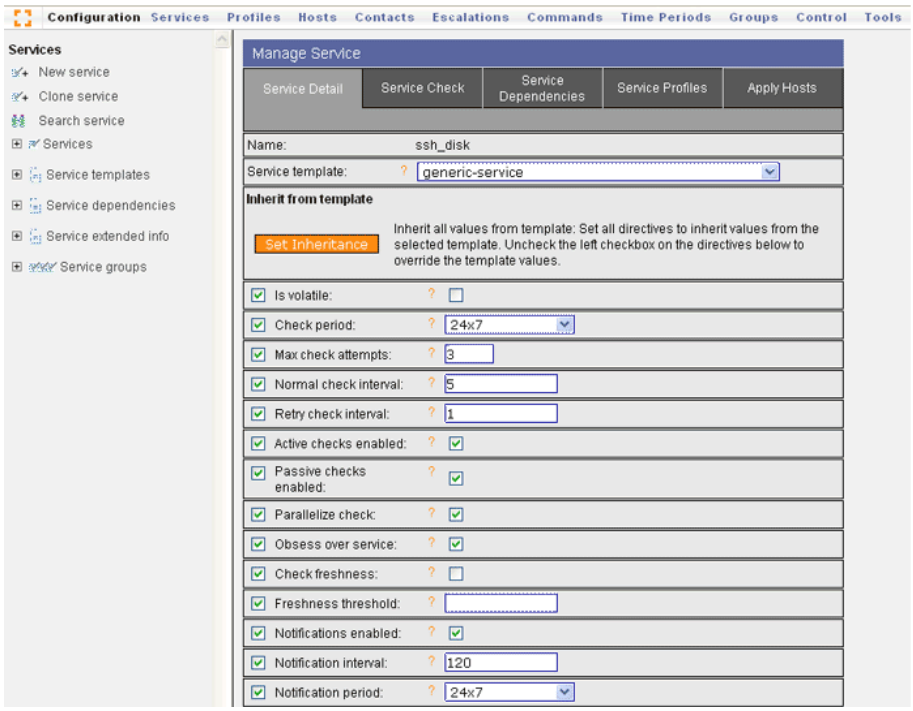



*Figura 18: Service detail parte 1*

Grazie a quest'ultimo, il servizio verrà creato con proprietà che sono state già settate, infatti, cliccando su *Add* ci ritroveremo sulla schermata mostrata nella Figura 18.

Come è possibile notare dalla prima parte dei dettagli del servizio, ci sono delle proprietà che sono già settate, questo grazie alla scelta del service template (in questo caso è stato scelto il generic-service, ma se si fosse scelto un altro template le impostazioni sarebbero cambiate). È chiaro che le opzioni possono cambiare da un template all'altro, ma anche essere cambiate manualmente secondo i desideri dell'utente. Andiamo ad analizzare le opzioni:

- *Set Inheritance*, che, quando schiacciato, setta tutti i valori ereditandoli dal service template passato in precedenza. Per comprendere meglio il suo utilizzo facciamo un esempio: come si può notare dalla Figura 18, tutti i campi sono selezionati. L'utente, se lo desidera, può cancellare la selezione di alcuni e mantenerne altri. Se vuole ripristinare le impostazioni, basta cliccare sul tasto set inheritance che riporterà la selezione di tutte le opzioni. Questo per il semplice motivo che il service template impostato è stato realizzato con tutte le opzioni selezionate, quindi il set inheritance non fa altro che settare le impostazioni ereditate dal template;
- *Is Volatile* che viene utilizzata per denotare se il service è volatile; normalmente non lo sono;
- *Check Period* specifica il nome di un periodo di tempo durante il quale vengono effettuati i check. Nella Figura 18, ad esempio, troviamo 24x7 che sta ad indicare che i check verranno effettuati sette giorni su sette e ventiquattro ore su ventiquattro. Ci sono periodi già definiti che un utente può scegliere, ma c'è anche la possibilità di crearsi dei periodi personalizzati, in cui si vanno a settare il nome per poter identificare il periodo, un alias per dargli una breve descrizione ed, infine, i giorni della settimana (Sunday-Saturday) in cui bisogna specificare l'orario in cui effettuare i check (ad esempio 07:00-17:00 indica che i check verranno fatti dalle sette del mattino alle cinque del pomeriggio, oppure c'è la possibilità di specificare orari differenti per giorni differenti; ancora se in un giorno non inseriamo nessun orario, questo implica che non verranno fatti check in quel determinato giorno, ecc…);
- *Max Check Attempts* definisce il numero di tentativi fatti da Nagios nel effettuare nuovamente il check di un servizio quando il suo stato risulta essere

differente dall'OK. Se si supera tale numero e lo stato del servizio non è diventato OK, allora Nagios inizierà ad inviare delle notifiche di allerta;

- *Normal Check Interval* definisce l'intervallo di tempo in cui effettuare il check di un servizio. Ad esempio, se lo impostiamo a 5, questo implica che il check del servizio verrà fatto ogni 5 minuti. Tale intervallo viene rispettato sia se lo stato del servizio risulta essere OK, sia in caso contrario, ma in quest'ultima circostanza si va a considerare anche il max check attempts;
- *Retry Check Interval* rappresenta l'intervallo di tempo che bisogna aspettare prima di pianificare un ricontrollo del servizio. I service vengono rischedulati secondo questa opzione, quando lo stato del servizio risulta essere diverso da OK. Una volta che viene superato il numero di tentativi massimo senza un cambiamento di stato, il servizio viene rischedulato con un intervallo di tempo pari a quello definito in normal check interval;
- *Active/Passive Checks Enabled* vengono utilizzati per indicare se i check attivi/passivi di questo servizio sono abilitati o meno;
- *Parallelize Check* indica se i check di questo servizio possono essere fatti in parallelo oppure no;
- *Obsess Over Service* determina se Nagios sarà "ossessionato" dai risultati del check del servizio ed eseguirà il comando definito dall'utente. Tale opzione è utile per il monitoraggio di performance distribuite;
- *Check Freshness* determina se i "freshness" check per questo servizio sono abilitati o meno;
- *Freshness Threshold* rappresenta la soglia di "freshness" (in secondi). Se tale valore viene settato a 0, sarà Nagios a scegliere automaticamente il threshold;
- *Notification Enabled* viene utilizzata per indicare se sono abilitate o meno le notifiche per questo servizio;
- *Notification Interval* rappresenta l'intervallo di tempo per notificare un utente che lo stato del servizio non è più OK o che lo stato è cambiato. Se per questa opzione non viene settato nessun valore, allora automaticamente verranno considerati 60 minuti; se viene posto a 0 allora Nagios non notificherà. Una cosa importante è che tale valore deve essere superiore al normal check interval, in quanto viene richiesto di fare prima il controllo per vedere se lo

stato è cambiato o meno e, successivamente, in caso di esito negativo, inviare una notifica;

- *Notification Period* specifica il nome di un periodo di tempo durante il quale vengono lanciate le notifiche. Esattamente come il check period, anche qui è possibile indicare quando lanciare le notifiche. Nella Figura 18 è stato impostato 24x7, cioè il servizio di notifiche funzionerà sette giorni su sette e ventiquattro ore su ventiquattro. Anche in questo caso, c'è la possibilità di personalizzare il periodo.

Come detto in precedenza, la Figura 18 rappresenta la prima parte dei dettagli del servizio. La seconda parte è la seguente:

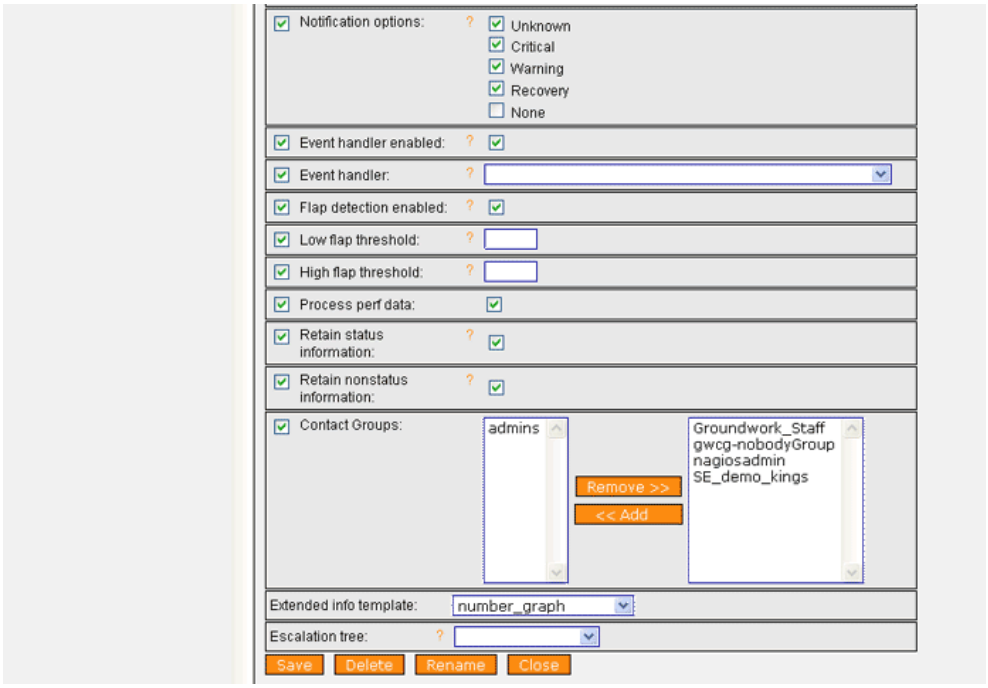



*Figura 19: Service detail parte 2*

- *Notification Options* stabilisce lo stato del servizio per cui deve essere inviata una notifica. Per comprendere meglio facciamo un esempio: nella Figura 19 sono selezionati tutti gli stati diversi dall'OK, di conseguenza quando lo stato di un servizio passa da OK ad un qualsiasi altro stato, allora Nagios invierà la notifica ai contatti specificati. Se noi togliessimo la selezione sullo stato Warning, quello che succede è che nel caso in cui lo stato del servizio passa da OK a Warning, Nagios non invierebbe nessuna notifica, in quanto tale opzione non è stata selezionata;

- *Event Handler Enabled* stabilisce se il gestore degli eventi deve essere abilitato oppure no;

- *Event Handler* specifica il nome del comando che dovrebbe essere eseguito ogni volta che viene rilevato il cambiamento di stato dell'host su cui si trova il servizio (ad esempio ogni volta che l'host va giù oppure viene riavviato);
- *Flap Detection Enable* specifica se abilitare o meno il rilevamento di flap;
- *Low/High Flap Threshold* specifica la soglia bassa/alta utilizzata per il rilevamento di flap;
- *Process Perf Data* stabilisce se si vogliono processare i dati relativi alla performance del servizio oppure no;
- *Retain Status/Non-Status Information* determina se le informazioni di stato del servizio devono essere mantenute o meno attraverso il programma di riavvio;
- *Contact Groups* indica i gruppi a cui Nagios deve inviare le notifiche;
- *Extended Info Template* definisce ulteriori informazioni;
- *Escalation Tree* indica la escalation tree impostata per il servizio.

Una volta che tutti i dettagli del servizio che abbiamo creato sono stati settati è necessario andare a specificare il check di tale servizio. Finora si è parlato spesso di check senza però avere un'idea chiara di cosa possa essere. Il check di un service non è altro che un controllo o un comando che viene eseguito sulla macchina in cui il servizio è stato inserito. Facciamo un esempio per essere più chiari: abbiamo un servizio che si chiama *Current Users* il cui obiettivo è quello di monitorare il numero di utenti connessi alla macchina su cui il service si trova. In questo caso, il check è rappresentato dal fatto di voler controllare, ad intervalli di tempo regolari, quanti utenti sono connessi alla macchina. Ogni volta che che viene fatto il check, verrà eseguito un comando, in questo caso specifico chiamato *check_local_users*, che restituirà il numero di utenti.

In termini un po' più generali, potremmo affermare che un service consiste nell'esecuzione di uno script sulla macchina che si sta monitorando per visionarne il risultato. In base a questo, si può decidere se intervenire manualmente, impostare le notifiche per avvisare chi di dovere e molto altro. Nagios fornisce dei service standard per poter iniziare a lavorare, ma come detto se ne possono creare altri a seconda delle proprie necessità e richieste.

Detto questo, vediamo come configurare il check di un servizio.

La prima cosa da fare, considerando la Figura 18, è quello di spostarsi dalla sezione Service Detail a quella del Service Check:

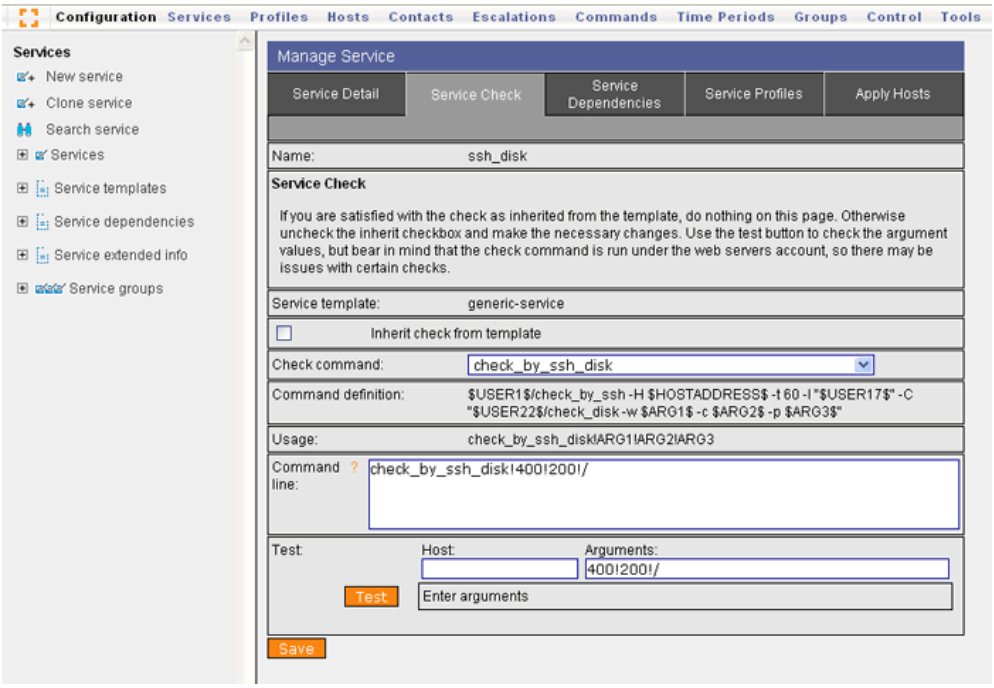



*Figura 20: Service Check*

In questa circostanza troviamo:

- *Inherit check from template* che consente all'utente di decidere se ereditare tutto dal template oppure impostare manualmente i singoli parametri. Solitamente si adotta la seconda soluzione in quanto un service difficilmente presenterà gli stessi parametri di un altro;
- *Check Command* definisce l'operazione che si vuole eseguire. Ci sono diversi command che vengono forniti direttamente da Nagios, altri possono essere creati in base alle proprie esigenze. In tal caso, bisogna realizzare lo script corrispondente e poi si può creare il comando. Una volta selezionato, nel campo *Command Definition* comparirà il path in cui si trova lo script con i corrispondenti parametri di ingresso. Quelli che compaiono anche nel campo *Usage* sono quelli che devono essere settati, tutti gli altri appartengono già al command e non possono essere modificati (per farlo bisognerebbe modificare il command stesso);
- *Command line* rappresenta il campo in cui bisogna settare i parametri di ingresso mostrati in usage. Nella Figura 20 possiamo notare come il command check_by_ssh_disk viene realizzato con lo script check_by_ssh che prende

diversi parametri in ingresso, ma solo ARG1, ARG2 e ARG3, mostrati in usage, possono essere modificati in questo campo;

- *Test* consente di testare il service check per verificare che funzioni oppure no.

Una volta che anche la sezione Service Check è stata completata, è possibile cliccare su *Save*. Anche in questa circostanza, esattamente come per gli host, è necessario andare nella sezione *Control* ed effettuare la commit affinché il servizio venga salvato ed inserito nel database di Monarch. Se questa operazione non viene fatta, il servizio viene perso.

È chiaro che una volta creato il service è possibile andare a modificare i dati anche in un secondo momento; infatti, cliccando su *Services*, invece di andare su *New Services*, bisogna andare su *Services* e scegliere dalla lista il servizio che si vuole modificare. Quando si clicca sul servizio desiderato, comparirà la stessa schermata mostrata nella Figura 18 in modo da poter accedere nelle diverse sezioni come Service Detail e Service Check e modificare ciò che si vuole. Anche quando vengono effettuate le modifiche bisogna sempre ricordare di effettuare la commit affinché le variazioni possano essere applicate.

Un altro aspetto importante è che se vogliamo creare un service simile ad uno già esistente, ma con qualche proprietà differente, c'è la possibilità di clonare il service esistente (in quel caso si cliccherà su *Clone service*) e successivamente accedere ai suoi dettagli per poter effettuare le modifiche desiderate.

Come si può notare dalle figure mostrate in precedenza, oltre alla sezione Service Detail e Service Check ci sono altre sezioni che non cito in quanto non sono state considerate dal sottoscritto perché non utili ai fini del lavoro svolto e spiegato nei Capitoli 4 e 5. Come riportato nel sotto-paragrafo 2.3.2, per maggiori dettagli è possibile consultare la documentazione ufficiale.

#### 2.3.2.3 – Command

Il command è un semplice nome che rappresenta una linea di comando in cui sono presenti lo script da eseguire e i corrispondenti parametri di ingresso:

command_name = check_local_users

command_line = $USER1$/check_users -w $ARG1$ -c $ARG2$

come possiamo notare, il command non è altro che un nome che fa riferimento all'esecuzione di uno script, check_users, che prende in ingresso i parametri ARG1 e ARG2.

La definizione del command risulta essere molto vantaggiosa perché quando si va a creare un service, non bisogna fare altro che inserire, al suo interno, il command che si desidera  e automaticamente verranno indicati i parametri di ingresso da settare; molto più veloce e più comodo rispetto a dover scrivere ogni volta e per qualsiasi servizio l'intera linea di comando.

Vediamo adesso come è possibile creare un comando.

La prima cosa importante da fare è capire se utilizzare uno script già fornito da Nagios oppure volerne creare uno nuovo; in tal caso è necessario scrivere lo script (ricordando che questo deve essere scritto in Perl) ed inserirlo nell'apposita directory /usr/local/groundwork/nagios/libexec. Arrivati a questo punto bisogna entrare nel menù di configurazione di Monarch e cliccare su *Commands*. In questa sezione è possibile creare, copiare o modificare un comando. Supponiamo di volerne creare uno nuovo:


*Figura 21: Creazione di un command*

Dalla Figura 21 possiamo notare che bisogna selezionare la *macro resource* che bisogna configurare. Se consideriamo l'esempio citato all'inizio del paragrafo, abbiamo $USER1$/check_users; questo equivale a scrivere /usr/local/groundwork/nagios/libexec/check_users. Da questo si evince che bisogna selezionare la macro resource che corrisponde al path in cui si trova lo script di interesse. Fatto questo e cliccando sul tasto *Add* ci ritroviamo nella sezione *Command Wizard*:

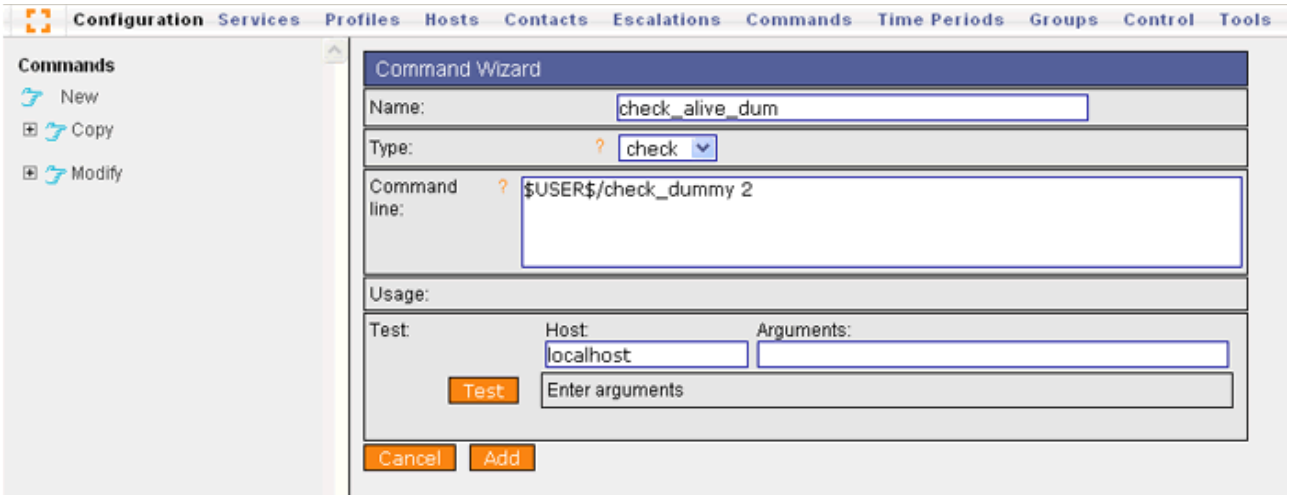



*Figura 22: Command Wizard*

Le variabili da settare sono:

- *Name* che rappresenta il nome che vogliamo dare al nostro command;
- *Type* specifica il tipo di command che vogliamo realizzare. È possibile selezionare:
  - *check* indica che il command è un controllo;
  - *notify* indica che il command è di notifica;
  - *other* indica che il command è di altro tipo, ad esempio Performance Data Collectors;
- *Command Line* definisce ciò che viene eseguito da Nagios, di conseguenza verrà specificato lo script da eseguire, i parametri fissi che non potranno essere modificati nei service ed, infine, i quelli che dovranno essere settati. Esempio:

  $USER1$/script.pl -H $HOSTNAME$ $ARG1$

  script.pl rappresenta ciò che si vuole eseguire, -H $HOSTNAME$ rappresenta il parametro fisso, quindi, ogni volta che creiamo un nuovo service ed inseriamo tale command, automaticamente verrà considerato il parametro fisso, mentre $ARG1$ rappresenta il parametro che varia. Questo dovrà essere settato nel servizio stesso;
- *Test* permette di verificare se il command funzioni oppure no.

Una volta che abbiamo settato tutte queste variabili, salviamo attraverso il tasto *Save* e, come per gli host e i service, anche qui bisogna effettuare la commit in modo che il comando venga creato e reso disponibile.

### 2.3.2.4 – Notification

Ultimo concetto chiave che deve essere configurato affinché un sistema di monitoraggio possa essere definito tale è il sistema delle notifiche. Sappiamo che non è assolutamente desiderabile dover fissare, ogni minuto, l'interfaccia per la visualizzazione dei dati, ma sarebbe meglio che quando succede qualcosa, il sistema si prenda la briga di avvisare chi di dovere, in modo da dover intervenire se si verifica qualche problema oppure essere semplicemente notificato che il problema è stato risolto. Per la notificazione, vengono coinvolti diversi elementi che devono essere correttamente configurati affinché i messaggi possano essere inviati.

Il primo passo da fare è quello di definire un command (Paragrafo 2.3.2.3 - Command) che si occupa di referenziare il corrispondente script, il quale a sua volta rappresenta il modo user-defined di invio di messaggi. Questi possono essere inviati tramite Email, SMS o qualunque altro modo l'utente desideri, in quanto è lui stesso che dovrà realizzare lo script da associare successivamente al command. Aspetto importante da ricordare è che nella definizione e configurazione del command, Figura 22, bisogna settare il *Type* come *notify* e non come *check*.

Definito il command per l'invio delle notifiche, il prossimo passo da fare è quello di definire il contatto o contatti a cui devono essere inviati i messaggi. Come sempre bisogna entrare nel menù di configurazione di Monarch, cliccare su *Contacts*, andare su *Contacts* ed, infine, su *New*.

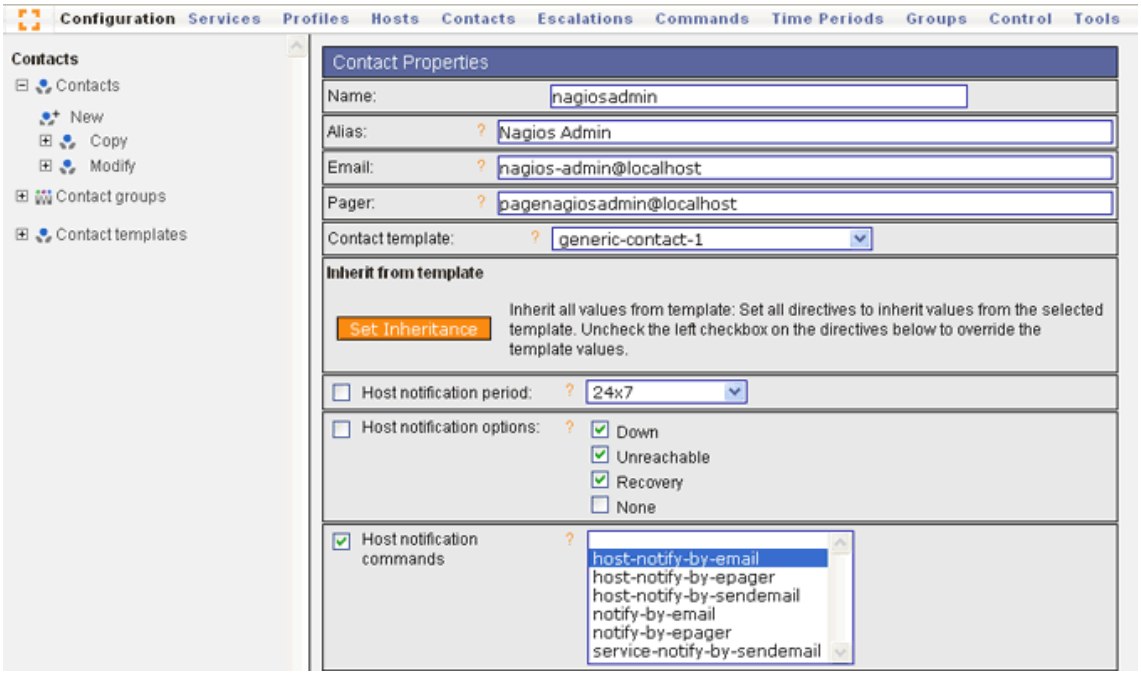



*Figura 23: Definizione di contatto parte 1*

Dalla Figura 23, possiamo notare quali sono i primi parametri che devono essere settati:

- *Name*: nome del contatto;
- *Alias*: nome più lungo o una breve descrizione del contatto;
- *Email*: indirizzo email per il contatto. In base a come è stato realizzato il command, questo indirizzo può essere usato per inviare messaggi di allarme al contatto;
- *Pager*: numero del cercapersone per il contatto. Questo campo può rappresentare anche l'indirizzo email al gateway del cercapersone;
- *Contact template*: permette di scegliere il template più adeguato al contatto;
- *Set Inheritance*: cliccando su questo tasto, vengono ereditate tutte le direttive provenienti dal template; infatti, se si modificano le opzioni, grazie a questo tasto siamo in grado di ripristinare il tutto secondo i parametri del template specificato;
- *Host notification period*: nome del periodo durante il quale il contatto può essere notificato sulle problematiche o i recovery relativi all'host. Nella Figura 23, 24x7 indica che il contatto può essere notificato tutti i giorni in qualsiasi orario;
- *Host notification options*: stati dell'host per cui la notifica deve essere inviata. Nella Figura 23 sono selezionati DOWN, UNREACHABLE e RECOVERY; questo implica che quando l'host passa allo stato DOWN o UNREACHABLE allora il contatto deve essere notificato. Stesso discorso in caso di ritorno allo stato OK. Se, ad esempio, noi eliminassimo la selezione dello stato DOWN, questo implicherebbe che quando l'host finisce in questo stato, il contatto non viene notificato;
- *Host notification commands*: lista di uno o più commands utilizzati per notificare il contatto in caso di problemi o recovery relativi all'host. Un esempio di tale command potrebbe essere *host-notify-by-email*, il quale è associato ad uno script che in determinate circostanze invia messaggi.

Esiste anche una seconda parte relativa alla configurazione del contatto ed è la seguente:

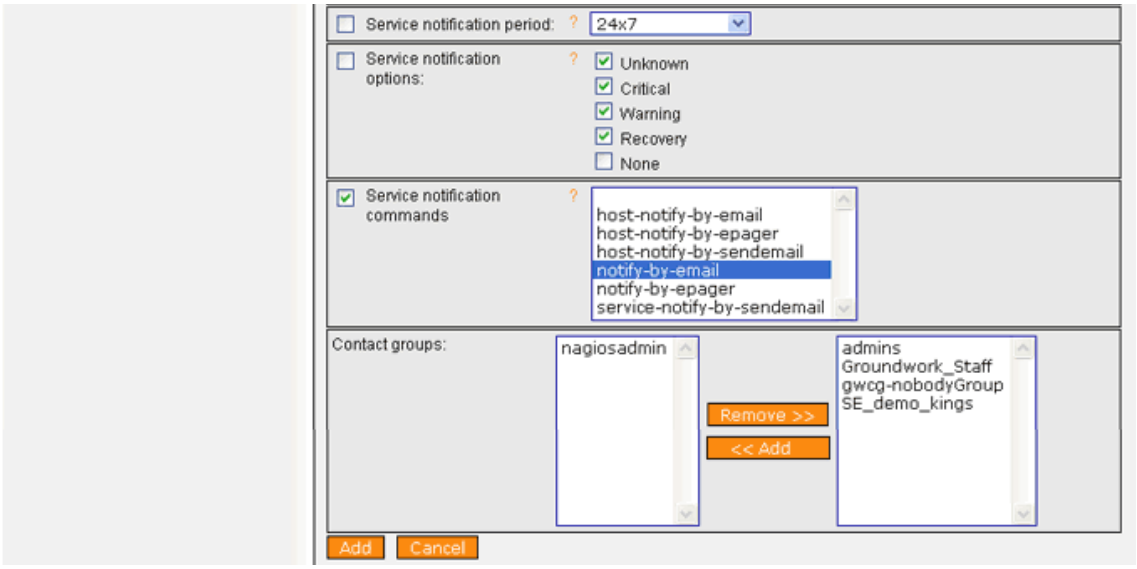



*Figura 24: Definizione di contatto parte 2*

Come si può vedere dalla Figura 24, i parametri da settare sono:

- *Service notification period*: nome del periodo durante il quale il contatto può essere notificato sulle problematiche o i recovery relativi al service che girano su un determinato host. Nella Figura 24, 24x7 indica che il contatto può essere notificato tutti i giorni in qualsiasi orario;
- *Service notification options*: stati del service per cui la notifica deve essere inviata. Nella Figura 24 sono selezionati UNKNOWN, CRITICAL, WARNING e RECOVERY*;* questo implica che quando il service passa in uno stato fra quelli selezionati allora il contatto deve essere notificato. Stesso discorso in caso di ritorno allo stato OK. Se, ad esempio, noi eliminassimo la selezione dello stato WARNING, questo implicherebbe che quando il service finisce in questo stato, il contatto non viene notificato;
- *Service notification command*: lista di uno o più command utilizzati per notificare il contatto in caso di problemi o recovery relativi al service. Un esempio di tale command potrebbe essere *notify-by-email*, il quale è associato ad uno script che in determinate circostanze invia messaggi;
- *Contact groups*: gruppo o gruppi a cui il contatto appartiene e a cui inviare le notifiche. Aspetto importante è che ci sono gruppi già definiti in Nagios, ma c'è la possibilità di crearsi gruppi personalizzati in cui specificare il nome, l'alias ed, infine, i contatti che appartengono a tale gruppo.

Una volta definiti i contatti a cui inviare le notifiche, bisogna andare a creare e configurare le escalation e le escalation tree.

Le escalation rappresentano il modo con cui Nagios avvisa gli utenti del cambiamento di stato di un dispositivo o di un servizio, mentre un'escalation tree rappresenta un gruppo di escalation che sono state assegnate ad un dispositivo e/o ad un servizio.

Detto questo, vediamo come configurare il tutto. Diamo per scontato il fatto di essere già all'interno della pagina di configurazione di Monarch in quanto abbiamo creato contatti e gruppi di contatti. A questo punto clicchiamo su *Escalations* e ci ritroviamo in una schermata che ci permette di creare e configurare le escalation e le escalation tree.

Partiamo dalle escalation, quindi clicchiamo su *Escalations* e da qui possiamo crearla sia per l'host che per il service.

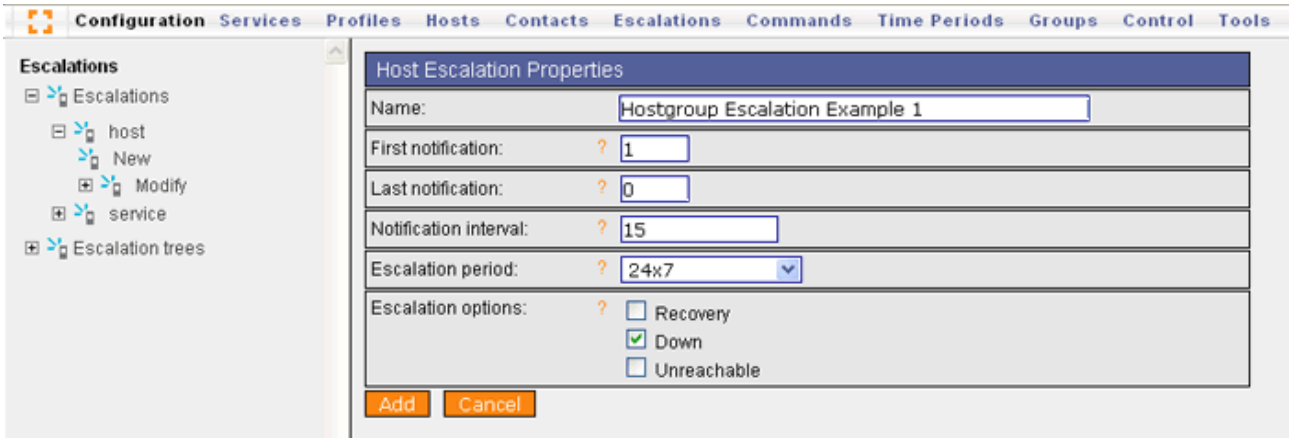



*Figura 25: Creazione Host/Service escalation*

Dalla Figura 25 possiamo notare quali sono i parametri da configurare:

- *Name*: nome dell'host/service escalation;
- *First notification*: numero che identifica la prima notifica per cui l'escalation diventa effettiva. Ad esempio, se inseriamo il valore 3, questa esclation diventerà effettiva solo se l'host o il service avranno uno stato diverso da OK per un periodo abbastanza lungo da consentire l'invio della terza notifica. Dopo che tre notifiche sono state inviate, viene resa effettiva l'escalation;
- *Last notification*: numero che identifica l'ultima notifica per cui l'escalation risulta effettiva. Ad esempio, se questo valore viene settato a 5, questa escalation non sarà utilizzata se più di 5 notifiche, relative all'host o al service, sono state inviate;
- *Notification interval*: intervallo di tempo durante il quale vengono inviate le notifiche e l'escalation risulta essere valida;

- *Escalation period*: nome del periodo durante il quale l'escalation risulta essere valida. In questo caso, 24x7 indica che l'escalation è valida per tutti i giorni e per tutte le ore;
- *Escalation options*: stati dell'host o del service per cui l'escalation può essere utilizzata. Nella Figura 25 sono indicati gli stati dell'host perché questa rappresenta la creazione di un host escalation, ma nel caso in cui creiamo una service escalation, i parametri risultano essere uguali a patto le opzioni, qui elencate, che diventano WARNING, CRITICAL, UNKNOWN e RECOVERY.

Una volta creata la service/host escalation possiamo andare a creare l'escalation tree. Anche in questo caso, è possibile crearsi delle escalation tree sia per l'host che per il service. Ricordando sempre di trovarci nella pagina di configurazione di Monarch, andiamo su *Escalations* e clicchiamo su *Escalation trees*. Da qui si può scegliere se crearne una per il service o una per l'host; la procedura è la stessa. Clicchiamo su *New* e la prima cosa che ci viene chiesta è inserire il nome dell'escalation tree, mentre il tipo risulta essere già settato (se si clicca in New nel settore host allora il tipo sarà settato a host, in caso contrario in service). Clicchiamo sul tasto *Add* per andare avanti e ci viene chiesto di aggiungere l'escalation; è chiaro che noi andiamo ad aggiungere quella che abbiamo creato precedentemente. Fatto questo, ci viene chiesto di assegnare il gruppo dei contatti che verrà notificato quando l'escalation risulterà essere valida. Terminata questa procedura, abbiamo creato e configurato anche la nostra esclation tree.

Una volta che abbiamo creato e configurato tutto ciò che è necessario, affinché il sistema di notifiche possa funzionare non dobbiamo fare altro che selezionare il servizio o l'host per cui vogliamo essere notificati ed inserire l'escalation tree che abbiamo creato. Nella Figura 19 possiamo notare il campo *Escalation tree* relativo al servizio, nella Figura 14, invece, quello relativo all'host.

Come sempre, anche se ripetuto moltissime volte, va ricordato che per salvare definitivamente le modifiche, bisogna andare su *Control* nella pagina di configurazione di Monarch ed effettuare la commit affinché tutte le creazione e modifiche effettuate possano diventare effettive.

# 3 – Zabbix

Zabbix è un software di livello enterprise progettato per il monitoraggio della disponibilità e delle performance dei componenti che costituiscono l'infrastruttura IT. In Zabbix, grazie al monitoraggio real-time ad elevate prestazioni, si possono monitorare decine di migliaia di server, macchine virtuali e dispositivi della rete simultaneamente. Insieme alla gestione dei dati, sono disponibili anche caratteristiche di visualizzazione (overview, grafi, mappe, screen, ecc…) così come modi molto flessibili di analisi dei dati per la gestione dei malfunzionamenti ed operazioni di avviso.

Zabbix consente di effettuare il monitoraggio distribuito grazie all'utilizzo di Zabbix proxy; inoltre, tale sistema di monitoraggio è dotato di un'interfaccia web-based, un'autenticazione sicura dell'utente e uno schema di permessi utente molto flessibile. Sono supportati sia il polling che il trapping e c'è la possibilità di raccogliere informazioni relative ad un dispositivo (cpu, memoria e molto altro) grazie ad agenti ad elevati prestazioni, ma se si vuole è possibile operare anche in modalità agentless (ovvero senza agenti, ma attraverso scripts esterni).

Sono supportati sia il monitoraggio Web che il monitoraggio di macchine virtuali VMware, inoltre, Zabbix può rilevare automaticamente server e dispositivi di rete ed eseguire il rilevamento di basso livello con metodi di controllo di disponibilità e prestazioni dei componenti individuati.

Zabbix è nato come sistema di monitoraggio distribuito della rete con un'interfaccia web centrale in cui è possibile gestire qualunque cosa si stia monitorando. A partire dalla versione 2.4, il numero di possibili architetture si è ridotto all'installazione di un solo server e di Zabbix proxy distribuiti.

Oggi, la più semplice architettura che permette di capire il funzionamento di Zabbix prevede tre componenti:

- Web server;
- RDBMS server;
- Zabbix server.

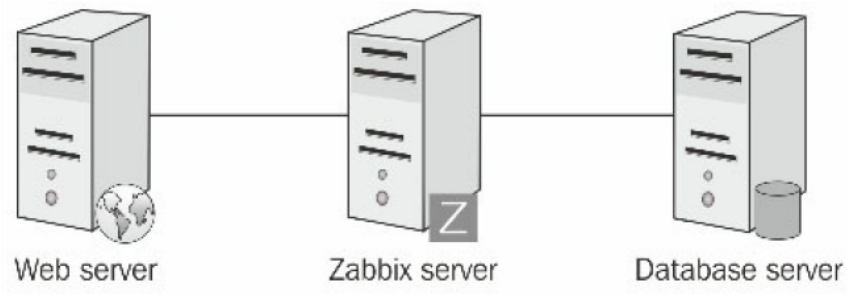



*Figura 26: Architettura Zabbix*

Zabbix server è il processo centrale del software Zabbix. Esegue il polling e il trapping dei dati, calcola i trigger ed invia le notifiche agli utenti. Rappresenta il componente centrale a cui Zabbix agent e proxy inviano dati sulla disponibilità ed integrità del sistema; inoltre, può controllare a distanza servizi di rete (come web server o mail server) utilizzando semplici check di servizio.

Zabbix server può essere visto anche come repository centrale in cui tutti i dati operazionali e statistici vengono memorizzati e come entità attiva che avviserà gli amministratori di sistema quando si verificheranno problemi su una qualsiasi macchina monitorata.

Tutte le informazioni di configurazione vengono memorizzate all'interno del database con cui Zabbix server e Web server interagiscono. Ad esempio, quando viene creato un item utilizzando l'interfaccia Web, questo viene aggiunto nella tabella degli item presente nel database. Successivamente, ogni minuto circa, Zabbix server interrogherà la tabella degli item per ottenere una lista di quelli che sono attivi e memorizzarli all'interno di una sua cache.

Come si può notare dalla Figura 26, vengono utilizzati dei server dedicati con lo scopo di consentire il funzionamento del sistema di monitoraggio anche in ambienti molto più grandi. Inoltre, c'è la possibilità di incrementare le performance del sistema estendendo l'architettura con una molteplicità di Zabbix proxy distribuiti; di conseguenza, risulta chiaro che ogni componente necessiterà di essere monitorato affinché si possa garantire il corretto funzionamento dell'intera infrastruttura.

Questo modo di lavorare è di facile mantenimento e consente di avere una soluzione di monitoraggio tendenzialmente centralizzata.

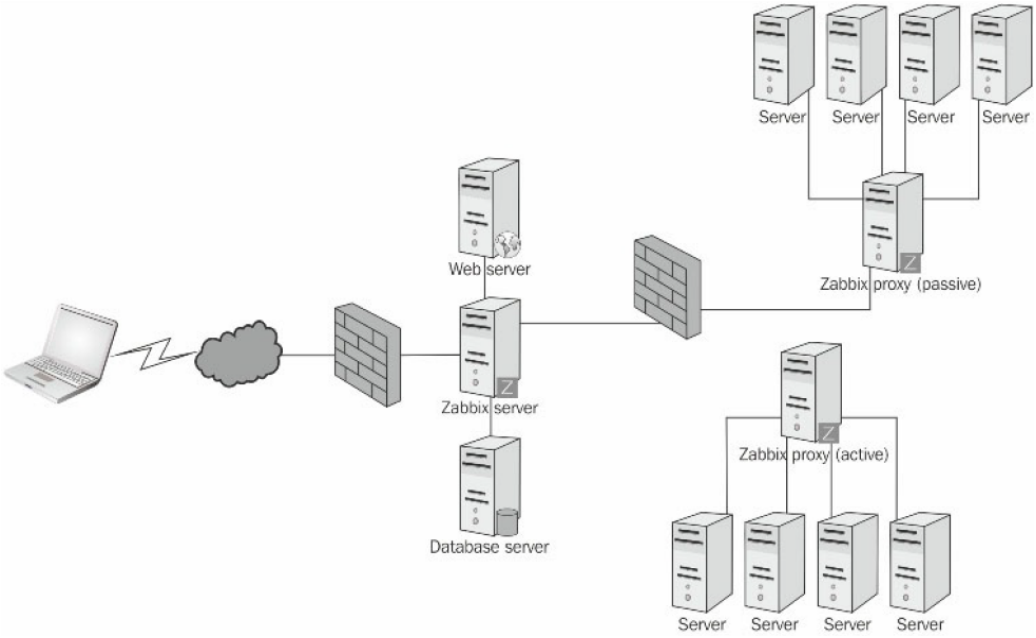



*Figura 27: Estensione dell'architettura Zabbix con proxies*

Il Zabbix proxy può essere definito come un server che viene utilizzato per raccogliere dati provenienti da un certo numero di host o dispositivi, acquisendo tutte le metriche richieste e agendo come un generico proxy. Questo significa che ha la facoltà di mantenere memorizzati i dati per un periodo di tempo arbitrario, basandosi su un database dedicato per farlo. Zabbix proxy non possiede un frontend, pertanto deve essere gestito direttamente dal server centrale (Zabbix server).

Zabbix proxy si limita al raccoglimento dei dati senza valutazioni o azioni di attivazione; per questo motivo, è sempre meglio utilizzare un server RDBMS efficiente e robusto affinché si possa prevenire la perdita di dati in caso di crash.

L'obiettivo, infatti, è quello di controllare e ottimizzare il flusso dei dati monitorati attraverso le reti e tale proxy consente di dividere ed isolare gli elementi e i dati su reti differenti. La caratteristica principale è che le metriche acquisite vengono memorizzate nel database, di conseguenza, in caso di interruzione della rete, esse non vengono perse.

Il flusso di dati standard di Zabbix è composto da una serie di componenti che inviano i dati al Zabbix server. Di tutte le fonti che possono inviare i dati al server, è possibile identificarne quattro molto importanti:

- Zabbix agent: è un agent distribuito per controllare attivamente risorse e applicazioni locali (hard drive, memoria, processore, ecc…). Ha il compito di raccogliere informazioni operazionali che poi passa al Zabbix server per un'ulteriore elaborazione; in caso di fallimento, come hard disk pieno o crash

di un processo di servizio, il server può avvisare gli amministratori della macchina che ha riscontrato tale problema. Gli agent sono estremamente efficienti in quanto effettuano chiamate native di sistema per raccogliere le informazioni desiderate. Ogni agent può eseguire controlli passivi o attivi (passive or active check):

  - in un passive check, Zabbix agent risponde ad una richiesta di dati, ovvero, il server chiede un determinato controllo (ad esempio il carico della CPU) e l'agent gli risponde fornendo il risultato;
  - active check richiedono un'elaborazione più complessa; infatti, l'agent deve prima recuperare una lista di item presenti sul Zabbix server per poterli processare in modo indipendente. Gli active check che un server può ottenere sono elencati nel parametro *ServerActive* del file di configurazione dell'agent. La frequenza di richiesta di questi controlli, invece, viene settata nel parametro *RefreshActiveChecks* presente nello stesso file di configurazione. Infine, l'agent invia periodicamente i nuovi valori al server;

- Zabbix sender: è una command line utility che può essere utilizzata per inviare i dati di performance e disponibilità al server per una successiva elaborazione. Zabbix sender può essere utilizzato anche per inviare una molteplicità di dati contenuti in un file di input;
- Zabbix proxy: è un processo che può raccogliere i dati da uno o più dispositivi monitorati e inviare le informazioni al server Zabbix; in sostanza, lavora per conto del server. Tutti i dati raccolti vengono bufferizzati a livello locale e poi trasferiti al server Zabbix a cui il proxy appartiene. L'utilizzo di uno o più proxy è opzionale, ma può essere molto utile per distribuire su di loro il carico di un singolo Zabbix server; infatti, se si lascia ai proxy il compito di raccogliere i dati, l'elaborazione sul server richiede meno risorse computazionali. Il proxy rappresenta la soluzione ideale per il monitoraggio distribuito e un aspetto molto importante è rappresentato dal fatto che richiedono un database separato;
- Altri agent (script esterni o altri componenti).

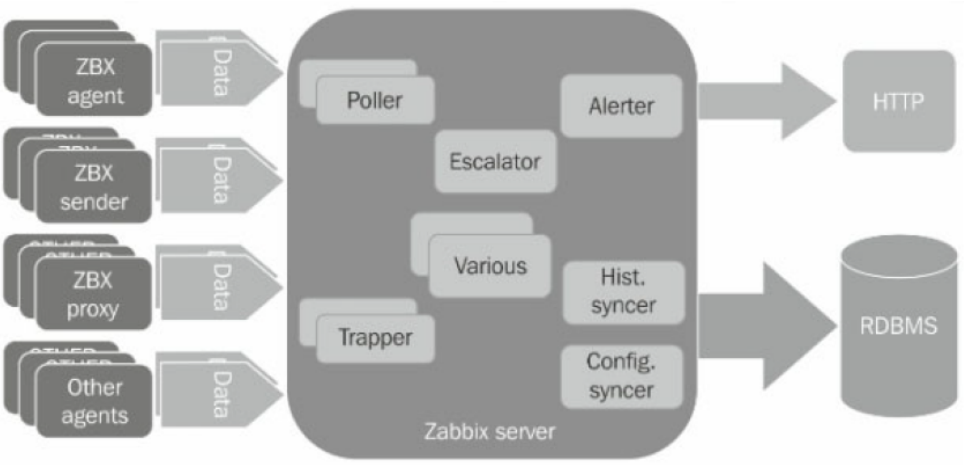



*Figura 28: Flusso dei dati*

Come è possibile notare dalla Figura 28, i dati vengono acquisiti da moltissime fonti; successivamente vengono forniti al server, il quale, a sua volta, gira il tutto alla GUI, che praticamente rappresenta gli utenti connessi (HTTP) e il database (RDBMS) utilizzato per la memorizzazione delle informazioni ricevute.

## 3.1 – Caratteristiche

In questo paragrafo, andiamo ad elencare le caratteristiche principali di Zabbix che permettono di capire con che tipo di sistema di monitoraggio si ha a che fare. È chiaro che, per comprendere fino in fondo il valore di un sistema, bisogna provarlo.

### 3.1.1 – Monitor everything

Ogni cosa, all'interno della rete, può essere monitorata: performance e disponibilità dei server, applicazioni Web, database, dispositivi di rete, applicazioni e molto altro. Utilizzando Zabbix, è possibile raccogliere tutti i dati relativi ai componenti monitorati e realizzare delle statistiche particolarmente accurate.

Il monitoraggio di indicatori di performance come CPU, memoria, rete, spazio del disco e processi possono essere facilmente monitorati attraverso agent che sono disponibili in sistemi UNIX, Linux e Windows. Gli agent, inoltre, sono processi nativi, pertanto non richiedono uno specifico ambiente di sviluppo come Java o .NET.

Zabbix supporta gli SNMP agent presenti in tutti i dispositivi di rete come router o switch; in questo modo aiuta il monitoraggio di tali dispositivi.

Attraverso i gateway di Java, c'è la possibilità di monitorare Java Application Server (JBoss, Tomcat, Oracle Application Server e molto altro) direttamente su JMX senza installazione di terze parti o integrazione di più livelli.

Fornisce anche gli IPMI agent che consentono di monitorare i parametri a livello hardware come temperatura del dispositivo, voltaggio, la velocità delle ventole e stato del disco in modo da evitare i downtime.

Quando non c'è la possibilità di installare agent sul dispositivo che si vuole monitorare, si può ricorrere al monitoraggio in modalità agentless; infatti, esistono diversi metodi che permettono di agire in questo modo.

Zabbix può usare anche regole di discovery di basso livello per individuare automaticamente macchine virtuali e VMware hypervisor e creare degli host, basati su prototipi predefiniti, per monitorarli.

Zabbix permette di controllare, in modo dettagliato, qualsiasi database come MySQL, PostgreSQL, Oracle e Microsoft SQL Server. I database sono molto importanti, però, spesso, la distribuzione delle informazioni avviene attraverso siti web o sistemi IT web-based. Zabbix fornisce una soluzione built-in anche per il monitoraggio web;

infatti, è possibile definire dei metodi che permettono di analizzare un sito web, in particolar modo, controllare la sua disponibilità e la velocità di download.

Infine, le incredibili possibilità di estensione e personalizzazione di Zabbix fanno si che tale sistema può essere integrato in qualsiasi ambiente. Inoltre, non ci sono limiti sui linguaggi di programmazione da poter usare: Shell, Perl, Python o qualsiasi altro linguaggio.

### 3.1.2 – Enterprise Ready

Zabbix è stato progettato per scalare da ambienti piccoli con decine di dispositivi a quelli molto più grandi con centinaia di migliaia di componenti monitorati. Tale sistema è in grado di monitorare 100 mila dispositivi, processando più di 3 milioni di check al minuto e di raccogliere gigabytes di dati al giorno. Questo livello di scalabilità è possibile grazie ad algoritmi intelligenti ed efficienti, che approfittano dei moderni hardware e della modularità software per garantire performance superiori. Proprio a tal scopo, gli agent utilizzano pochissime risorse di CPU e memoria , mentre Zabbix server e proxy utilizzano varie soluzioni di data caching, riducendo il carico sul database. Infine, anche i protocolli di comunicazione di rete sono estremamente efficienti.

Una risorsa molto importante per i servizi e per le applicazioni è l'alta disponibilità e per questo motivo che tutti i componenti Zabbix sono immuni alla comunicazione e alla rete grazie all'utilizzo di un data buffer control.

Altro aspetto importante è che in ambienti enterprise ci saranno moltissimi sistemi "invecchiati" che non potranno essere facilmente rimpiazzati o aggiornati. Forzare un aggiornamento di un agent solo perché il principale sistema di monitoraggio è stato aggiornato non è accettabile. Proprio per questi motivi, l'aggiornamento di Zabbix è molto facile e non richiede cambiamenti al database; infatti, il tutto è supportato attraverso molte procedure che vengono fornite.

Per quanto riguarda la sicurezza, l'accesso al frontend può essere fatto attraverso una connessione protetta SSL, garantendo la sicurezza fra utenti e server. In aggiunta, il frontend presenta un'auto-protezione contro gli attacchi di forza bruta. I componenti possono comunicare fra di loro in quanto accettano connessioni solo provenienti da indirizzi IP autorizzati, tutte le altre richieste vengono rifiutate. Infine, con il supporto alla crittografia è possibile rendere sicure le comunicazioni utilizzando un protocollo TLS (Transport Layer Security).

### 3.1.3 – Monitoraggio pro-attivo

Con lo scopo di ridurre i costi operativi, evitare i downtime e migliorare la qualità di servizio, Zabbix, per ogni evento notevole, può inviare messaggi di notifica attraverso email, SMS, Jabber o qualsiasi altro metodo user-defined, incluso la creazione di tickets automatici in Service Desk o Service Catalog System.

È in grado di effettuare automaticamente semplici azioni come il riavvio di un servizio o l'indirizzamento verso un server alternativo. Inoltre, se una prima notifica o un task automatico non sono sufficienti per risolvere un problema, è possibile utilizzare le escalation per notificare tecnici esperti e amministratori di sistema.

Quando qualcuno sta lavorando su un problema, l'analista può riconoscerlo e anche commentare. Questa caratteristica aiuta a migliorare il lavoro di squadra, consentendo la gestione ad alto livello di problemi. Il risultato è un ambiente operativo più controllato, con tempi di inattività ridotti e clienti più soddisfatti.

### 3.1.4 – Pianificazione delle capacità

Ottenere nuove attrezzature può richiedere diverse settimane, pertanto è richiesta una pianificazione anticipata, da parte dell'amministratore dei sistemi, che impedisce l'utilizzo delle risorse per mesi. Zabbix fornisce dati per effettuare analisi statistiche con molta facilità; per esempio esaminando la crescita di utilizzo del disco si può stimare esattamente quando lo spazio disponibile sarà esaurito. Così è possibile impedire il verificarsi di situazioni critiche, come ad esempio un sovraccarico di energia, uso eccessivo di link Internet o esaurimento di spazio di archiviazione.

In Zabbix si può facilmente rilevare lo spreco di CPU, memoria, disco o larghezza di banda, non solo su un singolo dispositivo, ma su un intero gruppo di server; in questo modo, si sarà in grado di riallocare le applicazioni e le apparecchiature per utilizzare al meglio le risorse disponibili.

### 3.1.5 – Open Source

Zabbix è rilasciato sotto la licenza GPL, quindi è gratuito sia per uso commerciale che non. Non ci sono limitazioni sul numero di dispositivi che Zabbix può monitorare. È possibile modificare il codice sorgente, da utilizzare internamente, per adattare il sistema, nonché sviluppare strumenti personalizzati da integrare con Zabbix stesso. Tutte le impostazioni ed i valori raccolti vengono memorizzati in un formato semplice e sono completamente aperti, facile da esportare o integrati con altri sistemi.

## 3.2 – Concetti chiave

Come ogni sistema di monitoraggio, anche Zabbix presenta dei concetti chiave che caratterizzano il suo funzionamento e la sua qualità. Ci sono degli aspetti che vanno assolutamente considerati in quanto rappresentano l'essenza del sistema stesso.

### 3.2.1 – Raccoglimento dei dati

Zabbix presenta un insieme di fonti per il raccoglimento di informazioni relative alla disponibilità e alle performance delle infrastrutture monitorate. Tutto questo offre, all'amministratore di sistema, una grande flessibilità quando ci si trova a dover scegliere la soluzione migliore per affrontare una situazione.

La sorgente migliore è rappresentata dall'utilizzo di Zabbix agent, in quanto sono processi nativi sviluppati in linguaggio C che possono eseguire su diverse piattaforme (come UNIX, Linux e Windows) e consentono di raccogliere dati come CPU, memoria, disco e interfaccia di rete utilizzati da un dispositivo. Grazie al suo piccolo footprint, ogni agent può eseguire su un dispositivo con risorse limitate; inoltre, le configurazioni di monitoraggio sono tutte incentrate sul server rendendo la gestione dell'agent particolarmente semplice, infatti, ognuno di questi utilizza un singolo file di configurazione su tutti i Zabbix server.

Gli agent supportano sia i check passivi (polling) che quelli attivi (trapping) che possono essere schedulati secondo intervalli di tempo personalizzati.

Nei check passivi, il server o il proxy richiede un determinato dato all'agent, il quale raccoglierà le informazioni e successivamente risponderà.

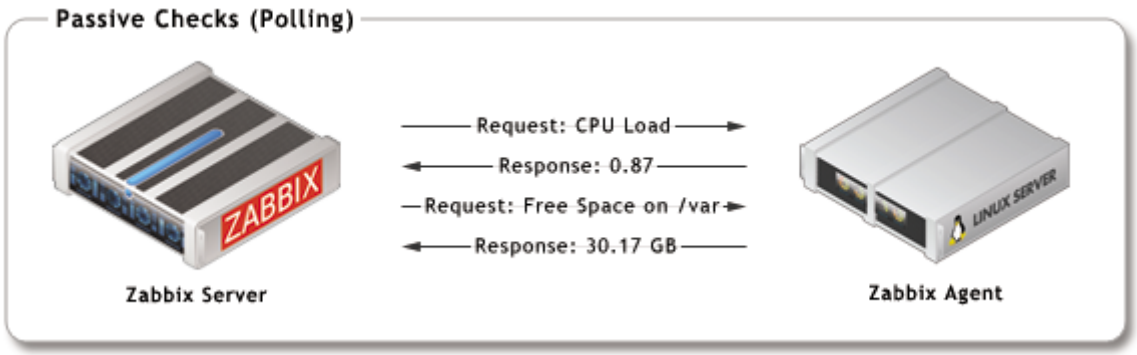



*Figura 29: Check passivo*

Nei check attivi, l'agent richiede al server o al proxy una lista di controlli attivi. Successivamente, l'agent si occuperà di mandare, periodicamente, i risultati dei check.

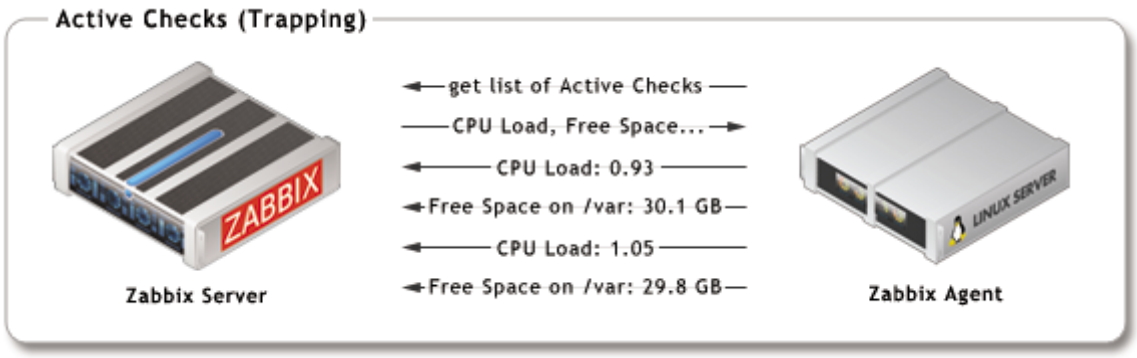



*Figura 30: Check attivo*

Per quanto riguarda alcune funzionalità offerte da un agent potremmo considerare la seguente tabella:

| | |
|---|---|
| Network | Packets/bytes transfered<br>Errors/dropped packets<br>Collisions |
| CPU | Load average<br>CPU idle/usage<br>CPU utilization data per individual process |
| Memory | Free/used memory<br>Swap/pagefile utilization |
| Disk | Space free/used<br>Read and write I/O |
| Service | Process status<br>Process memory usage<br>Service status (ssh, ntp, ldap, smtp, ftp, http, pop, nntp, imap)<br>Windows service status<br>DNS resolution<br>TCP connectivity<br>TCP response time |
| File | File size/time<br>File exists<br>Checksum<br>MD5 hash<br>RegExp search |
| Log | Text log<br>Windows eventlog |
| Other | System uptime<br>System time<br>Users connected<br>Performance counter (Windows) |

*Figura 31: Alcune funzionalità di un agent*

Inoltre, un agent è in grado di scrivere e monitorare sia file di log di testo, ma anche Windows Event Log. Tali file vengono costantemente analizzati dall'agent in modo da poter notificare il server, il quale deciderà sul da farsi. Supporta anche il Windows Management Instrumentation (WMI) per monitorare ed ottenere facilmente informazioni e performance real-time da workstation e da server Windows.

Infine, una potentissima funzionalità dell'agent è quella di eseguire degli script user-defined; infatti, ogni agent può essere esteso aggiungendo alle sue funzionalità script scritti in diversi linguaggi di programmazione come Shell, Perl, Python, Ruby, ecc….

Il risultato ottenuto dall'esecuzione dello script viene inviato al server che deciderà cosa fare.

Poi ci sono gli SNMP agent (versioni supportate sono v1, v2 e v3) che permettono di raccogliere dati da quei dispositivi su cui è installato SNMP. Questi non sono solo presenti su componenti di rete, ma anche su stampanti, NAS, UPS. Tipicamente, ogni dispositivo presente sulla rete può essere monitorato attraverso questi agent.

Per ottenere, invece, informazioni relative all'hardware di un dispositivo, vengono utilizzati IPMI agent, che sono presenti di default su server con architetture Intel come HP iLO e Dell DRAC. Attraverso questi agent, siamo in grado di controllare la temperatura della CPU, la velocità di ventilazione, voltaggio del sistema e stato del disco fisico.

Attraverso Java gateway, è possibile monitorare applicazioni Java utilizzando la tecnologia JMX (Java Management Extensions); in questo modo, il server richiede al gateway uno specifico JMX counter, che, da remoto, attraverso le API JMX può raccogliere informazioni su applicazioni Java senza la necessità di installare ulteriori componenti. Alcuni esempi di applicazioni che possono essere monitorati con JMX sono JBoss, Tomcat, GlassFish, WebSphere, ecc….

Zabbix supporta anche il monitoraggio VMware e di macchine virtuali; infatti, presenta regole di discovery di basso livello che permettono di trovarli automaticamente. Per automatizzare pienamente il processo, vengono prima creati dei prototipi di host con tali regole di discovery, successivamente, quando una macchina virtuale viene trovata, le informazioni ricevute vengono utilizzate per girare questi prototipi in macchine reali.

Attraverso la tecnologia ODBC, un server può memorizzare i dati in RDBMS database come MySQL, PostgreSQL, Oracle e Microsoft SQL Server. In seguito ad una query, il risultato viene memorizzato per consentire di creare grafi, allarmi o notifiche in caso di malfunzionamenti, indisponibilità o fallimenti.

In conclusione, oltre alla molteplicità di agent forniti da Zabbix, c'è la possibilità di agire anche in modalità agentless.

| | |
|---|---|
| Network Services | TCP port availability<br>TCP port response time<br>Service check |
| ICMP Ping | Server availability<br>ICMP response time<br>Packet loss |
| Remote Check | Executing commands via SSH or Telnet |

*Figura 32: Funzionalità agentless*

Zabbix server può controllare se un servizio è in ascolto su una determinata porta o se risponde in modo appropriato. Questi metodi, al momento, sono supportati per FTP, IMAP, HTTP, HTTPS, LDAP, NNTP, POP3, SMTP, SSH, TCP e TELNET.

Zabbix può controllare se un host risponde a ICMP ping in modo da capire se c'è la disponibilità e analizzare il tempo di risposta e la presenza di pacchetti persi.

Infine, se la configurazione di un agent non è possibile, allora il server può eseguire comandi attraverso SSH e Telnet (se disponibili) raccogliendo i risultati ottenuti.

### 3.2.2 – Rilevamento dei problemi

Una volta che i dati sono stati raccolti, si può passare all'analisi degli stessi. Regole di valutazione dei dati, chiamati trigger, forniscono informazioni sullo stato del dispositivo monitorato. Quando la soglia di un trigger viene superata, esso cambia il suo stato da OK a PROBLEM e viceversa in caso si ritorna sotto soglia. Tale metodo è molto importante per la rilevazione dei problemi, ma ancora di più potrebbe essere la predizione del problema. Zabbix fornisce delle funzioni che analizzano il trend dei dati in ingresso, costruiscono una previsione di come le cose potrebbero andare e forniscono all'utente la possibilità di agire pro-attivamente.

Per ogni trigger, è possibile definire una o più espressioni logiche che possono essere collegate fra loro attraverso operatori logici AND e OR. Gli si può assegnare, anche, uno fra i sei livelli di severity che viene applicata alla rappresentazione grafica del trigger stesso.

| SEVERITY | DEFINITION | COLOUR |
|---|---|---|
| **Not classified** | Unknown severity. | Grey |
| **Information** | For information purposes. | Light blue |
| **Warning** | Be warned. | Yellow |
| **Average** | Average problem. | Orange |
| **High** | Something important has happened. | Light red |
| **Disaster** | Disaster. Financial losses, etc. | Red |

*Figura 33: Livelli di severity*

Zabbix consente di utilizzare item differenti appartenenti a host diversi per creare un'espressione del trigger; questo permette di creare delle soglie complesse e molto intelligenti.

Le funzionalità di tale sistema consentono di controllare anche lo stato del dato corrente con quello ottenuto in passato; di conseguenza è possibile confrontare intervalli di tempo differenti in modo da analizzare le variazioni che si sono avute durante quel periodo.

Infine, per gestire il flapping, Zabbix fornisce funzioni di isteresi che presentano due soglie: una alta e una bassa. Questo permette di definire delle "zone" al di fuori delle quali è possibile lanciare degli allarmi; ad esempio se consideriamo l'utilizzo della CPU, attraverso funzioni di isteresi possiamo definire due soglie: 10% e 20%. Se il valore corrispondente alla CPU è all'interno della finestra definita dalla due soglie, non ci sono problemi; in caso contrario lo stato del trigger passa da OK a PROBLEM.

### 3.2.3 – Visualizzazione

Per la visualizzazione dei dati da monitorare, Zabbix offre diverse soluzioni:

- **Dashboard globale** che rappresenta la pagina centrale del frontend di Zabbix e consente di personalizzare l'ambiente di monitoraggio come si vuole. Le informazioni disponibili sono lo stato del Zabbix server, stato del sistema, stato delle macchine, ultimi 20 problemi, monitoraggio web, stato di discovery, grafici, screen e map preferite.

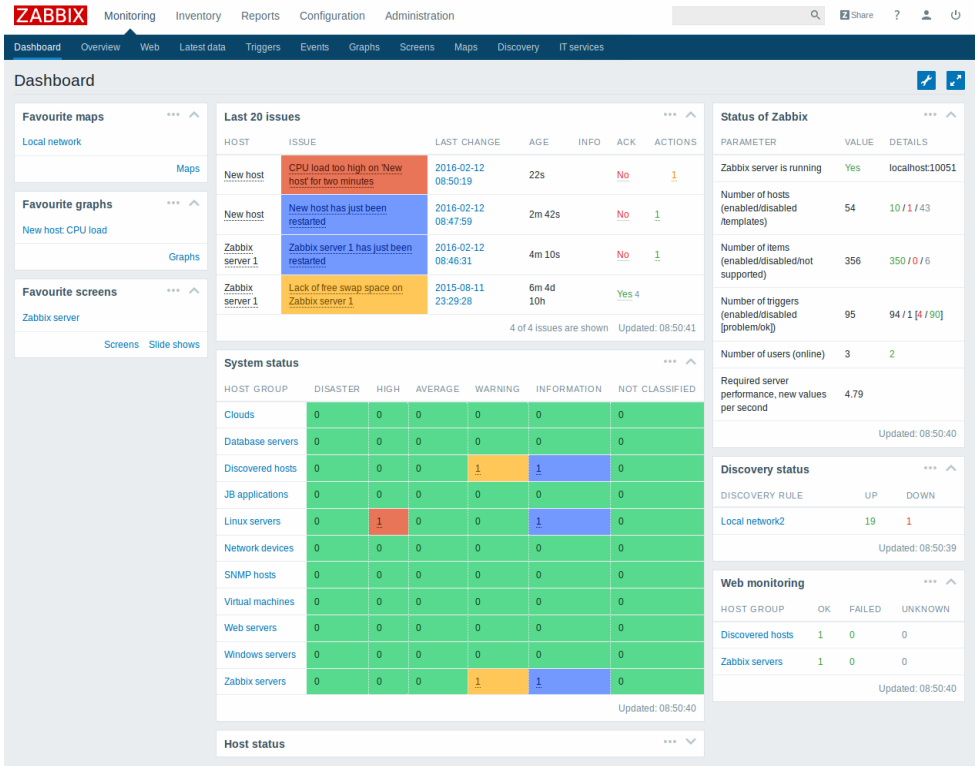



*Figura 34: Dashboard*

- **Grafici** che rappresentano l'andamento dei dati raccolti e memorizzati. Grafici standard per valori numerici sono disponibili, a runtime, senza richiedere alcun tipo di configurazione.

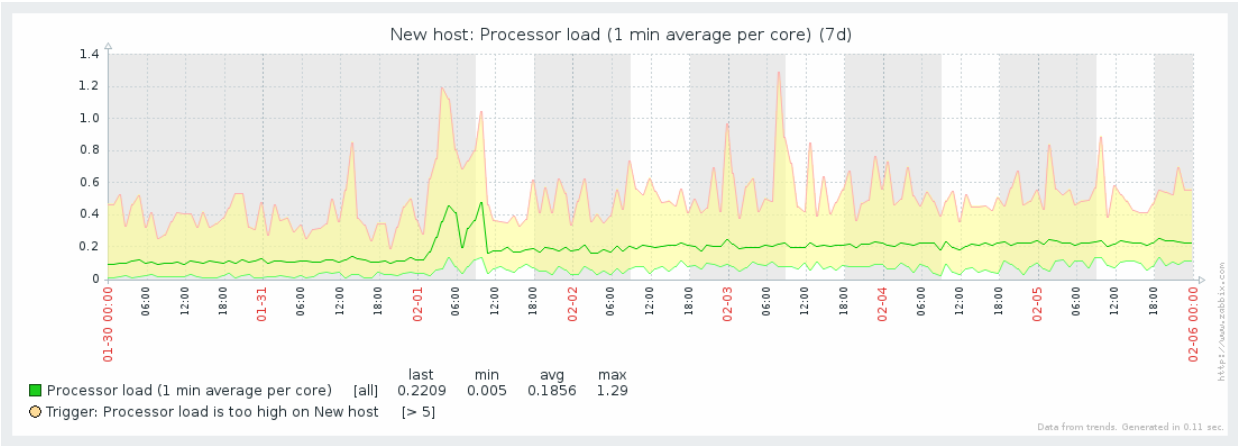



*Figura 35: Grafico standard*

C'è la possibilità di zoomare porzioni di grafico per analizzarlo in modo più dettagliato e stabilire intervalli di tempo di interesse da visualizzare.

I grafici possono essere personalizzati così da specificare lo stile desiderato e le linee guida di interesse da mostrare. Un aspetto molto interessante è rappresentato dal fatto che si possono creare grafici che mostrano il confronto fra item differenti.

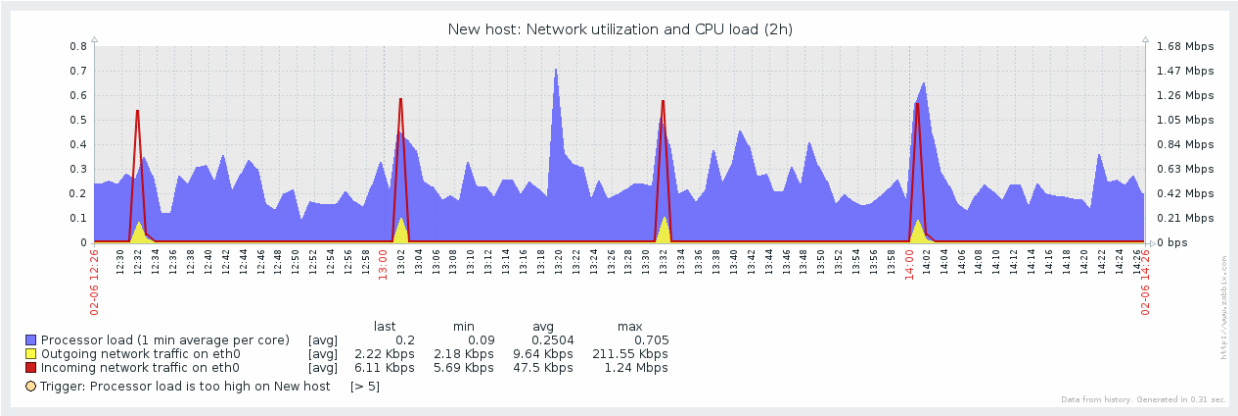



*Figura 36: Grafico personalizzato*

Nella Figura 36 viene mostrato il carico del processore (in blu), il traffico di rete in uscita sull'interfaccia eth0 (in giallo) e il traffico di rete in entrata sull'interfaccia eth0 (in rosso);

- **Mappe** che offrono la possibilità di creare un ambiente monitorato user-friendly. Ogni elemento sulla mappa può rappresentare un host, un gruppo di host, un trigger, un'immagine o anche un'altra mappa. Dal momento in cui gli elementi della mappa vengono collegati fra di loro e con uno o più trigger, tale mappa si anima mostrando quello che succede. Quando si verifica un evento, le icone e il colore dei link cambiano automaticamente per consentire all'utente di osservare quello che succede nell'infrastruttura che si sta monitorando;

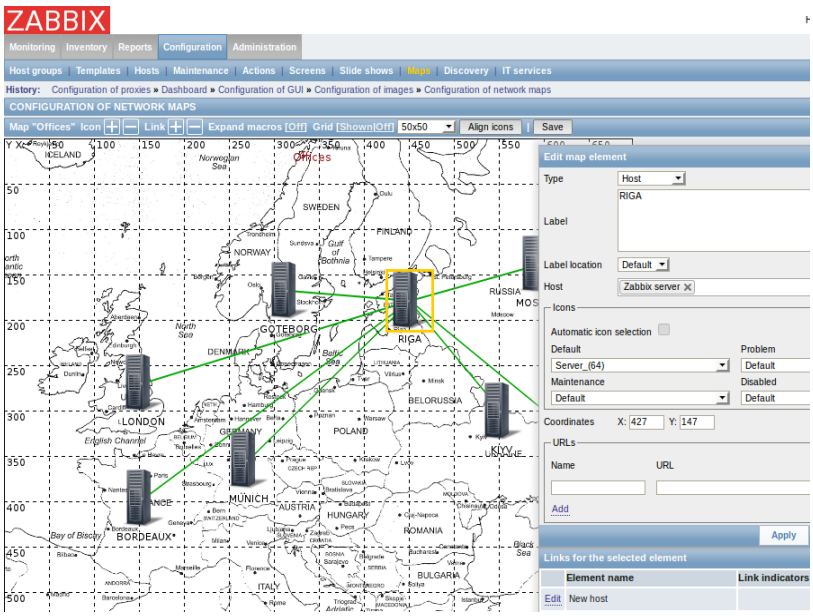



*Figura 37: Esempio di mappa*

- Tutti i dati raccolti e memorizzati non necessariamente devono essere mostrati in formato grafico, ma possono essere visti anche come **semplici dati**;

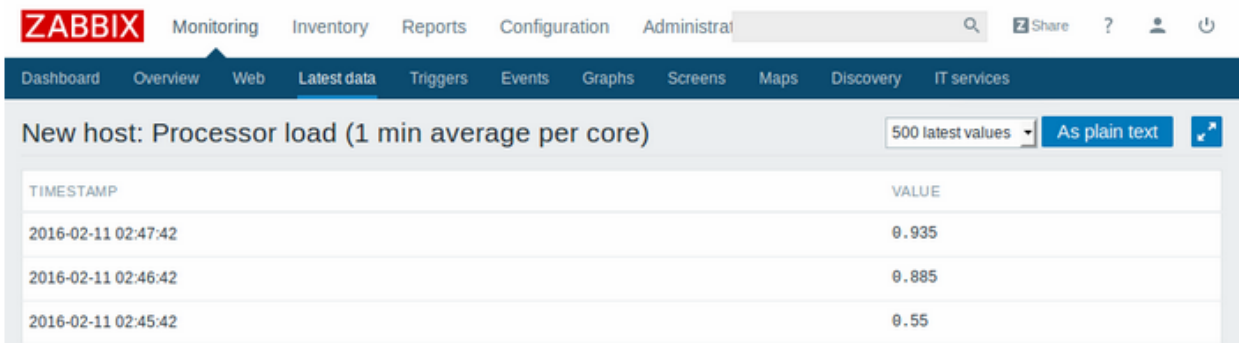



*Figura 38: Raw Data*

- **Eventi e dettagli di notifica** che vengono mostrati attraverso una lista

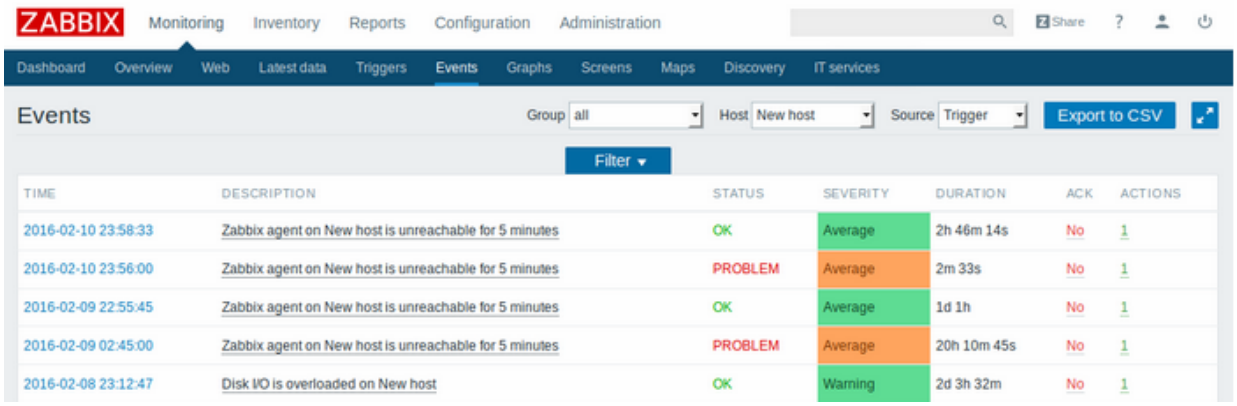



*Figura 39: Eventi e dettagli di notifica*

## 3.2.4 – Sistema di notifica

Zabbix non solo consente di acquisire, memorizzare ed analizzare informazioni sull'ambiente monitorato, ma anche di informare il personale addetto del verificarsi di eventi importanti, usando diversi canali e opzioni. Zabbix fornisce un workflow completo per inviare notifiche, raccogliere gli acknowledge degli amministratori, scalare il problema ad altri operatori e addirittura eseguire azioni in automatico sui sistemi coinvolti.

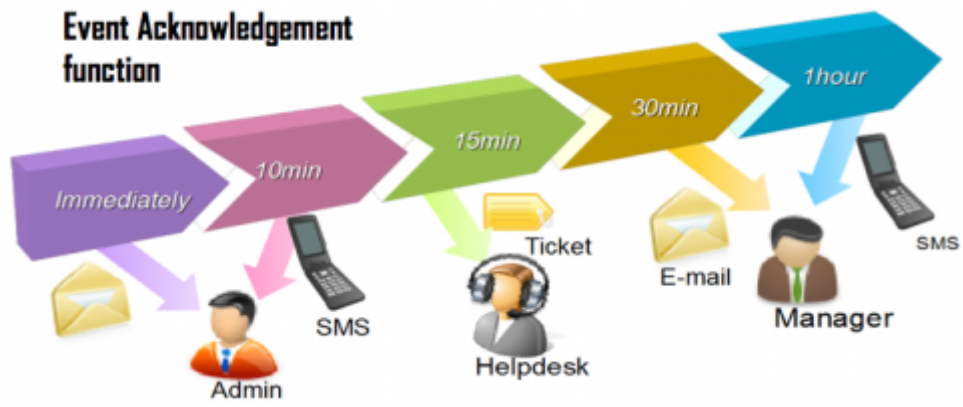



*Figura 40: Sistema di notifiche*

Zabbix dispone di diversi metodi predefiniti per l'invio delle notifiche. Gli amministratori possono ricevere notifiche:

- via e-mail;
- via SMS;
- via Jabber (messaging protocol);
- attraverso uno script personalizzato.

Inoltre gli alert possono essere controllati da script. Il contenuto delle notifiche è completamente personalizzabile in base al contesto; infatti, all'occorrenza ogni contatto può essere configurato per ricevere notifiche solo per i livelli di severity desiderati, utilizzando i canali prescelti e solo negli orari e giorni specificati.

Ogni notifica può contenere tutte le informazioni per consentire all'amministratore di comprendere il problema e prendere la giusta decisione: i dati identificativi del dispositivo, l'estratto dei file di log, un link verso l'interfaccia di management del dispositivo o alla documentazione online.

Nell'invio ad uno specifico utente o gruppo, il messaggio relativo ad un medesimo problema può essere personalizzato per includere un set di informazioni differenti, in base al ruolo del destinatario (es.: operatore, amministratore, manager).

Zabbix offre anche l'opportunità di eseguire delle azioni automatiche; infatti, quando un trigger si attiva, comandi di shell possono essere eseguiti automaticamente sui sistemi remoti, ad esempio per rimediare a situazioni in cui un sistema è sovraccarico o dei servizi hanno smesso di funzionare. L'utilizzo tipico di questa funzione è per riavviare un servizio o eseguire il reboot di un server. I comandi possono essere eseguiti:

- su Zabbix server;

- su Zabbix agent;
- utilizzando il protocollo IPMI;
- utilizzando il protocollo Telnet oppure SSH.

Infine, Zabbix, quando si verifica un problema, può notificare anche attraverso l'escalation.

L'escalation descrive uno scenario in cui, inizialmente, viene inviata una notifica ad un solo destinatario. Se il problema persiste e nessuno acknowledge viene ricevuto, verranno notificati altri destinatari ed, in assenza di ulteriori riscontri, il server eseguirà un comando automatico. Zabbix fornisce regole estremamente efficaci e flessibili per definire uno scenario di escalation, infatti, le funzioni supportate sono:

- notifica immediata dei nuovi problemi agli utenti;
- monitoraggio pro-attivo con Zabbix che esegue script predefiniti (comandi remoti);
- ripetizione della notifica finché il problema non viene risolto;
- notifiche ritardate;
- scalatura del problema ad altri gruppi di utenti;
- possibilità di scalare i problemi con o senza acknowledge;
- invio di messaggi di recovery a tutti i contatti coinvolti;
- possibilità di avere un numero illimitato di escalation.

Nelle notifiche può essere inserita l'intera cronologia di escalation in modo che il destinatario possa vedere cosa sta succedendo e perché ha ricevuto il messaggio.

## 3.3 – Configurazione degli elementi principali

Come per ogni sistema di monitoraggio, anche per Zabbix gli elementi principali sono i dispositivi che si vogliono monitorare, il servizio che si vuole eseguire sul dispositivo in modo da raccogliere i risultati, l'espressione logica che valuta i dati raccolti dal servizio e calcola lo stato del sistema ed, infine, il sistema di notifica che allerta gli utenti quando si verifica un problema o quando viene ripristinata la situazione.

Ricapitolando ed utilizzando la terminologia di Zabbix, gli elementi chiave sono:

- Host;
- Item;
- Trigger;
- Notification.

### 3.3.1 – Host

Tipicamente Zabbix host sono i dispositivi che si vogliono monitorare (server, workstation, switch, ecc…). La creazione dell'host è il primo passo da fare per poter iniziare a lavorare con il sistema di monitoraggio; infatti, se si vogliono monitorare dei parametri, bisogna stabilire prima di quale dispositivo e successivamente creare i corrispondenti servizi.

Per poter creare e configurare un host, bisogna andare nella sezione *Configuration* del frontend di Zabbix e cliccare su *Hosts*. Arrivati qui, sulla destra, cliccare su *Create host*.

Un aspetto importante è che c'è la possibilità di clonare un dispositivo, selezionandolo e cliccando su *Clone* o *Full clone*. Nel primo caso, vengono ereditati i parametri dell'host e il collegamento al template, nel secondo caso, invece, vengono ereditate anche tutte le entità configurate per quel dispositivo come applicazioni, item, trigger, grafici, ecc….

Un'osservazione da non trascurare è che ogni dispositivo clonato eredita il template con cui l'originale è collegato, pertanto qualsiasi cambiamento, relativo a livello di template, viene effettuato sull'originale non viene ereditato dalla copia in quanto quest'ultima eredita tutte le caratteristiche del template.

Ritornando alla creazione di un nuovo host, noi avremo la seguente schermata:

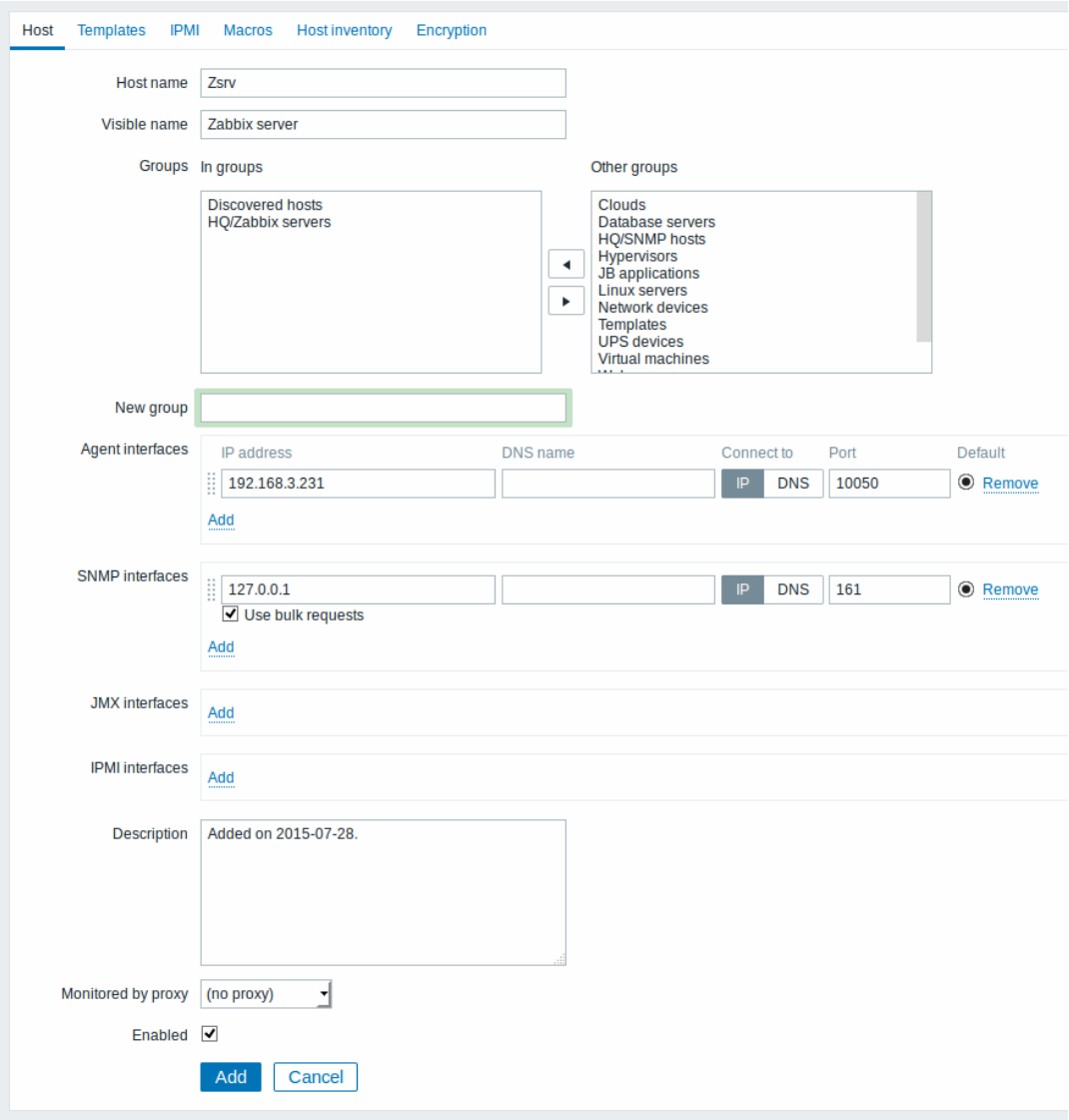



*Figura 41: Creazione di un host*

Nella sezione Host mostrata nella Figura 41, i parametri da settare sono:

- *Host name*: rappresenta il nome del dispositivo che si vuole creare. Se su tale device esegue anche un Zabbix agent, allora è necessario settare il parametro Hostname del file di configurazione dell'agent con tale valore; questo è necessario per poter elaborare gli active check;
- *Visible name*: se settato, specifica il nome che verrà visualizzato in liste, mappe, ecc…;
- *Groups*: indica il gruppo al quale l'host appartiene. Il dispositivo deve appartenere ad almeno un gruppo;
- *New host groups*: consente di creare un nuovo gruppo a cui l'host appartiene; se viene lasciato vuoto viene ignorato;

- *Interfaces*: permette di specificare il tipo di interfaccia del dispositivo: Agent, SNMP, JMX e IPMI. Per poterla configurare, bisogna cliccare su *Add* nella tipologia di interesse, inserire l'indirizzo IP se il *Connect to* è settato su IP, DNS name se è settato su DNS, la porta ed, infine, selezionare quale interfaccia deve essere quella di default;
- *Description*: rappresenta una breve descrizione della macchina;
- *Monitorated by proxy*: specifica se il dispositivo deve essere monitorato dal Zabbix server (no proxy) oppure da Zabbix proxy. In questo caso si seleziona il nome del proxy corrispondente;
- *Enabled*: consente di stabilire se attivare il monitoraggio del dispositivo oppure no.

Nella sezione successiva, ovvero Templates, avremo questa schermata:

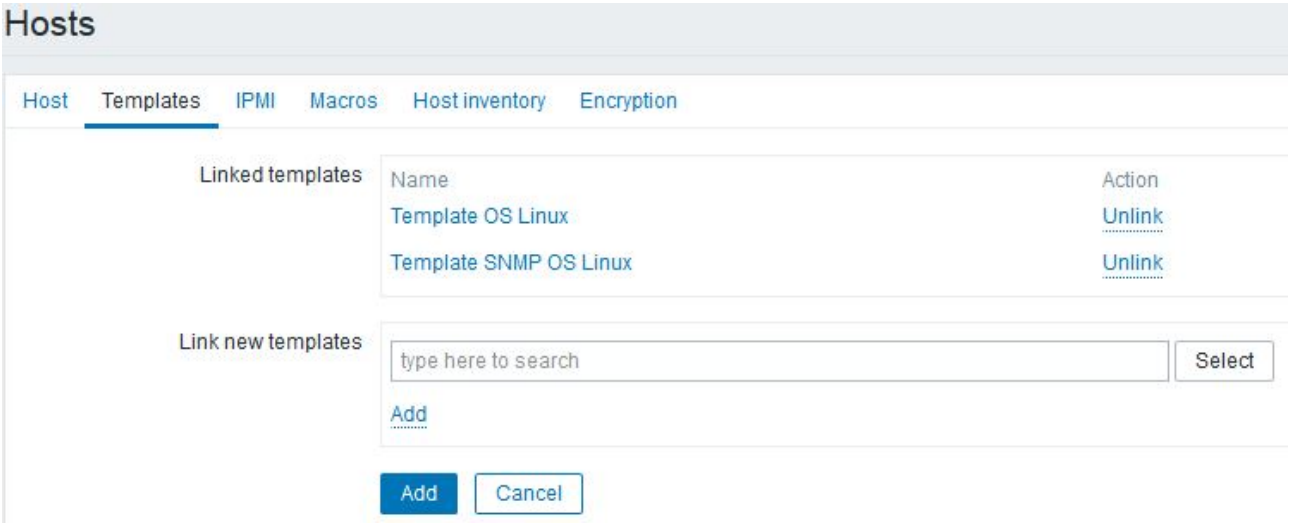



*Figura 42: Templates*

in questo caso si possono selezionare diversi template che vogliamo associare al nostro dispositivo. La prima cosa da fare è cliccare su *Select* che ci mostrerà una lista di template già forniti da Zabbix; chiaramente si possono creare nuovi template del tutto personalizzati o clonarli in caso di lievi modifiche. Dalla lista andiamo a selezionare quelli di interesse (nella Figura 42 abbiamo Template OS Linux e Template SNMP OS Linux) che compariranno in *Link new templates*. A questo punto clicchiamo su *Add* (al di sotto della barra di scrittura) per inserirli in *Linked templates*. Dalla Figura 42, inoltre, si può notare un'ulteriore azione che può essere svolta che prende il nome di *Unlink*. Tale operazione permette di "scollegare" il template dal nostro dispositivo.

Nella sezione successiva, cioè IPMI, vengono definite delle opzioni per la gestione IPMI come *Algoritmo di autenticazione* (MD2, MD5, OEM, ecc…), *livello di privilegio* (User, Operator, Admin, ecc…), *username* e *password* per l'autenticazione.

Andando ancora avanti, abbiamo la sezione Macros che permette di definire le macro utente a livello di host.

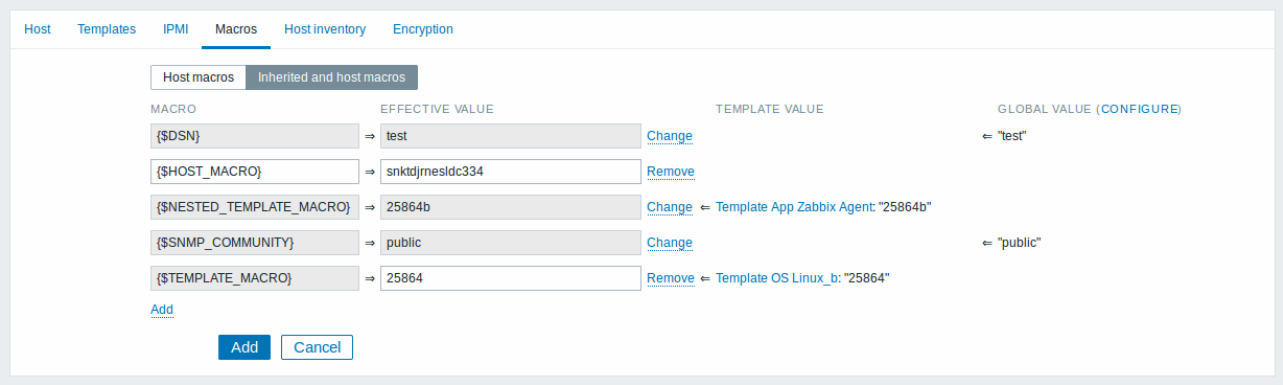



*Figura 43: Macros*

È possibile, inoltre, visualizzare anche le macro globali e quelle a livello di template andando su *Inherited and host macros*.

Le ultime due sezioni rimaste sono Host Inventory ed Encryption. La prima permette di ottenere informazioni di inventario; in questo caso è possibile disattivare tale opzione, inserire le informazioni manualmente o consentire il riempimento automatico. La seconda consente di creare delle connessioni crittografate col dispositivo. Non avendole considerate per il lavoro svolto, rimando alla documentazione ufficiale [3]. Una volta che tutte gli attributi sono state configurati è possibile cliccare su tasto *Add* (in fondo ad ogni Figura) per creare il nostro dispositivo pronto per essere monitorato.

## 3.3.2 – Item

L'item è quel servizio che consente di raccogliere i dati provenienti dal dispositivo che si sta monitorando. Una volta configurato l'host, infatti, bisogna aggiungere item desiderati per ottenere i dati di interesse. Un modo veloce per aggiungere diversi item è quello di collegare uno o più template (con item predefiniti) al dispositivo che si vuole controllare; in alternativa, si possono creare e configurare item secondo i propri desideri e necessità.

Per la creazione di un item, la prima cosa da fare è accedere al frontend di Zabbiz, cliccare su *Configuration*, poi *Hosts*, selezionare il dispositivo di interesse e andare

nella sezione *Items*. In alto a destra possiamo trovare il tasto *Create item*, sul quale cliccando accediamo alla seguente schermata:

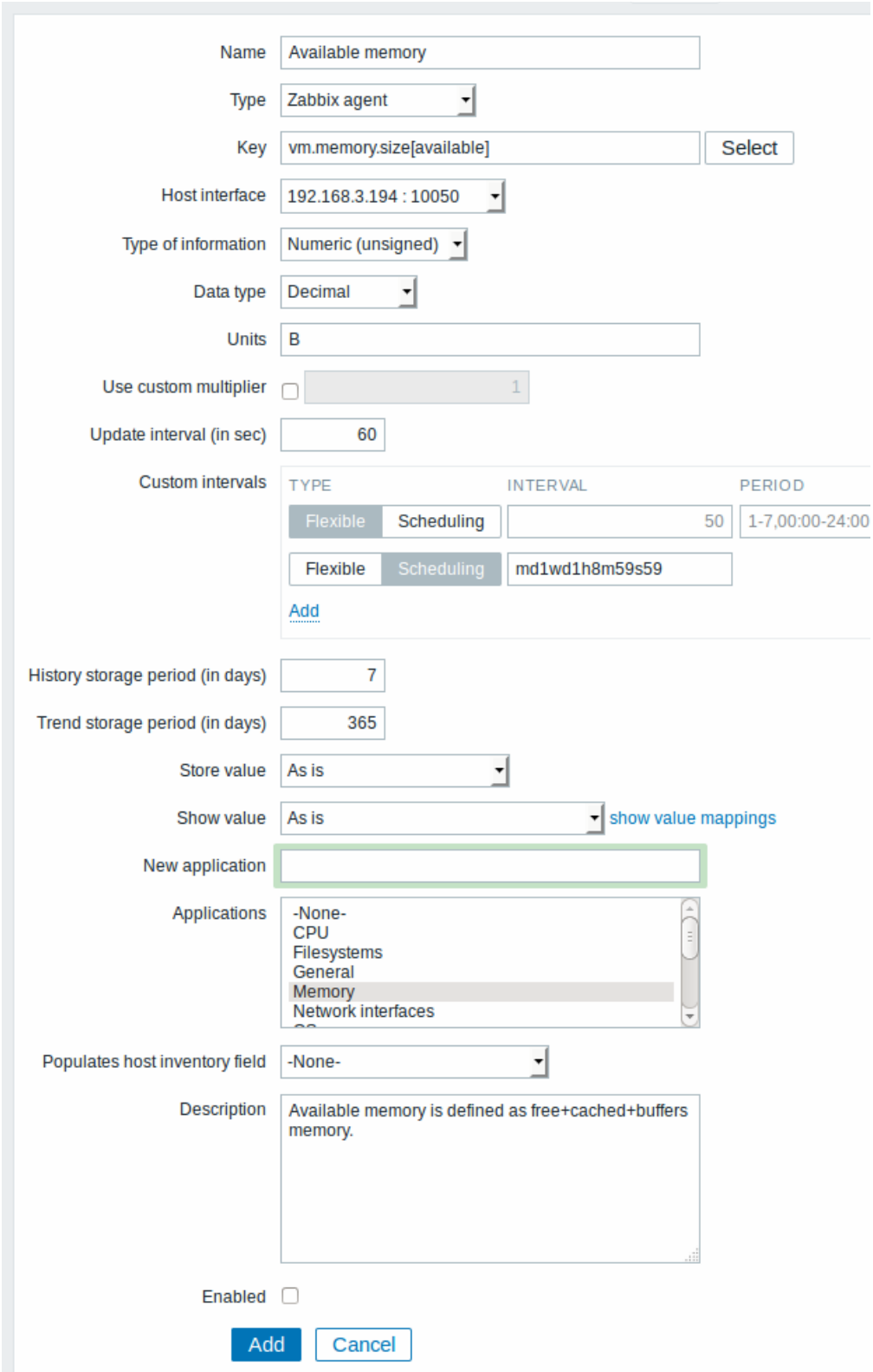



*Figura 44: Creazione item*

In questa circostanza ci sono diversi parametri che devono essere settati affinché l'item possa funzionare:

- *Name*: rappresenta il nome dell'item;
- *Type*: serve per specificare il tipo di item. Questi possono essere:
  - Zabbix agent / Zabbix agent (active): vengono utilizzati gli agent di Zabbix per raccogliere i dati. I primi permettono di effettuare check passivi, i secondi check attivi;
  - Simple check: sono semplici controlli che vengono normalmente utilizzati per fare check remoti agentless di servizi. In questo caso, non sono necessari Zabbix agent in quanto il server o il proxy sarà responsabile dell'esecuzione del check. Tali controlli vengono forniti da Zabbix stesso;
  - SNMP agent: vengono utilizzati quando si vogliono monitorare dispositivi attraverso SNMP come stampanti, switch di rete, router o UPS che, solitamente, sono SNMP-enabled e su cui non sarebbe pratico installare determinati sistemi o Zabbix agent. Check basati su SNMP vengono effettuati solo sul protocollo UDP;
  - SNMP trap: questa soluzione è l'opposta della precedente. In questo caso i dati vengono inviati da un dispositivo SNMP-enabled e memorizzati o "trapped" da Zabbix. Solitamente, i trap vengono inviati sotto condizioni di cambiamento e l'agent si connette al server sulla porta 162 (in contrapposizione alla porta 161 sul lato agent che viene utilizzata per le query). Il loro utilizzo consente il rilevamento di problemi che si verificano tra intervalli di query;
  - Zabbix internal: consente di effettuare il controllo di processi interni a Zabbix; in altre parole permettono il monitoraggio di quello che succede sul server o sul proxy;
  - Zabbix trapper: permette di prendere dati in ingresso piuttosto che fare delle query. È utile per tutti i dati che si potrebbero volere inserire in Zabbix;
  - Zabbix aggregate: tale soluzione permette al server di raccogliere informazioni aggregate di item differenti per fare query al database in modo diretto. Questi non richiedono la presenza di agent attivi sul dispositivo che si vuole monitorare;
  - External check: il server esegue controlli esterni come script in shell o binary. Anche in questo caso non sono richiesti agenti attivi sul dispositivo che si vuole monitorare;

- Database monitor: viene utilizzato ODBC che è un linguaggio di programmazione C con API middleware per l'accesso ai sistemi di gestione dei database (DBMS). Il concetto ODBC è stato sviluppato da Microsoft e in seguito portato su altre piattaforme. Zabbix può interrogare qualsiasi database che è supportato da ODBC; per farlo, Zabbix non si connette direttamente ai database, ma utilizza l'interfaccia ODBC e driver installati al suo interno. Questa funzione consente il monitoraggio più efficiente dei database per qualsiasi tipo di scopo;
- IPMI agent: consente di monitorare la "salute" e la disponibilità di dispositivi Intelligent Platform Management Interface (IPMI). IPMI è un'interfaccia standardizzata che permette di monitorare parametri come temperatura, voltaggio, velocità delle ventole e molto altro;
- SSH/Telnet agent: consentono di monitorare i dispositivi di interesse attraverso SSH oppure telnet. In questo caso, non sono richiesti agenti attivi sui dispositivi, ma sarà necessario conoscere username e password per poter andare avanti;
- JMX agent: consente di monitorare applicazioni Java. Il monitoraggio JMX ha un supporto nativo in Zabbix in forma demone che prende il nome di Zabbix Java gateway;
- Calculated: permette di realizzare delle espressioni sulla base di altri item. I risultati vengono memorizzati dal server all'interno del database come per un qualunque altro item.

  Una cosa importante da dire è che la scelta ricade in base a ciò che si vuole fare e come si vuole realizzare il proprio servizio. La Figura 44 mostra una schermata in cui il type è settato come Zabbix agent, ma nel caso il tipo viene modificato anche la schermata cambierà in quanto ognuno avrà i propri parametri che devono essere configurati affinché l'item possa funzionare;

- *Key*: rappresenta una chiave unica all'interno del singolo host che specifica l'operazione che si vuole eseguire. Ogni key esegue un'operazione differente e varia in base al type selezionato in precedenza;
- *Host interface*: permette di selezionare l'interfaccia host;

- *Type of information*: indica il tipo di dato memorizzato nel database in seguito alla conversione. Questi possono essere: Numeric (unsigned), Numeric(float), Character, Log or Text;
- *Data type*: tale parametro è settabile solo se il campo precedente è un Numeric (unsigned). Di conseguenza, il tipo di dato può essere Boolean, Octal, Decimal o Hexadecimal;
- *Units*: se tale campo viene settato, Zabbix effettuerà un'ulteriore elaborazione del valore ricevuto e lo mostrerà con il suffisso inserito in unit. Ad esempio, se inseriamo B (Byte) o Bps (Byte Per Second) Zabbix automaticamente dividerà per 1024 così che 1536 viene mostrato come 1.5 KB;
- *Use custom multiplier*: se abilitata questa opzione, tutti i valori ricevuti verranno moltiplicati per l'intero o il floating-point inserito in questo campo;
- *Update interval*: rappresenta l'intervallo di tempo dopo il quale viene aggiornato il dato;
- *Custom intervals*: consente di creare regole personalizzate per il controllo di un item: *Flexible* rappresenta un'eccezione all'update interval, infatti, si possono avere intervalli con frequenza differente; *Scheduling* permette di creare una propria programmazione dei check;
- *History storage period*: indica il periodo, in giorni, di permanenza della history nel database. Terminati questi giorni, i dati vengono eliminati automaticamente;
- *Trend storage period*: indica il periodo, in giorni, di permanenza della history aggregata nel database. Terminati questi giorni, i dati vengono eliminati automaticamente;
- *Store value*: indica il modo in cui i dati devono essere memorizzati;
- *Show value*: applica il valore di mapping all'item. Tale valore non cambia il dato ricevuto, infatti, è solo per la visualizzazione del dato stesso;
- *New application*: permette di creare una nuova applicazione a cui l'item apparterrà. L'applicazione la possiamo definire come un gruppo di item accomunati da una stessa caratteristica; ad esempio, item che permettono di calcolare dati differenti relativi alla CPU potrebbero essere raggruppati in

un'applicazione chiamata CPU Check. Quindi, non avremo altro che una classificazione degli item che facilitano la ricerca di ciò che è di interesse;

- *Applications*: collega l'oggetto ad una o più applicazioni esistenti;
- *Populates host inventory field*: permette di selezionare un campo dell'inventory che il valore dell'item andrà a popolare. Questo funzionerà solo se l'opzione di popolazione automatica dell'inventory è abilitata;
- *Description*: semplice descrizione dell'item;
- *Enables*: permette di abilitare o disabilitare l'item che si sta creando.

Per terminare non rimane altro che cliccare su *Add* in modo da salvare l'item realizzato e renderlo disponibile per i dispositivi che si vogliono monitorare.

### 3.3.3 – Trigger

I trigger sono delle espressioni logiche che permettono di valutare i dati raccolti dagli item a rappresentare lo stato corrente del sistema. Mentre gli item si occupano solo del raccoglimento dei dati, i trigger li analizzano attraverso delle espressioni che consentono di definire una soglia per cui un dato può essere considerato accettabile. Un trigger può assumere due soli stati:

- OK: questo rappresenta il caso normale in cui tutto funziona in modo corretto;
- PROBLEM: questo indica che è stata superata una certa soglia e quindi il dato non è più accettabile.

Lo stato del trigger viene ricalcolato dal server ogni volta che riceve un nuovo valore che fa parte dell'espressione.

Per la configurazione del trigger, accediamo al frontend di Zabbix, andiamo su *Configuration*, *Hosts* e clicchiamo sulla sezione *Triggers* relativo all'host di interesse. A questo punto, in alto a destra, clicchiamo su *Create trigger* ottenendo la seguente schermata:

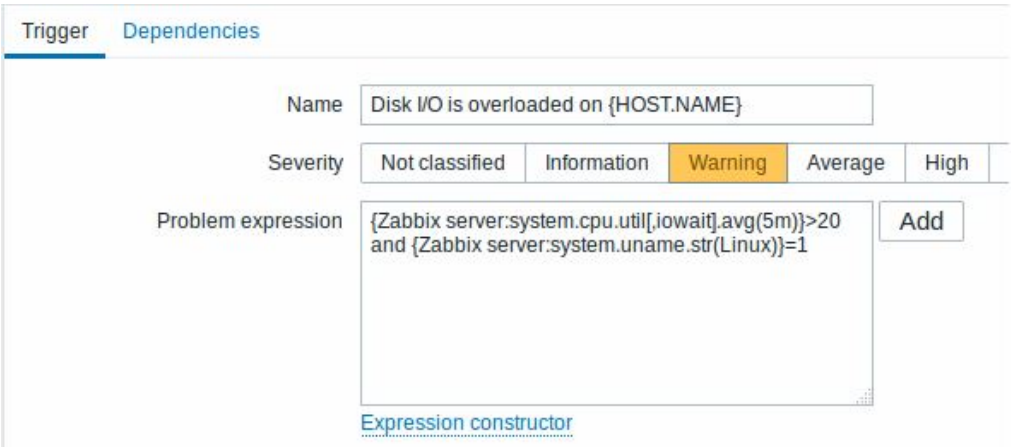



*Figura 45: Creazione trigger parte 1*

In questa prima parte per la creazione del trigger, i parametri da configurare sono:

- *Name*: indica il nome che vogliamo dare al trigger. Può contenere le macros come {HOST.NAME}, {HOST.CONN}, {HOST.IP}, ecc…;
- *Severity*: rappresenta l'importanza del trigger. Osservando la Figura 33 è possibile sapere quali sono i tipi di severity supportati. Queste sono utili per una rappresentazione grafica del trigger (un colore per ogni severity), per allarmi audio (un suono per ogni severity) e per user media, cioè canali di notifica differenti per ogni severity. Una caratteristica importante delle severity è che possono essere create e personalizzate con cambiamento di nome e/o di colore;
- *Problem expression*: espressione logica utilizzata per definire la condizione di un problema. Le espressioni sono molto flessibili e permettono di creare delle condizioni molto complesse. Un'espressione è definita nel seguente modo:

  *{server1:system.cpu.load[all,avg1].last()}>5*

  questa indica che se il valore più recente fornito dalla funzione system.cpu.load con determinati parametri di ingresso è maggiore di 5, allora in questo caso il trigger cambia stato passando da OK a PROBLEM e manifestando la severity impostata. Dalla Figura 45 è possibile notare che si può abilitare anche l'*Expression constructor* costruttore che aiuta l'utente a creare delle espressioni senza doverle scrivere. È possibile scrivere delle espressioni molto complesse e dettagliate in quanto Zabbix fornisce molte funzioni e operatori. Per approfondire il tutto è possibile consultare la documentazione da cui ho effettuato i miei studi [4].

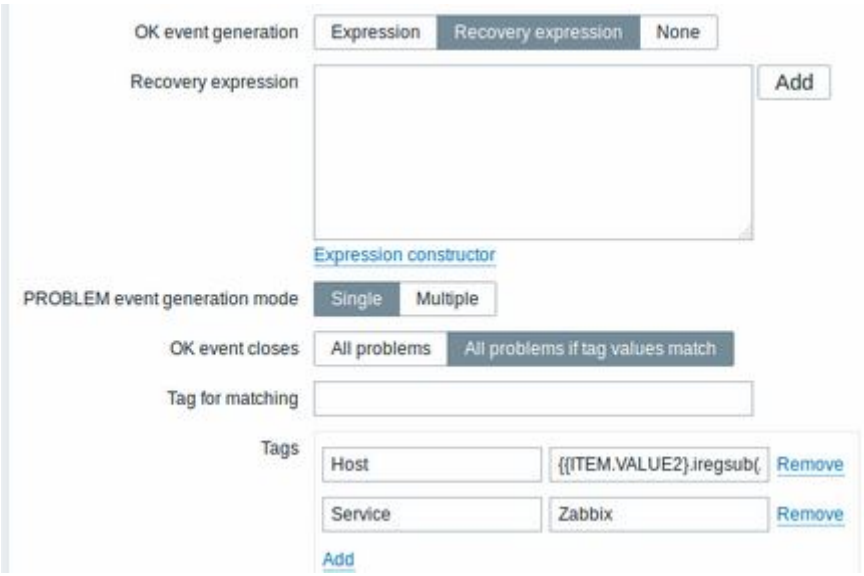



*Figura 46: Creazione trigger parte 2*

Nella seconda parte per la creazione del trigger troviamo:

- *OK event generation*: rappresenta le opzioni per la generazione di eventi OK. Si possono avere: *Expression*: eventi OK vengono generati in base alla stessa espressione come eventi problematici; *Recovery expression*: eventi OK vengono generati se l'espressione di recupero è TRUE e l'espressione problema restituisce FALSE; *None*: il trigger non ritornerà mai ad uno stato OK da solo;
- *Recovery expression*: viene definita l'espressione logica utilizzata per definire le condizioni quando il problema viene risolto. Anche in questo caso, abbiamo un costruttore di espressioni che aiuta l'utente a crearle;
- *PROBLEM event generation mode*: rappresenta la modalità per la generazione di eventi relativi ad un problema: *Single*: viene generato un singolo evento quando un trigger passa dallo stato OK a PROBLEM per la prima volta; *Multiple*: viene generato un evento al momento della valutazione di ogni stato PROBLEM del trigger;
- *OK event closes*: si seleziona *All problems* se l'evento OK risolve tutti i problemi del trigger; *All problems if tag values match* se l'evento OK risolve solo quei problemi che fanno match con i valori dell'event tag;
- *Tag for matching*: questo campo viene mostrato solo se selezionato *All problems if tag values match* nel campo precedente e serve per specificare il nome dell'event tag da utilizzare per effettuare il matching;
- *Tags*: rappresentano tag personalizzati per marcare eventi trigger. Sono costituiti da una coppia nome-valore;

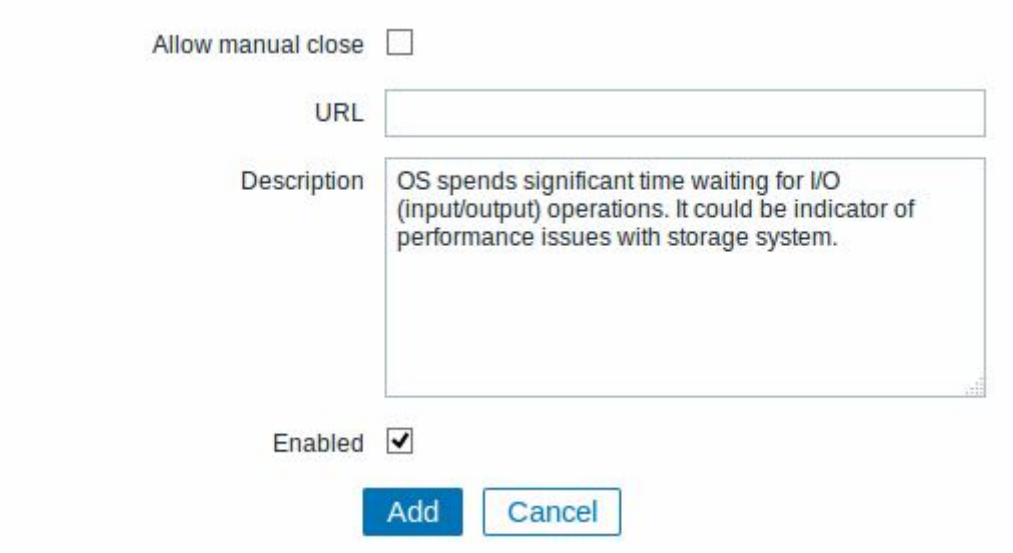



*Figura 47: Creazione trigger parte 3*

Nella terza e ultima parte abbiamo:

- *Allow manual close*: consente di chiudere manualmente l'evento relativo al problema generato dal trigger.
- *URL*: se non è vuoto, tale campo rappresenta il collegamento al nome del trigger;
- *Description*: rappresenta una breve descrizione del trigger;
- *Enabled*: consente di abilitare o disabilitare il trigger appena creato.

Come possiamo notare dalla Figura 45, oltre alla sezione Trigger, ne abbiamo un'altra che prende il nome di Dependencies. A volte la disponibilità di un dispositivo può dipendere da un altro, ad esempio, un server potrebbe risultare irraggiungibile a causa dei router, ad esso collegati, che sono giù. Configurando il trigger con dipendenze è possibile inviare notifiche anche in situazioni come queste. Mentre Zabbix non supporta le dipendenze dirette fra gli host, queste potrebbero essere definite attraverso altri e più flessibili metodi, ovvero, dipendenze dei trigger.

Un trigger può avere uno o più trigger da cui dipende. Detto questo e riconsiderando l'esempio di prima, possiamo dire che è possibile configurare un trigger del server e settare la sua dipendenza sul rispettivo trigger del router. Con questa dipendenza il trigger del server non cambierà stato finché il trigger da cui dipende (quello del router) è sullo stato PROBLEM, di conseguenza nessuna azione verrà effettuata e nessuna notifica inviata. Se router e server sono entrambi giù e tra loro c'è dipendenza, Zabbix non eseguirà azioni per il trigger dipendente.

Settate tutte queste opzioni, come sempre, andiamo a cliccare sul tasto *Add* in modo da rendere disponibile il trigger.

### 3.3.4 – Notification

Una volta che item e trigger sono stati configurati, al cambiamento di stato dei trigger vengono generati degli eventi, pertanto è necessario dover andare a considerare possibili azioni che possono essere fatte. Per prima cosa, è chiaro che non si vuole passare tutto il tempo a fissare i trigger o la lista degli eventi; infatti, sarebbe meglio ricevere una notifica quando si verifica qualcosa di significante; inoltre, altro aspetto desiderabile è che tutte le persone di interesse vengano notificate del problema.

Zabbix consente l'invio delle notifiche, di conseguenza, bisogna definire chi e quando deve essere notificato al verificarsi di un determinato evento.

Per poter inviare e ricevere notifiche da Zabbix bisogna:

- **definire i media**. I media sono canali di distribuzione utilizzati per l'invio di notifiche e allarmi. Esistono diversi tipi di media che possono essere configurati in Zabbix:
  - E-mail: per creare questo tipo di canale, bisogna configurare l'email come media e assegnare indirizzi specifici agli utenti. Per la configurazione, bisogna accedere al frontend di Zabbix, andare su *Administration*, *Media types* e cliccare su *Create media type*. Arrivati a questo punto si aprirà una schermata in cui ci saranno dei parametri da settare:
    - *Name*: nome del media;
    - *Type*: tipo del media. In questo caso selezioniamo Email;
    - *SMTP server*: inserire il server SMTP per la gestione dei messaggi in uscita;
    - *SMTP server port*: inserire la porta del server SMTP per la gestione dei messaggi in uscita;
    - *SMTP helo*: solitamente rappresenta il nome del dominio;
    - *SMTP email*: viene inserito l'indirizzo email utilizzato come indirizzo mittente per i messaggi inviati;
    - *Connection security*: permette di selezionare il livello di sicurezza della connessione;

- *SSL verify peer*: abilita la verifica del certificato SSL del server SMTP;
- *SSL verify host*: verifica se il campo *Common Name* o *Subject Alternate Name* del certificato del server SMTP fa match;
- *Authentication*: seleziona il livello di autenticazione;
- *Username*: specifica il nome utente;
- *Password*: specifica la password;
- *Enabled*: consente di abilitare o meno questo media.

◦ SMS: Zabbix supporta l'invio di messaggi SMS utilizzando un modem GSM seriale connesso alla porta seriale del server Zabbix. Per creare questo tipo di canale, bisogna configurare SMS come media e assegnare i rispettivi numeri di telefono per gli utenti. Per la configurazione, bisogna accedere al frontend di Zabbix, andare su *Administration*, *Media types* e cliccare su *Create media type*. Arrivati a questo punto si aprirà una schermata in cui ci saranno dei parametri da settare:

- *Name*: nome del media;
- *Type*: tipo del media, in questo caso SMS;
- *GSM Modem*: nome seriale del dispositivo del modem GSM;

◦ Jabber: Zabbix supporta l'invio di messaggi Jabber. Quando vengono inviate le notifiche, Zabbix, per primo, prova a cercare il Jabber SRV record e se fallisce utilizza un address record per quel dominio. Tra i diversi Jabber SRV record quello che ha la maggiore priorità e maggior peso viene scelto; se fallisce non vengono provati altri record. Per la configurazione, bisogna accedere al frontend di Zabbix, andare su *Administration*, *Media types* e cliccare su *Create media type*. Arrivati a questo punto si aprirà una schermata in cui ci saranno dei parametri da settare:

- *Name*: nome del media;
- *Type*: tipo del media, in questo caso Jabber;
- *Jabber identifier*: identificatore Jabber;
- *Password*: password corrispondente all'identificatore.

◦ Ez Texting: si può utilizzare un partner di Zabbix per l'invio dei messaggi. Per creare questo tipo di canale, bisogna configurare Ez Texting come

media e assegnare l'identificazione del destinatario per gli utenti. Per la configurazione, bisogna accedere al frontend di Zabbix, andare su *Administration*, *Media types* e cliccare su *Create media type*. Arrivati a questo punto si aprirà una schermata in cui ci saranno dei parametri da settare:

- *Name*: nome del media;
- *Type*: tipo del media, in questo caso Ez Texting;
- *Username*: rappresenta il nome utente Ez Texting;
- *Password*: specifica la password corrispondente all'utente;
- *Message text limit*: specifica il limite della lunghezza del messaggio;

- Script di allarme personalizzati: consente di creare gli script che permettono di gestire le notifiche come si desidera. Questi verranno eseguiti dal server a patto che siano inseriti nella directory specificata nella variabile *AlertScriptsPath* del file di configurazione del server stesso. Per la configurazione, bisogna accedere al frontend di Zabbix, andare su *Administration*, *Media types* e cliccare su *Create media type*. Arrivati a questo punto si aprirà una schermata in cui ci saranno dei parametri da settare:

  - *Name*: nome del media;
  - *Type*: tipo del media, in questo caso Script;
  - *Script name*: nome dello script;
  - *Script parameters*: parametri della command-line dello script. In questo caso, si possono utilizzare anche le macro.

  Una volta che il media è stato configurato bisogna andare su *Administration* e *Users*. Cliccare sull'utente desiderato e andare nella sezione *Media*. Arrivati a questo punto bisogna settare i parametri:

  - *Type*: rappresenta il tipo di media per l'invio dei messaggi; Email, SMS, Jabber o Ez Texting;
  - *Send to*: specifica l'indirizzo Email, numero di telefono o destinatario a cui mandare i messaggi;

- *When active*: rappresenta l'intervallo di tempo in cui i messaggi devono essere inviati. Ad esempio 1-7,00:00-24:00 indica che i messaggi possono essere inviati tutti i giorni in qualsiasi orario;
- *Use if severity*: permette di selezionare la severity per cui si desidera ricevere messaggi;

  *Enabled*: permette di abilitare o disabilitare il sistema di notifica;

- **configurare un'azione** che invia un messaggio ad uno dei media definiti. Le azioni consistono in condizioni e operazioni; in sostanza, quando le condizioni sono soddisfatte, allora le operazioni vengono eseguite. Se si vuole eseguire un'operazione in risposta ad un evento è necessario configurare un'azione. Le azioni possono essere definite in risposta ad eventi di tutti i tipi supportati:
  - Trigger event: quando lo stato del trigger cambia da OK a PROBLEM e viceversa;
  - Discovery event: quando viene effettuato il discovery della rete;
  - Auto registration event: quando nuovi active agent si auto registrano;
  - Internal event: quando gli item non sono più supportati o i trigger passano allo stato di UNKNOWN.

  Per configurare un'azione bisogna andare su *Configuration*, *Actions* e cliccare su *Create action*. In seguito a questo, si avrà una schermata in cui sono presenti tre sezioni. La prima è l'Action e permette di settare i seguenti valori:
  - *Name*: nome dell'azione che deve essere univoco;
  - *Type of calculation*: rappresenta il tipo di calcolo che si desidera: And, Or, And/Or e Custom expression. Quest'ultima è una formula user-defined per la valutazione delle condizioni;
  - *Conditions*: lista delle condizioni;
  - *New condition*: permette di creare e aggiungere una nuova condizione nella lista;
  - *Enabled*: permette di abilitare o disabilitare l'azione che si sta creando.

  Nella sezione successiva, Operations, è possibile configurare le operazioni da eseguire per tutti gli eventi. Le opzioni da settare sono:

- *Default operation step duration*: rappresenta la durata di un'operazione. Ad esempio se settiamo 3600 secondi significa che se viene fatta l'operazione, dovrà passare un'ora prima del prossimo step;
- *Default subject*: rappresenta il soggetto di default del messaggio per le notifiche;
- *Default message*: rappresenta il messaggio per le notifiche;
- *Pause operations while in maintenance*: consente di ritardare le operazioni per la durata del periodo di manutenzione;
- *Operations*: in questo campo vengono mostrare le operazioni con i loro dettagli: *Steps* rappresenta gli step di escalation a cui l'operazione viene assegnata; *Details* specifica il tipo di operazione e i suoi destinatari; *Start in* indica quando l'operazione deve essere eseguita in seguito ad un evento; *Duration* rappresenta la durata dello step dell'operazione; *Action* permette di modificare o rimuovere l'operazione;
- *Operation details*: questo blocco viene utilizzato per configurare i dettagli dell'operazione. In questo caso le opzioni da settare sono:
  - *Steps*: permette di selezionare gli step da assegnare all'operazione in una schedulazione di escalation. In questo caso si possono settare due valori: il primo indica lo step di partenza, il secondo quello di arrivo;
  - *Step duration*: rappresenta la durato degli steps;
  - *Operation type*: permette di selezionare il tipo di operazione. Solo due sono disponibili:
    - *Send Message*: permette di mandare il messaggio all'utente in seguito ad un evento. Se si seleziona questo tipo di operazione i parametri da settare sono:
      - *Send to user groups*: seleziona il gruppo di utenti a cui inviare il messaggio;
      - *Send to users*: seleziona gli utenti a cui inviare il messaggio;
      - *Send only to*: consente di mandare il messaggio a tutti i tipi di media definiti o sono a quelli selezionati;

- *Default message*: se selezionato, il messaggio di default viene utilizzato;
- *Subject*: rappresenta il soggetto del messaggio;
- *Message*: rappresenta il messaggio;

- *Remote Command*: consente di eseguire automaticamente dei comandi sul dispositivo monitorato se soddisfatte le condizioni. I parametri da settare, in questo caso, sono:
  - *Target list*: rappresenta la lista dei target su cui eseguire il comando. Può essere *Current Host*: il comando viene eseguito sull'host del trigger che ha scatenato l'evento problema; *Host*: il comando viene eseguito su host selezionati; *Host group*: il comando viene eseguito su tutti gli host appartenenti al gruppo specificato;
  - *Type*: specifica il tipo di comando: IPMI, Custom script, SSH, Telnet, Global script;
  - *Execute on*: esegue il comando sul server o sull'agent Zabbix;
  - *Commands*: specifica il comando che si vuole eseguire;

- *Conditions*: indica le condizioni per cui l'operazione deve essere eseguita.

Nell'ultima sezione, Recovery Operations, vengono configurate le operazioni che consentono di notificare gli utenti quando i problemi vengono risolti.

Ultimo aspetto importante relativo al sistema di notifica è rappresentato dalle escalation. Con queste è possibile creare scenari personalizzati per l'invio delle notifiche o esecuzione di comandi remoti. In termini pratici questo significa che gli utenti possono essere informati immediatamente sui nuovi problemi, le notifiche possono essere ripetute finché il problema non viene risolto, l'invio di una notifica può essere ritardato ed, infine, i comandi remoti possono essere eseguiti immediatamente o quando un problema non è risolto per un lungo periodo. La creazione delle escalation può essere fatta durante la configurazione delle operazioni.

## 3.4 – Confronto con Nagios e Pandora FMS

Oggigiorno il sistema di monitoraggio più utilizzato è Nagios, in quanto risulta essere anche il più "vecchio". Zabbix, che è molto più giovane, tende a riprendere diversi aspetti di Nagios, mentre Pandora FMS risulta essere giovanissimo; di conseguenza il confronto viene fatto su tre sistemi di monitoraggio che hanno età differenti, ma con vantaggi e svantaggi che rendono sempre più difficile scegliere il sistema da adottare per monitorare dispositivi, rete, applicazioni e molto altro.

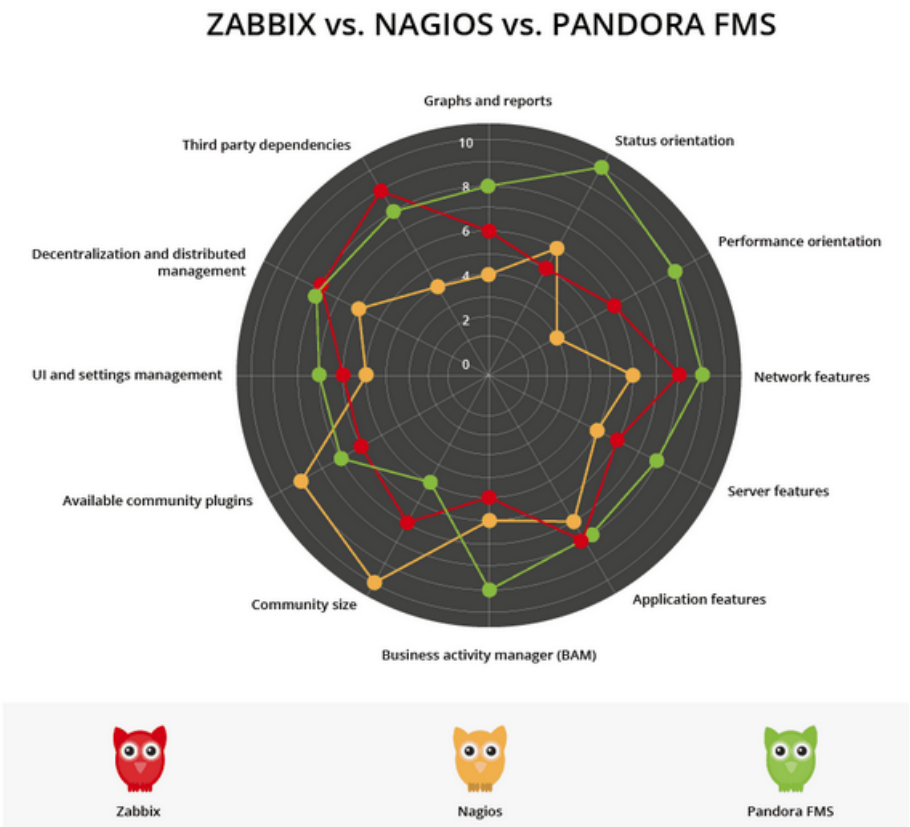



*Figura 48: Confronto fra Zabbix - Nagios - Pandora FMS*

Nagios viene considerato, da coloro che hanno speso molto tempo nel mondo IT, come il sistema di monitoraggio Open Source standard. Questo potremmo considerarlo anche vero, in quanto è stato il primo vero sistema a lavorare in modo corretto; Nagios, infatti, è stato introdotto nel 1999 e prima vi erano solo soluzioni amatoriali e specifiche che non si avvicinavano nemmeno alle innovazioni introdotte da Nagios.

Zabbix viene introdotto nel 2001, quindi più giovane rispetto a Nagios e ancora in pieno sviluppo. Ha una caratteristica molto importante, ovvero, avere una visione molto olistica sul monitoraggio, infatti, ricopre le performance di un dispositivo e non solo gli stati, caratteristica significativa che manca in Nagios. Zabbix oltre ad avere

un sistema web-based che consente la gestione centralizzata, non presenta quei fastidiosi file di configurazione di cui Nagios non può farne a meno.

Pandora FMS è nato nel 2004, quindi rispetto ai precedenti risulta essere il più giovane. La sua caratteristica principale sta nel fatto che è più di un sistema di monitoraggio IT, infatti, è un framework che consente qualsiasi cosa: dal monitoraggio di infrastrutture (reti e server) a monitoraggio di performance e applicazioni e anche monitoraggio di business transazionale. Pandora FMS presenta un sistema di gestione centrale e si basa su un database relazionale SQL.

La prima grande differenza che si può considerare fra i sistemi di monitoraggio citati è la gestione e l'installazione degli stessi. Nagios, essendo molto antico, presenta ancora un complesso intreccio di file di testo, script e procedure manuali che richiedono anche l'utilizzo di terze parti per consentirne l'installazione. Questo rappresenta (o rappresentava) un vantaggio in quanto non richiedeva la memorizzazione di informazioni nel database e quindi meno utilizzo di risorse computazionali. Col passare degli anni, però, il collo di bottiglia non è più rappresentato dall'hardware, ma dalla capacità di gestire efficacemente la configurazione, situazione completamente opposta per quanto riguarda Nagios.

Zabbix e Pandora FMS sono esattamente l'opposto di Nagios da questo punto di vista; infatti, sono soluzioni solide con un'architettura complessa e modulare che è cresciuta col passare degli anni fornendo sempre più un design diretto. Dal punto di vista della configurazione, non richiedono, come Nagios, particolari interventi manuali o modifiche di file di configurazioni, ma tendono ad essere più lineari e di facile comprensione in modo che chiunque possa gestire ed effettuare un'installazione.

Zabbix e Nagios necessitano l'installazione di ulteriori plugin per offrire una serie di funzionalità complete, ma nel complesso, così come sono, consentono di effettuare un ottimo monitoraggio. È chiaro che ogni volta bisogna considerare ciò che si desidera fare e soprattutto come farlo.

Zabbix non presenta una libreria di plugin ufficiale per la community, mentre Nagios ha un'enorme libreria, però con bassa manutenzione dal momento che tutti i plugin sono al 100% Open Source e di conseguenza non c'è nessuna società che se ne prende cura. Pandora FMS, invece, ha una libreria molto più piccola, ma gestita da una società e, trascurando il fatto che alcuni plugin sono enterprise (sotto licenza e a pagamento), il tutto è concentrato verso prodotti reali e non esclusivamente verso la

tecnologia aperta. Pandora FMS, nella sua versione Open Source, ha una collezione di plugin di default e di moduli che sono “plug and play, ready to use”.

Zabbix ha un sistema potente di template e definizione di trigger basate su espressioni regolari; nonostante tutto questo sia particolarmente “potente” dal punto di vista del monitoraggio, questo risulta essere anche complesso nell’utilizzo, nel senso che per utilizzare i trigger è necessario avere un’ottima conoscenza delle espressioni regolari. In Nagios non vi è nulla di tutto questo, in quanto veniva restituito direttamente lo stato del dispositivo, mentre in Pandora FMS ci sono screen e wizard nell’interfaccia web che sono maggiormente user-friendly.

Principalmente, queste sono le differenze che si dovrebbero considerare, ovvero, installazione e configurazione del sistema di monitoraggio e presenza, se necessaria, di plugin o terze parti che ne garantiscono il corretto funzionamento e l’efficienza richiesta. È chiaro che andando sempre più in profondità si possono scoprire altri aspetti che possono essere simili o diversi, come ad esempio utilizzo o meno di agent (basti pensare che Nagios non presenta agent, quindi il tutto viene fatto attraverso script definiti dall’utente, mentre Zabbix e Pandora FMS consentono il monitoraggio sia con agent, ma anche in modalità agentless), oppure la visualizzazione dei grafici che possono essere più o meno dettagliati, ma anche real-time o ritardati.

## 3.5 – Requisiti di sistema ed installazione

Per poter effettuare l'installazione di Zabbix e dei suoi componenti è necessario soddisfare dei requisiti senza i quali non è possibile procedere. Quelli che andrò a riportare qui di seguito si riferiscono alla versione 3.2 di Zabbix (ultima versione stabile in quanto la 3.4 ancora non lo è), ma se si è interessati ad altre versioni o a maggiori dettagli è possibile fare riferimento alla documentazione ufficiale sulla quale sono stati effettuati gli studi [5]:

- **Piattaforme supportate**: a causa di requisiti di sicurezza e la natura misson-critical del server di monitoraggio, Linux/Unix risulta essere l'unico sistema operativo che può sempre fornire le performance necessarie e garantire la tolleranza ai guasti; ciononostante Zabbix è stato anche testato su IBM AIX, FreeBSD, NetBSD, OpenBSD, HP-UX, Mac OS X, Solaris, Windows a partire dalla versione 2000;
- **Hardware**: Zabbix richiede sia la memoria fisica che quella libera su disco. Un buon punto di partenza può essere rappresentato dall'utilizzare 128 MB di memoria fisica e 256 MB di spazio libero su disco. Tuttavia, le scelte si basano sul numero di host e parametri che si vogliono monitorare. Bisogna ricordare che ogni processo demone Zabbix richiede alcune connessioni al server database; in questo caso la quantità di memoria allocata per la connessione dipende dalla configurazione dell'engine del database. È chiaro che più memoria fisica si possiede e più velocemente lavorerà il database e quindi Zabbix stesso.

  Altro aspetto importante è rappresentato dalle risorse CPU che dipendono dal numero dei parametri monitorati e dall'engine del database scelto. Per un corretto funzionamento è richiesta una significante quantità di CPU;
- **Software**: in questo ambito ci sono diversi componenti che Zabbix richiede. Per quanto riguarda il database, è possibile scegliere fra:
  - MySQL dalla versione 5.0.3 in poi;
  - Oracle dalla versione 10g in poi;
  - PostgreSQL dalla versione 8.1 in poi;
  - SQLite dalla versione 3.3.5 in poi;
  - IBM DB2 dalla versione.

Per quanto riguarda l'esecuzione del frontend di Zabbix sono necessari:

- Apache dalla versione 1.3.12 in poi;
- PHP dalla versione 5.4.0 in poi con relative estensioni: php-gd, php-bcmath, php-ctype, php-libXML che comprende anche mlreader e xmlwriter, php-session, php-net-socket, php-mbstring, php-gettext, php-ldap, ed, infine, in base al database scelto ibm_db2/mysqli/oci8/pgsql/sqlite3;

Altri requisiti software sono: OpenIPMI, libssh2, fping, libcurl, libiksemel, libxml2 e net-snmp.

Infine, affinché la comunicazione possa avvenire, è necessario aprire le seguenti porte:

- ssh (22 TCP);
- zabbix agent (10050 TCP) e zabbix trapper (10051 TCP);
- HTTP (80 TCP) e HTTPS (443 TCP);
- SNMP (162 UDP);
- NTP (53 UDP).

Per poter fare questo utilizziamo le seguenti linee di comando:

- *firewall-cmd --zone=public --permanent --add-port=22/tcp*
- *firewall-cmd --zone=public --permanent --add-port=10050/tcp*
- *firewall-cmd --zone=public --permanent --add-port=10051/tcp*
- *firewall-cmd --zone=public --permanent --add-port=80/tcp*
- *firewall-cmd --zone=public --permanent --add-port=443/tcp*
- *firewall-cmd --zone=public --permanent --add-port=162/udp*
- *firewall-cmd --zone=public --permanent --add-port=53/udp.*

Il firewall-cmd è la riga di comando del demone firewalld che fornisce l'interfaccia per la gestione della configurazione. Il demone firewalld è un tool per la gestione dei firewall per sistemi operativi Linux. Fornisce funzioni di firewall agendo come un frontend per il sistema di filtraggio dei pacchetti fornito dal Linux kernel. Attraverso questo comando, quindi, siamo in grado di aprire quelle porte di ascolto necessarie per la ricezione dei pacchetti.

Una volta che tutti questi requisiti sono soddisfatti, allora è possibile proseguire con l'installazione di Zabbix server, agent e proxy (se necessario). Anche in questo caso, verrà spiegata l'installazione per Red Hat Enterprise Linux/CentOS versione 7 in quanto è su questo che si è basato il mio lavoro. Per altri sistemi operativi o altre versioni è possibile sempre fare riferimento alla documentazione ufficiale [5]. Partiamo con il server, in quanto risulta essere quello più complesso. La prima cosa da fare è installare il pacchetto di configurazione:

- *rpm -ivh http://repo.zabbix.com/zabbix/3.2/rhel/7/x86_64/zabbix-release-3.2-1.el7.noarch.rpm*

Fatto questo è possibile installare Zabbix server e il web frontend con il database corrispondente (nel nostro caso il database è MySQL):

- *yum install zabbix-server-mysql zabbix-web-mysql*

Terminata l'installazione, bisogna crearsi il database zabbix e corrispondente utente. Per prima cosa entriamo in MySQL:

- *mysql -u root -p*

da questo comando ci verrà richiesta la password di root per potervi accedere. Una volta entrati in MySQL, andiamo a creare il database:

- *create database zabbix character set utf8 collate utf8_bin;*

successivamente, andiamo a creare l'utente zabbix fornendogli tutti i privilegi relativamente al database creato:

- *grant all privileges on zabbix.* to zabbix@localhost identified by '<password>';*

fatto questo possiamo uscire da MySQL:

- *quit.*

È chiaro che il database creato risulta essere vuoto pertanto bisogna riempirlo affinché il tutto possa funzionare, quindi dobbiamo importare gli schemi iniziali e i dati:

- *zcat /usr/share/doc/zabbix-server-mysql-3.2.*/create.sql.gz | mysql -u zabbix -p zabbix*

dove al posto di * inseriamo la versione del server che abbiamo installato. Per ottenere la versione:

- *rpm -q zabbix-server-mysql.*

Una volta creato e riempito il database zabbix, bisogna effettuare delle modifiche, in particolar modo, bisogna settare, nel file /etc/zabbix/zabbix_server.conf, i seguenti campi:

- *# vi /etc/zabbix/zabbix_server.conf*
- *DBHost=localhost*
- *DBName=zabbix*
- *DBUser=zabbix*
- *DBPassword=<password>.*

Arrivati a questo punto, bisogna andare a modificare anche il file di configurazione Apache per il frontend Zabbix, cioè /etc/httpd/conf.d/zabbix.conf. Molti campi PHP sono già configurati, ma è necessario de-commentare la riga "date.timezone" e settarla nel modo corretto:

- *php_value max_execution_time 300*
- *php_value memory_limit 128M*
- *php_value post_max_size 16M*
- *php_value upload_max_filesize 2M*
- *php_value max_input_time 300*
- *php_value always_populate_raw_post_data -1*
- *# php_value date.timezone Europe/Riga → php_value date.timezone Europe/Rome*.

Ricordando che l'installazione viene fatta per Red Hat Enterprise Linux/CentOS versione 7, bisogna andare a configurare anche SELinux.

Partiamo col dire che Security-Enhanced Linux (SELinux) è un modulo kernel di Linux che fornisce una serie di strumenti per monitorare le politiche di sicurezza del sistema operativo. Il suo compito è quello di cercare di separare l'applicazione delle regole di sicurezza dalla definizione delle regole stesse, riducendo il numero di software incaricati di verificarne il rispetto.

Detto questo, poiché SELinux è forzatamente abilitato nel sistema operativo (andando a disabilitarlo si renderebbe il sistema su cui si lavora assolutamente

insicuro), bisogna abilitare la connessione del frontend di Zabbix con il server Apache:

- *setsebool -P httpd_can_connect_zabbix on.*

Ora bisogna riavviare il server web Apache:

- *systemctl restart httpd*.

Prima di poter avviare Zabbix server è necessario effettuare un'ultima configurazione relativa alla politica di sicurezza del sistema. Per prima cosa eseguiamo questo comando:

- *cat /var/log/audit/audit.log | grep zabbix_server | grep denied | audit2allow -M zabbix_server_setrlimit > zabbix_server_setrlimit.te*

L'audit.log è un file all'interno del quale Linux Audit system, sulla base di regole preconfigurate, va a registrare tutte le informazione sugli eventi che stanno accadendo nel sistema; in particolar modo, Audit system tiene traccia delle informazioni rilevanti per la sicurezza. Queste risultano cruciali per ambienti mission-critical per individuare coloro che violano le politiche di sicurezza e le loro azioni. Audit non fornisce maggiore sicurezza al sistema; piuttosto, può essere utilizzato per scoprire le violazioni delle politiche di sicurezza utilizzati nel sistema. Queste violazioni possono ulteriormente essere prevenuti da misure di sicurezza aggiuntive quali SELinux.

Introdotto questo concetto importante, possiamo dire che si va a verificare quali politiche di sicurezza sono state violate da Zabbix server (*cat /var/log/audit/audit.log | grep zabbix_server | grep denied*). Una volta individuate, eseguiamo il comando audit2allow (se non presente nel sistema operativo, lo si può installare attraverso yum install policycoreutils-python, pacchetto che contiene tale comando). Questa utility scansiona i file di log contenenti i permessi negati dal sistema all'operazione che si sta cercando di eseguire e genera una piccola regola che, se caricata nella politica di sicurezza, rende l'operazione stessa eseguibile. Tuttavia, questa utility genera regole consentite del tipo Type Enforcement (TE), infatti, viene creata la regola *zabbix_server_setrlimit.te* e compilata generando il file zabbix_server_setrlimit.pp.

Arrivati a questo punto eseguiamo:

- *semodule -i zabbix_server_setrlimit.pp*

semodule è un tool per la gestione dei moduli della politica SELinux, infatti, viene utilizzata per installazione, aggiornamenti e cancellazione dei moduli stessi. Detto questo, quindi, semodule -i permette di installare il modulo appena creato e far si che tale politica consenta l'operazione di avvio del Zabbix server.

Finalmente, siamo in grado di avviare il server, attraverso il comando:

- *systemctl start zabbix-server.*

Se siamo interessati all'avvio del server in fase di boot del sistema, allora eseguiamo anche questo comando:

- *systemctl enable zabbix-server*

che fa si che ogni volta che viene avviata la macchina il server risulta essere già attivo.

Per entrare nel frontend di Zabbix, apriamo un qualsiasi browser ed inseriamo nel campo dell'URL:

*http://ip_zabbix_server/zabbix*.

Terminata l'installazione del server, è possibile iniziare a monitorare tutto ciò che risulta di interesse.

Come detto in precedenza, le fonti che trasmettono i dati al Zabbix server possono essere gli agent oppure i proxy. È chiaro che anche questi necessitano di essere installati. Per quanto riguarda l'agent, lo installiamo:

- *yum install zabbix-agent*

modifichiamo il corrispondente file di configurazione, ovvero /etc/zabbix/zabbix_agentd.conf, inserendo nel campo Server l'indirizzo IP del Zabbix server (c'è anche la possibilità di inserire una lista di indirizzi IP, separati dalla virgola, se si hanno più server). In questo caso, l'agent accetterà solo connessioni in ingresso provenienti dagli host elencati in questo campo.

Nel campo ServerActive, invece, inseriamo la coppia IP:port (oppure hostname:port) del/i Zabbix server/s per consentire gli active check.

Fatto questo, andiamo a creare una regola per la politica di sicurezza (esattamente nello stesso modo spiegato nel server):

- *cat /var/log/audit/audit.log | grep zabbix_agentd | grep denied | audit2allow -M zabbix_agent_setrlimit > zabbix_agent_setrlimit.te*

- *semodule -i zabbix_agent_setrlimit.pp*

ed, infine, avviamo l'agent:

- *systemctl start zabbix-agent*.

Se di interesse, è possibile avviare l'agent al boot del sistema, attraverso il comando:

- *systemctl enable zabbix-agent*.

Installato anche l'agent, non ci rimane che installare anche il proxy. È chiaro che le installazioni vanno fatte in base a ciò che si vuole fare e alle proprie necessità, pertanto ciò che interessa realmente è solo l'installazione del Zabbix server.

Detto questo, per poter installare ed utilizzare il proxy, è necessario, come detto in precedenza, crearsi un apposito database in quanto sappiamo che il proxy ne ha uno completamente suo. Per la creazione e il riempimento del database è possibile fare riferimento alla spiegazione dell'installazione e configurazione del Zabbix server.

Una volta creato il database, installiamo il proxy:

- *yum install zabbix-proxy-<database_type>*

con l'ultimo campo che sta ad indicare il tipo di database utilizzato per il proxy. Ad esempio, potremmo scrivere

*yum install zabbix-proxy-mysql*.

Terminata l'installazione, modifichiamo il corrispondente file di configurazione, /etc/zabbix/zabbix_proxy.conf:

- *# vi /etc/zabbix/zabbix_proxy.conf*
- *DBHost=localhost*
- *DBName=zabbix*
- *DBUser=zabbix*
- *DBPassword=<password>*

ed avviamo il nostro proxy in modo da risultare funzionante:

- *systemctl start zabbix-proxy.*

Come sempre, se siamo interessati ad avviarlo al boot del sistema allora:

- *systemctl enable zabbix-proxy.*

## 3.6 – SysAid CMDB

SysAid è un software per la gestione dei servizi IT progettata per automatizzare i processi di una società per il supporto help desk, compresi configurazioni hardware, asset management, licenze software, attività e progetti, gestione dei dispositivi mobili e altro ancora. SysAid è disponibile su piattaforme in-house o Cloud e presenta edizioni Free, Pro, Enterprise, Education e MSP.

Una caratteristica notevole del software è IT Benchmark, uno strumento di misurazione dinamica che SysAid utilizza per tradurre i dati grezzi delle attività in informazioni significative e utili. Tale strumento consente agli amministratori di analizzare e valutare le loro statistiche IT attuali con quelle dei dati precedenti, così come confrontare le singole statistiche con migliaia di altri dipartimenti IT in tutto il mondo.

Altre caratteristiche includono il portale per l'utente finale, una Manager Dashboard Remote Control, Knowledge Base, CMDB (Configuration Management DataBase), strumenti di monitoraggio, password service, il supporto multi-azienda e una procedura guidata per creare report personalizzati.

Di tale software, ciò che interessa maggiormente è il SysAid CMDB, in quanto è uno degli strumenti coinvolti nel lavoro svolto.

SysAid CMDB è il software che aiuta a tracciare gli elementi di configurazione dell'ecosistema IT, a prevedere l'influenza dei cambiamenti per gestire meglio l'evoluzione IT e a mappare le connessioni che legano fra di loro asset, applicazioni, servizi di business, persone e qualsiasi altra cosa che possa essere di interesse aziendale.

Alcune sue caratteristiche sono:

- **importare automaticamente i dati nel CMDB**: non occorre importare manualmente gli elementi di configurazione (CI), in quanto SysAid CMDB è in grado di importare automaticamente asset, software e molto altro attraverso l'uso di altri moduli SysAid come il network rescovery. Gli oggetti possono essere facilmente importati anche attraverso file CSV;
- **rafforzare l'inventario degli oggetti**: SysAid CMDB permette di mappare l'intera struttura IT raccogliendo in un unico inventario gli elementi IT e non IT come CI e le relazioni che li legano. SysAid CMDB agisce anche come archivio centrale delle informazioni necessarie per tracciare le componenti e

per comprendere l'impatto che le modifiche su un dato CI hanno sui CI ad esso associati. Il CMDB permette di aggiungere un numero illimitato di CI, di classificarli per tipo e/o sottotipo e configurare più di 250 campi per descrivere i CI in modo più funzionale e dettagliato in base alle esigenze dell'utente;

- **esegue l'analisi delle cause principali**: SysAid CMDB segnala in modo evidente lo stato dei CI attraverso un'interfaccia di facile comprensione e una mappa grafica chiara. Ogni CI è rappresentato da una specifica icona: queste icone cambiano di colore nel caso in cui i CI siano associati a record di tipo incident, problem, change oppure ad eventi di monitoring;
- **stabilire le relazioni fra i CI**: SysAid CMDB permette di definire le relazioni tra CI e di presentarle all'interno di mappe grafiche. Definendo le connessioni fisiche (es. un computer connesso a una stampante) e logiche (es. il processo di fatturazione mensile dipende dal gestionale), è possibile pianificare e gestire al meglio i cambiamenti strategici. In questo caso si è in grado di creare un numero illimitato di relazioni fra CI, di mappare automaticamente le relazioni e di mostrare, per ciascuno di loro, i dettagli;
- **integrazione con tutti i moduli di SysAid**: SysAid CMDB è completamente integrato con le restanti funzionalità di SysAid.

Sappiamo che il SysAid CMDB può essere utilizzato con altri moduli appartenenti al software SysAid senza alcun tipo di problema, in quanto vengono fornite diverse funzionalità che ne permettono l'utilizzo.

Per il progetto, però, è stato necessario il SysAid WSDL file, già installato ed utilizzato dall'azienda, che fornisce un insieme di web services che consentono a software di terze parti (esattamente come il server Web oppure i service che girano su Skynet) di connettersi a SysAid; di conseguenza tutto questo può essere usato per inserire dati all'interno del SysAid CMDB oppure raccogliere dati da questo per l'utilizzo all'interno delle applicazioni.

Ricapitolando, per consentire l'uso di applicazioni esterne a SysAid, questo fornisce delle API che permettono di fare delle richieste ed ottenere delle risposte SOAP per e dal CMDB rendendolo a tutti gli effetti un Web service.

# 4 – Progetto

Il lavoro svolto durante questo periodo di tirocinio/tesi consiste nella realizzazione di un server Web che ha il compito di allineare specifici dati, monitorati dall'azienda attraverso Skynet e non solo, con il SysAid CMDB e nella realizzazione di service che permettono di aggiungere ulteriori informazioni con lo scopo di semplificare le necessità aziendali continuando a soddisfare le richieste dei clienti.

Tra i diversi service realizzati, ci sono quelli che vengono utilizzati per arricchire le informazioni iniziali che sono state memorizzate dal server Web all'interno del SysAid CMDB e altri che servono per il monitoraggio e controllo di parametri di interesse per l'azienda.

Il server ha il compito di controllare i dati monitorati e creare una CI (Configuration Item) all'interno del SysAid CMDB inserendo le informazioni iniziali. In realtà, tali informazioni sono quelle essenziali che devono essere obbligatoriamente presenti perché ritenuti fondamentali e indispensabili per SysAid CMDB affinché la scrittura al suo interno possa avvenire con successo. Tali parametri variano in base a ciò che si chiede di controllare e scrivere al server Web; tipicamente, quelli richiesti dal SysAid CMDB sono:

- *ciName*: rappresenta il nome del CI;
- *ciType*: rappresenta il tipo del CI. Questo potrebbe essere Server, Database, Middleware, NL-Product (NoemaLife Product) e molto altro. Il tutto dipende da ciò che l'azienda vuole registrare all'interno del SysAid CMDB;
- *ciSubType*: tale parametro, invece, rappresenta il sottotipo. Questo dipende dal modo in cui è stato settato il parametro precedente; ad esempio, se il ciType è Server allora il ciSubType può essere Phisical, Logical oppure Unknow; se il ciType è NL-Product allora il ciSubType può essere Galileo, Halia, People, DNWeb o altri prodotti che sono stati inventati dall'azienda e molto altro ancora;
- *company*: rappresenta un intero che indica la company a cui si fa riferimento; ad esempio, il numero 20 rappresenta BRESCIA POLIAMBULANZA ed è la company su cui sono stati effettuati i miei esperimenti e i miei test.

Oltre a questi, ci sono moltissimi altri parametri che possono essere settati come *historyVersion*, *status, subCategory*, *priority*, *owner*, *ownerGroup*, ecc… oppure

parametri che possono essere personalizzati; è chiaro, però, che il tutto dipende da quali informazioni si vogliono registrare all'interno del SysAid CMDB; nel mio caso, questi sono quelli fondamentali ai quali bisogna aggiungere quelli richiesti dall'azienda.

Una volta che il server Web ha creato il CI corrispondente con tali informazioni, possiamo aggiungere dettagli attraverso i diversi service. In particolar modo, quello che bisogna fare è inserire il service di interesse su Skynet e farlo girare sulla macchina di interesse.

Quando il service viene inserito, questo, ad intervalli di tempo regolari, eseguirà sulla macchina fornendoci i risultati, i quali verranno inseriti nel corrispondente campo all'interno del SysAid CMDB.

Infine, l'intero progetto è stato integrato in un altro sistema di monitoraggio, ovvero Zabbix, per confrontare i due sistemi in termini di prestazioni e qualità e verificare la fattibilità del lavoro, cioè constatare che il server Web e i service hanno una base solida per poter essere utilizzati anche in altri sistemi.

## 4.1 – Architettura

Andiamo ad analizzare l'architettura generale del progetto in modo da capire quali sono le parti coinvolte.

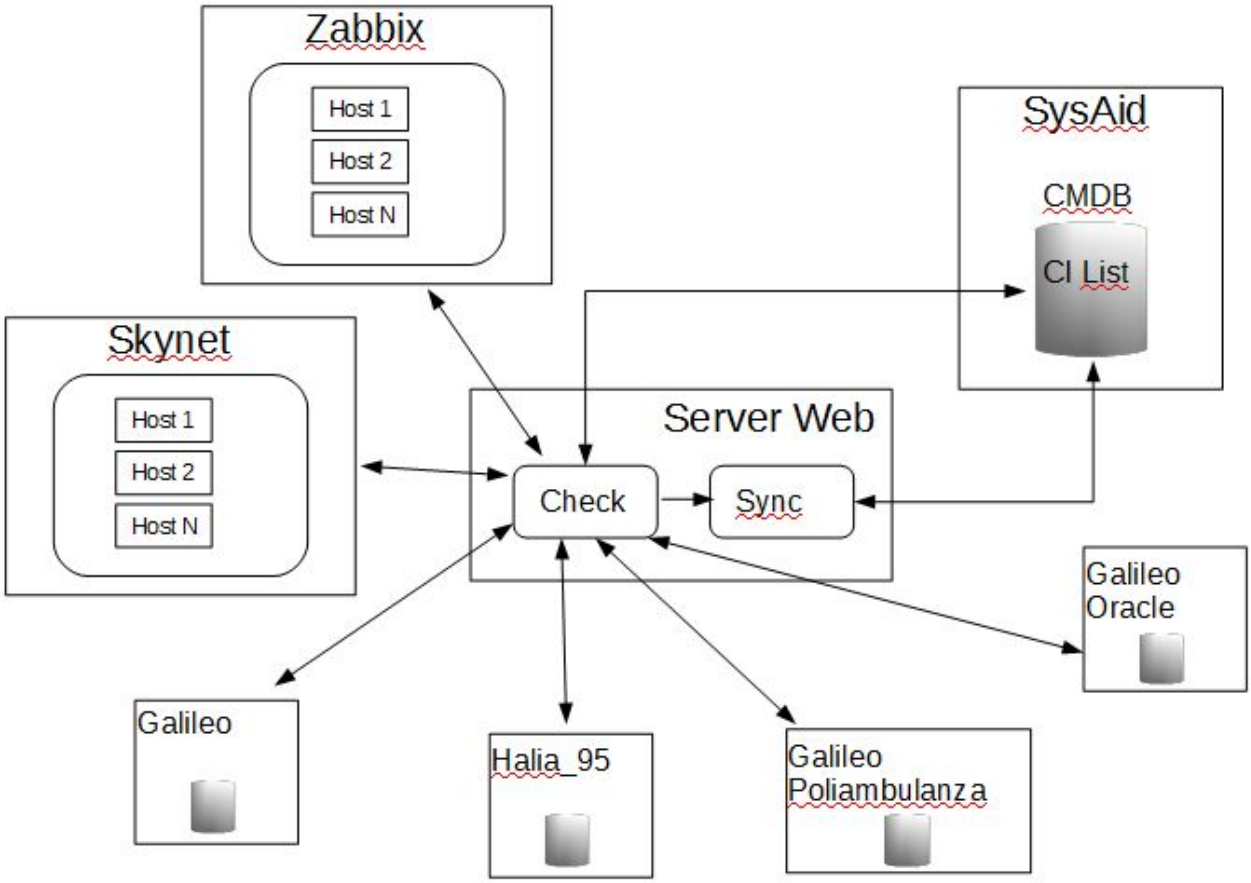



*Figura 49: Architettura generale*

Come possiamo vedere dalla Figura 49, il server Web effettua controlli su diverse macchine per svolgere i compiti richiesti. È possibile notare, infatti, come il server non interagisce solo ed esclusivamente con i sistemi di monitoraggio, ma anche con altre macchine che sono di interesse per l'azienda.

Il server Web presenta due processi che sono il Check e il Sync. Il primo consente al server di effettuare determinati controlli per restituire all'utente, attraverso il frontend, i risultati richiesti; il secondo, invece, dipende fortemente dal primo in quanto la scrittura non può essere fatta se non è stato effettuato alcun tipo di controllo; infatti, il Sync potrà essere eseguito solo ed esclusivamente dopo che il Check ha restituito i risultati della sua operazione.

Analizziamo, adesso, le componenti dell'architettura generale:

- SysAid è il software per la gestione dei servizi IT che contiene il SysAid CMDB all'interno del quale si vogliono scrivere i dati monitorati e contiene le

informazioni che sono fondamentali al customer service per la risoluzione dei problemi e per gestire e soddisfare le richieste dei clienti;

- Skynet è il sistema di monitoraggio dell'azienda su cui troviamo diversi host che vengono continuamente controllati e su ciascuno dei quali esegue un certo numero di service con lo scopo di fornire i parametri di interesse della macchina su cui eseguono;
- Galileo, Galileo_Poliambulanza, Halia_95 e Oracle_Galileo rappresentano i database con cui il server può interagire. Per potersi collegare con un database situato su macchine differenti, è necessario andare a modificare il file di configurazione dei database chiamato *tnsnames.ora*. Per l'azienda è risultato importante sapere quali database sono stati configurati per consentire alla macchina su cui si trova il server Web di interagire con loro;
- Zabbix è un altro sistema di monitoraggio. Anche su tale sistema troviamo diversi host che vengono controllati sempre e su ciascuno dei quale esegue un certo numero di item.

Dalla spiegazione di questi componenti, risulta chiaro, quindi, che il server può essere esteso per poter controllare e memorizzare all'interno del SysAid CMDB qualunque cosa possa risultare utile agli interessi e scopi aziendali.

## 4.2 – Funzionalità

Andiamo ad analizzare, nel dettaglio, i compiti svolti dal server Web e quelli svolti dai diversi service realizzati ricordando che una parte serve per aggiungere dettagli a ciò che il server ha inizialmente memorizzato, l'altra parte è di semplice monitoraggio per soddisfare le richieste aziendali.

### 4.2.1 – Server Web

In questo paragrafo andiamo ad analizzare le funzionalità presenti all'interno del server Web tenendo conto anche dell'architettura generale mostrata nella Figura 49.

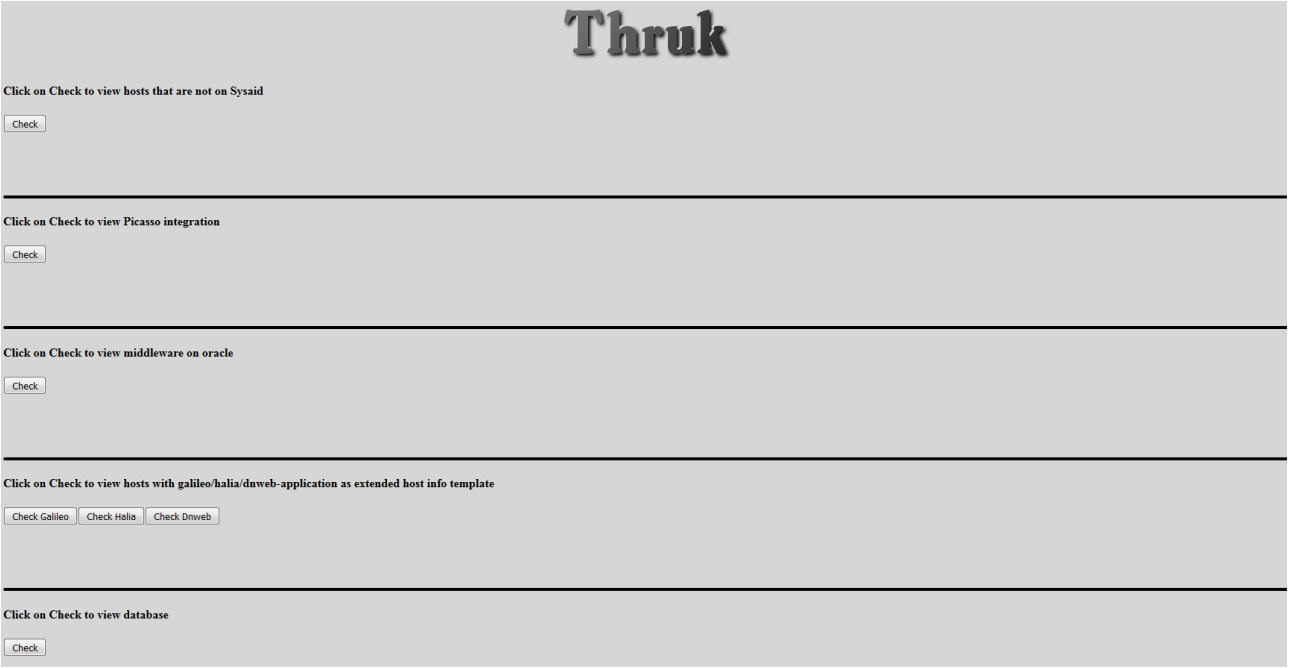



*Figura 50: Frontend del server Web*

La prima cosa importante da dire è che l'utente interagirà con il frontend del server Web (Figura 50) per decidere quale controllo effettuare e successivamente se voler effettuare la scrittura o meno all'interno del SysAid CMDB. Le scelte che un utente può fare sono le seguenti:

1. controllare se ci sono nuovi host monitorati con Skynet/Zabbix: nella sezione del frontend relativa a tale controllo (la prima della Figura 50 a partire dall'alto), l'utente non dovrà fare altro che cliccare sul tasto check per avviare il processo. Quello che succede è che il server, prima, va a controllare all'interno del database di Skynet/Zabbix quali sono gli host correntemente attivi ovvero quelli che sono monitorati dal sistema. Nello stesso tempo, però, il server effettua anche una richiesta SOAP verso il CMDB per ottenere a sua volta una lista di tutti gli host che sono già memorizzati. Una volta che il server

ha a disposizione queste informazioni, non fa altro che controllare quali host attivi sono già presenti nel CMDB andando a considerare, però, quelli che non sono presenti. Tali host vengono mostrati all'utente attraverso il frontend e sarà lui a decidere se effettuare la scrittura o meno. È chiaro, comunque, che se non ci sono nuovi host da scrivere, all'utente viene restituita una lista vuota; pertanto il processo termina in quanto non può essere effettuata la scrittura di nulla. In questo caso, l'utente può solo tornare alla pagina iniziale per decidere se effettuare altri controlli o meno. Aspetto importante è che, senza il check, il sync non potrà mai essere fatto. Nel momento in cui l'utente decide di effettuare la scrittura degli host all'interno del CMDB, clicca su tasto sync, che comparirà terminato il processo del check, e il server farà una richiesta SOAP al CMDB di voler effettuare la scrittura. In tale richiesta, oltre a trovare i parametri essenziali per la creazione di un CI, vengono aggiunti i parametri di interesse dell'azienda, vale a dire l'alias dell'host e il suo indirizzo IP. Una volta che questa è stata effettuata, SysAid risponderà al server fornendogli una lista di CIId, cioè id che identificano univocamente ciascun CI creato nel CMDB, che mostrerà all'utente attraverso il frontend. In questo caso, l'utente, accedendo all'interfaccia grafica di SysAid potrà notare come i suoi host sono stati correttamente salvati;

2. controllare su quali host un particolare servizio, chiamato HSO JBOSS PICASSO (non di mia creazione, ma già presente su Skynet), risulta essere attivo: in questo caso, se l'utente è interessato a questo controllo, cliccherà sul tasto check relativo a questa sezione (la seconda della Figura 50 a partire dall'alto) avviando il processo. Anche in questo caso, il server effettuerà una query sul database di Skynet/Zabbix per verificare su quali host, correntemente attivi, è presente il command che caratterizza il service HSO JBOSS PICASSO. Nello stesso tempo, farà una richiesta SOAP a SysAid per ottenere la lista di tutti gli host già memorizzati all'interno del CMDB e che hanno tale service attivo. Attraverso queste informazioni, il server controllerà quali host sono già memorizzati e quali ancora no e restituirà quest'ultimi all'utente attraverso il frontend. Se non ci sono nuovi host attivi che possiedono tale servizio verrà restituita la lista vuota; in tal caso, l'utente può solo tornare alla pagina iniziale per decidere se effettuare altri controlli o meno. Arrivati a questo punto, l'utente può decidere, a sua discrezione, se effettuare la scrittura cliccando sul tasto sync. Nel momento in cui viene fatta questa azione, il server si preoccuperà di fare una richiesta SOAP di scrittura con determinati

parametri, ovvero quelli necessari per la scrittura nel CMDB con l'aggiunta di uno di interesse per l'azienda, cioè l'identificativo della macchina su cui il service gira. Questo viene fatto in quanto nel CMDB viene salvato il service quindi per distinguerli è necessario l'identificativo della macchina su cui questi eseguono. SysAid, ricevuta tale richiesta, si occuperà di creare nuovi CI che avranno i parametri fornitogli dal server e restituirà una lista di CIId creati. Tale lista verrà mostrata all'utente attraverso il frontend, in modo che può andare a controllare sull'interfaccia Web di SysAid i nuovi CI che sono stati scritti;

3. controllare quali sono i middleware: in questa circostanza, il middleware potremmo definirlo come un servizio applicativo che convoglia i messaggi tra i service sender e quelli receiver. I messaggi possono essere inoltrati in diverso modo:
   - sender → receiver;
   - n sender → receiver;
   - sender → n receiver;
   - n sender → n receiver.

   Tutto questo serve per ricreare la mappatura tra service sender e receiver di uno dei middleware presenti in un ospedale. È chiaro che, avendo un convogliamento di messaggi di questo tipo, è necessario creare un CI corrispondente a ciascun collegamento fra i servizi. In questa circostanza, se l'utente è interessato a tale controllo (terza sezione della Figura 50 a partire dall'alto), nel momento in cui effettua il check corrispondente, il server si collega al database Galileo_Oracle (in cui sono contenute le informazioni relative al middleware) per effettuare la query e ottenere le informazioni che desidera. Fatto questo, fa una richiesta SOAP per ottenere la lista di middleware già presenti nel CMDB ed effettua il confronto per mostrare all'utente quali solo quelli che non risultano presenti nel CMDB. Una volta mostrati i risultati, l'utente, attraverso il tasto sync, avvia il processo di scrittura, di conseguenza, il server fa una richiesta SOAP che contiene i parametri essenziali per la creazione del CI e l'identificativo di ciascun service sender. SysAid, fatta la scrittura, restituisce la lista dei CIId appena creati al server, il quale a sua volta li mostra all'utente attraverso il frontend;

4. controllare quali sono quegli host su cui girano i prodotti NoemaLife Galileo, Halia e DNWeb: quando un host viene creato, gli viene attribuito un *extended host info template*/*groups* (in base al sistema di monitoraggio Skynet/Zabbix) che permette all'utente di capire quale prodotto gira su questa macchina. In altre parole è come se si formassero gruppi di host caratterizzati dallo stesso prodotto. Nel frontend del server, nella sezione relativa a questo controllo (la quarta nella Figura 50 a partire dall'alto), l'utente può decidere quale check fare in quanto sono presenti tre tasti check, uno per ogni prodotto. In base alla scelta dell'utente, il server effettuerà i corrispettivi controlli; in particolar modo controllerà su Skynet/Zabbix quali sono gli host attivi che hanno un determinato *extended host info template*/*groups* (chiamati galileo-application, halia-application e dnweb-application in Skynet e galileo-server, halia-server e dnweb-server in Zabbix); nello stesso tempo, farà una richiesta SOAP a SysAid per ottenere la lista di tutti gli host aventi un determinato template/groups che sappiamo dipende dal check che è stato richiesto. Da queste informazioni, il server verifica gli host che non sono presenti sul CMDB e li mostra nel frontend all'utente. L'utente decide o meno se effettuare la scrittura. Premendo il tasto sync, il server fa una richiesta SOAP di scrittura dei dati a SysAid, il quale restituisce la lista dei CIId appena creati. In questa richiesta, come parametri, vengono passati i soliti per la scrittura nel CMDB. ;
5. controllare quali sono i database con cui la macchina, su cui gira il server Web, è collegata: l'utente, se interessato a sapere quali sono i database che possono essere acceduti dalla macchina, nella sezione relativa a tale controllo (l'ultima della Figura 50) esegue il check. In questa circostanza, il server va a verificare quali sono i database con cui riesce a connettersi attraverso il suo file di configurazione *tnsnames.ora*; da questo file estrae l'indirizzo IP (serve solo per poter effettuare la connessione) della macchina su cui si trova il database, la porta su cui è in ascolto e il SID (identificatore di istanza del database). Per ogni database individuato, inoltre, effettua una query per estrarre la sua versione. Fatto questo, il server fa una richiesta SOAP per ottenere la lista di tutti i database che sono già registrati nel CMDB, effettua il confronto e mostra all'utente quelli che non risultano essere ancora memorizzati. A questo punto, l'utente, attraverso il tasto sync, effettua una richiesta SOAP per poter effettuare la scrittura. In tale richiesta, oltre ai soliti parametri essenziali che consentono la creazione del CI, troviamo anche la versione del database, la porta su cui è in ascolto e il SID. SysAid, terminata la creazione dei CI,

risponde al server con la lista dei CIId che sono stati appena creati. Ricevuta tale lista, il server la mostra all'utente attraverso il frontend;

Queste sono le funzionalità del server che sono state realizzate in quanto di interesse per l'azienda. Un aspetto importante, però, è che quando si tratta di monitoraggio è possibile controllare qualsiasi cosa possa essere di interesse personale o aziendale; pertanto, il server può essere ampliato a proprio piacimento facendo risparmiare moltissimo tempo nella scrittura dei singoli CI che dovrà essere effettuata manualmente.

### 4.2.2 – Service

In questo paragrafo, andiamo ad analizzare le funzionalità dei diversi service realizzati, ricordando che questi potremmo dividerli in due parti: service che permettono di arricchire le informazioni del CI contenuto all'interno del CMDB e service che vengono utilizzati per soli scopi di monitoraggio all'interno del sistema Skynet e Zabbix.

Partiamo da quei service che, quando eseguono su una macchina, non solo consentono di monitorare parametri di interesse per la macchina stessa, ma anche di andare a salvare o modificare i parametri stessi nel corrispondente campo del CI associata all'host su cui i service si trovano:

- GALILEO VERSION: Galileo è un prodotto che è stato realizzato con lo scopo di aiutare i medici ad avere, rapidamente, a disposizioni le informazioni necessarie per prendere decisioni, gestire e monitorare i processi clinici e ridurre errori; aiutare il personale infermieristico a pianificare l'assistenza e a raggiungere l'eccellenza nell'attività di cura ed, infine, aiutare il management a controllare i "key performance indicators" per prendere decisioni informate e incentivare il successo. Tale servizio ha il compito di controllare quale sia la versione del prodotto Galileo che è presente sull'host che vogliamo monitorare. In questo caso, la versione del prodotto viene scritta all'interno di un file di testo accessibile attraverso l'URL *http://ip:port/version.txt*. Quando il prodotto viene installato o viene aggiornato, all'interno di questo file viene scritta la nuova versione in questo modo:

  *2014-11-04 15:26:58 Installation of Version Galileo-1.5.0.0*

  *2016-04-14 16:59:29 Update to Version galileo.core-1.5.4*

  *2016-06-14 16:00:00 Hotfix to Version galileo.core-1.5.4.5*

Lo scopo del service, che esegue sulla macchina in cui è installato il prodotto e monitorata da Skynet/Zabbix, è quello di verificare l'ultima versione che è stata installata o aggiornata (quella più in basso) e di mostrarla attraverso il frontend del sistema di monitoraggio. Oltre a questo, controlla su SysAid CMDB, se esiste la macchina su cui sta eseguendo e se così fosse verifica se la versione presente all'interno del CMDB risulta essere diversa da quella monitorata. Se così fosse, effettua la scrittura di ciascun valore della versione nell'appropriato campo: HighVersion (primo numero da sinistra), MainVersion (secondo numero), Patch (terzo numero) e Hotfix (quarto numero);

- GALILEO MEMORY: questo service ha il compito di andare a controllare particolari parametri di "memoria" del prodotto Galileo installato sulla macchina in cui esegue il service stesso. Tali parametri sono i seguenti:
  - Xms: rappresenta la dimensione iniziale dell'heap;
  - Xmx: rappresenta la massima dimensione dell'heap;
  - XX:MaxPermSize: imposta la dimensione massima dello spazio di generazione permanente.

  Questi parametri sono fondamentali per il monitoraggio di applicazioni Java e rappresentano la quantità di memoria che la Java Virtual Machine ha inizialmente a disposizione e quantità massima che può utilizzare.

  Tale informazioni possono essere ottenute attraverso l'URL *http://$ip:$port/webmed/monitoring?jmxValue=java.lang:type=Runtime.InputArguments*.

  Anche in questo caso, tale service, ottenuti questi valori, va a verificare se nel CMDB vi è il CI corrispondente all'host su cui esegue il service, controlla se i parametri sono uguali o diversi e in quest'ultimo caso effettua la scrittura dei valori nei rispettivi campi: Xmx, Xms e XX:MaxPermSize;
- HALIA VERSION: Halia è un altro prodotto NoemaLife che ha il compito di semplificare la comunicazione fra due aspetti del Laboratory environment, di migliorare l'automatizzazione del flusso dei dati e di consolidarli. L'obiettivo di tale service è quello di controllare la versione di questo prodotto. In questa circostanza, la versione viene estratta dal database (HALIA_95 Figura 49) contenuto nella macchina su cui è installato il prodotto. Esattamente come GALILEO VERSION, anche qui il service ha il compito di verificare se la

macchina su cui il prodotto è installato è già presente all'interno del CMDB; se così fosse, verifica se la versione già memorizzata è uguale o diversa a quella restituita dal service e, in caso di diversità, effettua la scrittura della versione dividendola come sempre in HighVersion, MainVersion, Patch e Hotfix;

- DEVICES CONFIGURATION HALIA: tale service ha il compito di restituire la lista di quei dispositivi medicali collegati con Halia. Tali strumenti consentono ad Halia di gestire i campioni di laboratorio, di mandarli al laboratorio stesso e di associare il risultato ottenuto al campione. Per ottenere questa informazione, viene fatta una query sul database di Halia all'interno del quale sono registrati i dispositivi medicali in uso. Anche questo servizio si occupa di verificare se la macchina, su cui esegue, è presente nel SysAid CMDB e di controllare se i dispositivi memorizzati sono diversi da quelli ottenuti. Se così fosse, allora il service si occupa di effettuare anche la scrittura dei dispositivi nell'apposito campo del CI corrispondente all'host;
- AMBIENTE JAVA DNWEB: tale servizio, come dice il nome stesso, ha il compito di individuare la versione di Java Environment di un altro prodotto NoemaLife che prende il nome di DNWeb. Tale prodotto è stato realizzato con lo scopo di consentire a medici o infermieri di fare delle richieste di laboratorio dal reparto in cui effettuano il servizio al laboratorio stesso per un paziente ricoverato. Tali informazioni sono reperibili attraverso l'URL *http://$ip:$port/servlet/it.dianoema.dnweb.dnlis.servlets.About* da cui otteniamo l'informazione in questo formato:

  *Ambiente Java=Sun Microsystems Inc. 1.6.0_18*.

  Come i servizi precedentemente citati, anche questo, in seguito al controllo e l'ottenimento della versione dell'ambiente Java su cui è installato il prodotto, si occupa di controllare se l'host su cui gira è presente all'interno del CMDB e di verificare se la versione presente sia uguale o diversa da quella ottenuta durante il monitoraggio. In caso di diversità, la scrittura viene effettuata nel rispettivo campo;
- APACHE TOMCAT DNWEB: tale service ha il compito di controllare la versione dell'Apache Tomcat del prodotto DNWeb installato sull'host in cui il service stesso esegue. Le informazioni necessarie sono ottenute dallo stesso URL specificato nel service precedente nel formato:

  *Tipo=Apache Tomcat/5.5.36*.

Anche in questo caso, il service controlla se l'host è già memorizzato nel CMDB, verifica che la versione risulti essere differente da quella già presente ed effettua la scrittura facendo una richiesta SOAP a SysAid;

- OE LIS DNWEB: tale servizio permette di ottenere la versione di OE LIS del prodotto DNWeb, che non è altro che un micro-modulo del prodotto stesso che ha il compito di effettuare richieste di laboratorio. Tale informazione può essere ottenuta attraverso l'URL utilizzato per Apache Tomcat e Ambiente Java e si presenta nel seguente formato:

  *La versione di OE LIS in uso è la 3.6.0*.

  Estratta la versione, questa viene visualizzata nel sistema di monitoraggio e, come per i service precedenti, anche in questa circostanza si va a verificare se l'host su cui esegue tale service ha il corrispondente CI nel CMDB. In caso positivo, verifica se la versione risulta essere differente e se così fosse effettua la scrittura;

- PEOPLE VERSION: People è un altro prodotto di NoemaLife che può essere definito come un componente software che ha il compito di gestire, a livello centralizzato e ospedaliero, le informazioni anagrafiche di un paziente che viene ricoverato. Tale service, appunto, ha il compito di controllare la versione di questo prodotto che come per tutti gli altri prodotti risulta fondamentale in quanto se ci sono aggiornamenti e cambio di versioni il sistema potrebbe non funzionare. Tali informazioni possono essere ottenute attraverso l'URL *http:// $ip:$port/people/People.jnlp*. Infine, il service si occupa di verificare se l'host su cui esegue è presente nel CMDB, se la versione memorizzata è la stessa di quella ottenuta e in caso contrario viene scritta nel corrispondente campo scompattando la versione in HighVersion, MainVersion, Patch e Hotfix.

I service elencati sono quelli che vengono utilizzati per popolare i diversi campi presenti nei CI che sono stati creati dal server Web.

Passiamo adesso ad altri servizi che sono stati realizzati con lo scopo di monitorare parametri ritenuti importanti ai fini aziendali e che, andando a modificare leggermente il codice sorgente, possono diventare dei service che incrementano le informazioni contenute nel SysAid CMDB.

Partiamo con quei service che permettono di ottenere informazioni desiderate attraverso il monitoraggio con JavaMelody.

JavaMelody è un'applicazione Open Source che ha l'obiettivo di monitorare applicazioni Java e Java Enterprise Edition in ambienti di produzione e QA. Questo tool permette di misurare e calcolare statistiche su reali operazioni di un'applicazione che dipende sull'utilizzo della stessa da parte degli utenti. Alcuni indicatori importanti che fornisce JavaMelody sono numero di esecuzioni, tempi di esecuzione, percentuale di errori di richieste HTTP, SQL, Java memory, Java cpu, numero di sessioni utenti, numero di connessioni JDBC e molto altro. Verifichiamo adesso quali sono i service che sono stati realizzati considerando l'applicazione Java webmed che esegue sulla macchina SRV-NLGAL642-LB:

- ACTIVE CONNECTIONS: tale service permette di ottenere il numero di connessioni che sono correntemente attive presso l'applicazione Java che si sta monitorando. Per poter ottenere questo service è stato necessario estrarre l'ultimo valore riportato nel grafico corrispondente mostrato da JavaMelody:

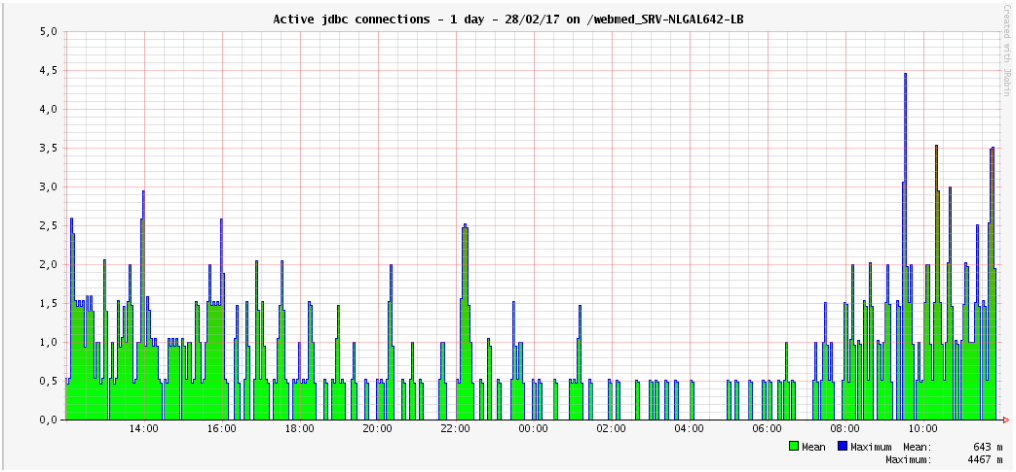



*Figura 51: Monitoraggio di connessioni attive con JavaMelody*

Dalla Figura 51 è possibile notare come l'andamento delle connessioni attive risulta essere vario; come detto in precedenza, però l'obiettivo è quello di ottenere sempre il valore più recente e mostrarlo sul sistema di monitoraggio;

- ACTIVE THREADS: tale service permette di ottenere il numero di thread che sono correntemente attivi sull'applicazione Java che si sta monitorando. Esattamente come per il service precedente, anche qui, per il monitoraggio attraverso Skynet, si va a considerare il valore più recente da estrarre dal grafico corrispondente mostrato in JavaMelody:

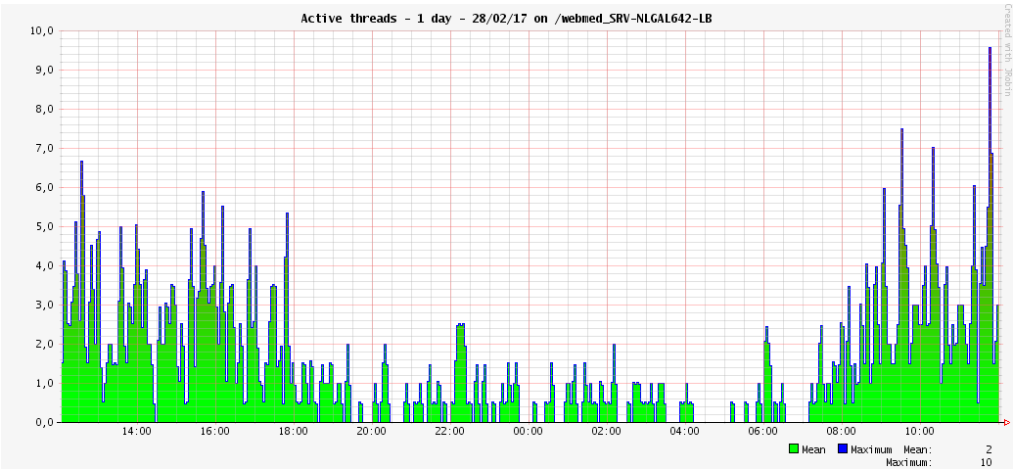



*Figura 52: Monitoraggio di thread attivi con JavaMelody*

Anche in questo caso, l'andamento dei thread è molto vario, ma l'obiettivo rimane sempre quello di mostrare all'utente l'ultimo valore assunto quando è stato fatto il check del service;

- CPU: questo service serve per monitorare la CPU utilizzata dall'applicazione Java che si sta controllando.

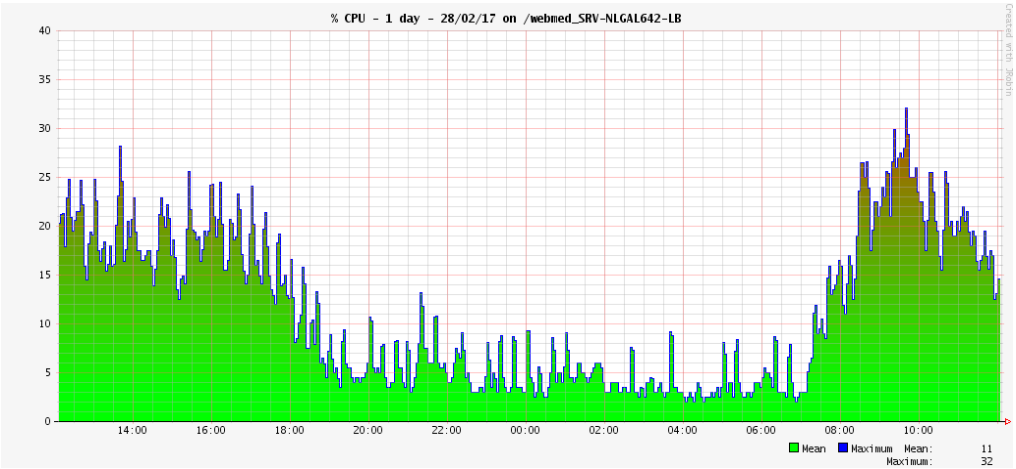



*Figura 53: Monitoraggio della CPU con JavaMelody*

L'obiettivo, anche in questo caso, è quello di andare a considerare il valore più recente estrapolandolo dalla Figura 53. Il valore aggiornato, chiaramente, sarà quello assunto dal service nel momento in cui ha fatto il check sull'host su cui sta girando;

- USED CONNECTIONS: tale service permette di sapere il numero di connessioni che sono correntemente attive con l'applicazione in questione.

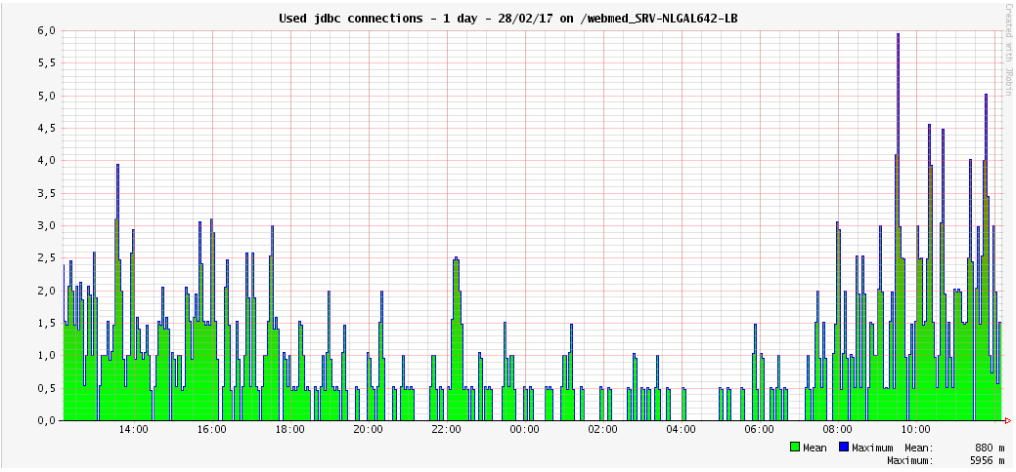



*Figura 54: Monitoraggio delle connessioni utilizzate con JavaMelody*

Anche per questo service, il discorso è esattamente lo stesso di quello fatto per i precedenti. Si va a considerare dal grafico, il valore più recente e lo si mostra sull'interfaccia di monitoraggio di Skynet, ricordando che la freschezza del valore è dato dal momento in cui il service esegue il check sulla macchina su cui sta eseguendo;

- USED MEMORY: questo service, infine, va a monitorare la memoria utilizzata dall'applicazione Java che si sta controllando.

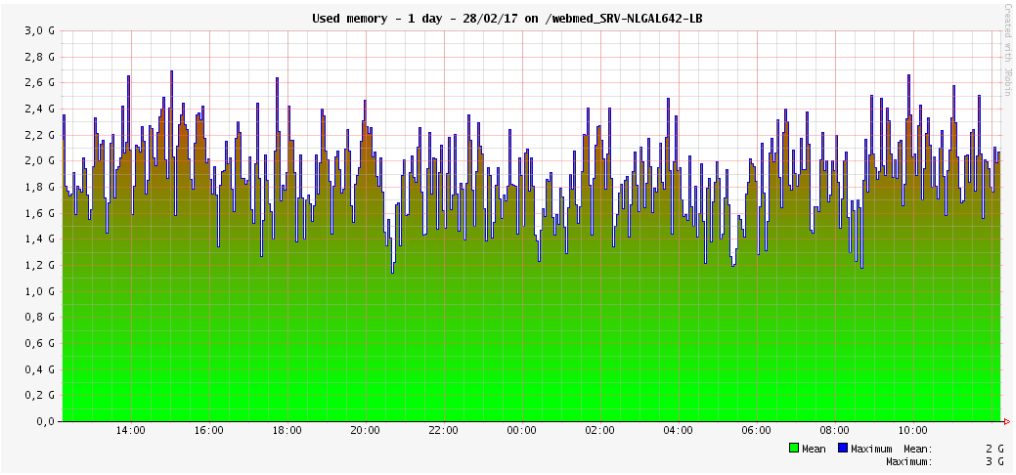



*Figura 55: Monitoraggio della memoria utilizzata con JavaMelody*

Nella Figura 55, si può notare come i valori sono molto più elevati. In questo caso la memoria viene monitorata con unità di misura pari al Giga. Anche in quest'ultimo service, si va a considerare il valore più recente e lo si mostra all'utente attraverso l'interfaccia grafica di Skynet.

Passiamo adesso ad altri service che permettono di ottenere indicatori dell'uso della soluzione informatizzata NoemaLife per svolgere le principali operazioni cliniche e diagnostiche.

Tali service, infatti, sono stati realizzati per scopi statistici con l'obiettivo di capire qual è la mole di lavoro fatto su Galileo. Andiamo a vedere quali sono:

- CARTELLE CLINICHE: questo service permette di ottenere il numero medio di accessi alla settimana alle cartelle cliniche;
- LETTERE DI DIMISSIONI: questo service permette di ottenere il numero medio di lettere di dimissioni realizzate in settimana;
- REFERTI DI CONSULENZA: questo service permette di ottenere il numero medio di referti di consulenza a settimana;
- RICHIESTE DI ANATOMIA PATOLOGICA: questo service permette di ottenere il numero medio di richieste di anatomia patologica fatta settimanalmente;
- RICHIESTE DI CONSULENZA: questo service permette di ottenere il numero medio di richieste di consulenza per settimana;
- RICHIESTE DI LABORATORIO: questo service permette di ottenere il numero medio di richieste di laboratorio per settimana;
- RICHIESTE DI RADIOLOGIA: questo service permette di ottenere il numero medio di richieste di radiologia per settimana.

Come si può notare dalla definizione di ciascun service, è chiaro che si tratta solo di dati statistici con lo scopo di fare una stima delle richieste e referti fatti settimanalmente. Attraverso Skynet, questi service vengono controllati una volta a settimana, precisamente il venerdì sera, in modo da avere le statistiche disponibili per l'inizio della settimana successiva. Tutto questo permette di automatizzare un'operazione che dovrebbe essere fatta manualmente e potrebbe richiedere moltissimo tempo. Infine, tali servizi vengono ottenuti attraverso delle query fatte sul database di Galileo (mostrato in Figura 49).

Andiamo ad analizzare, adesso, altri due servizi che riguardano sempre il prodotto Galileo. Questo prodotto può essere definito come un grande framework che permette di gestire anche la cartella clinica dei pazienti. Galileo espone delle API, di conseguenza c'è la possibilità di scrivere micro-moduli che, importando le API stesse, possono agganciarsi all'interno framework. Per questo motivo, il prodotto viene arricchito di micro-moduli in modo da rendere possibile l'aggiornamento, l'inserimento e la rimozione di questi senza stravolgere l'intero core. Due sono i

moduli ritenuti importanti dall'azienda per i quali sono stati realizzati service da monitorare attraverso Skynet:

- GINPATIENTMR;
- GUTIL.

Ciascuno di questi service ha il compito di estrarre la versione del micro-modulo corrispondente (GinPatientMR e Gutil appunto) in modo da rendere facile l'intervento nel caso in cui qualcosa nell'intera infrastruttura, lato cliente, non dovesse funzionare. Esattamente come per i service citati in precedenza, anche questi ottengono le informazioni richieste accedendo al database di Galileo.

Un altro service importante, relativo al prodotto Galileo è il seguente:

- GALILEO VERSION DB: questo service ha il compito, invece, di verificare quale sia la versione del database Galileo che si trova su una determinata macchina. Per poter fare questa operazione, viene fatta una query sul database.

Ritornando, invece, al prodotto Halia vi è un altro service particolare:

- BUS: il bus è un oggetto scritto in Java che si trova sul JBoss di Halia. Tale oggetto ha il compito di prendere i campioni di DNLab, importarli sul database di Halia e, ottenuti i risultati dei campioni, li importa anch'essi nel database. In pratica, possiamo definirlo come una specie di "integratore" che mantiene allineati i database. Per ottenere i bus del prodotto Halia, viene fatta una query sul corrispondente database.

I prossimi service, invece, sono stati realizzati con lo scopo di ottenere informazioni di base di una macchina Linux. Andiamo ad analizzarli:

- HSO OPERATING SYSTEM REMOTE SSH CPU NUMBER: questo service si occupa di controllare, ad ogni check, il numero di processori posseduti dalla macchina Linux che si sta monitorando e di conseguenza macchina su cui il service esegue. Tale operazione viene fatta attraverso la lettura di un particolare file, */proc/cpuinfo*, posseduto dall'host monitorato. In particolar modo, si va ad estrapolare il campo *Processor* che indica il numero di processori posseduti dalla macchina;
- HSO OPERATING SYSTEM REMOTE SSH CPU MODEL: questo service si occupa di monitorare, ad ogni check, il modello di processore che la macchina, su cui il service esegue, possiede. Anche questa informazione viene estratta dal

file */proc/cpuinfo*, in particolar modo si considera il campo *model name* che indica, appunto, il modello del processore posseduto dalla macchina;

- HSO OPERATING SYSTEM REMOTE SSH RAM: tale service fornisce l'informazione relativa alla dimensione della memoria di RAM posseduta dall'host su cui il service viene fatto eseguire. In questo caso, però, l'informazione viene estratta dal file */proc/meminfo* in particolar modo si va a considerare il campo *MemTotal* che indica, appunto, la dimensione massima della RAM posseduta dal dispositivo monitorato;
- HSO OPERATING SYSTEM REMOTE SSH OS: questo service restituisce il sistema operativo posseduto dall'host su cui il service viene messo in esecuzione. Per ottenere questa informazione, viene eseguito sulla macchina un particolare comando che prende il nome di *lsb_release* che restituisce tutte le informazioni di distribuzione del sistema che caratterizza l'host. Per estrarre il sistema operativo, in questo caso, si va a considerare il campo *Description*;
- HSO OPERATING SYSTEM REMOTE SSH HOSTNAME: questo service, infine, permette di sapere quale sia l'hostname della macchina su cui il service stesso esegue.

Tutte queste informazioni, come si può notare dal nome di ciascun service, vengono ottenute eseguendo il comando sulla macchina remota attraverso ssh.

Ultima categoria, se così possiamo chiamarla, di service realizzati è quella che permette di ottenere le stesse informazioni per i sistemi Linux, ma in ambiente Windows. Andiamo a vedere quali sono:

- HSO OPERATING SYSTEM OS: tale service permette di ottenere il sistema operativo della macchina Windows che si sta monitorando attraverso Skynet e su cui viene fatto eseguire il service;
- HSO OPERATING SYSTEM CPU NUMBER: tale service permette di ottenere il numero dei processori posseduti dalla macchina Windows che si sta monitorando attraverso Skynet e su cui viene fatto eseguire il service;
- HSO OPERATING SYSTEM CPU MODEL: tale service permette di ottenere il modello del processore della macchina Windows che si sta monitorando attraverso Skynet e su cui viene fatto eseguire il service;

- HSO OPERATING SYSTEM RAM: tale service permette di ottenere la dimensione della RAM posseduta dalla macchina Windows che si sta monitorando attraverso Skynet e su cui viene fatto eseguire il service;
- HSO OPERATING SYSTEM HOSTNAME: tale service permette di ottenere l'hostname della macchina Windows che si sta monitorando attraverso Skynet e su cui viene fatto eseguire il service.

Tutti questi service sono stati ottenuti attraverso lo studio e l'utilizzo del Windows Management Instrumentation Command-line (WMIC) tool. WMIC è una semplice riga di comando e interfaccia di scripting che semplifica l'utilizzo di strumenti di gestione di sistemi Windows. Attraverso delle query pre-formattate è possibile ottenere, da remoto, informazioni di monitoraggio dell'host di interesse.

Tutto il lavoro inizialmente è stato fatto su Skynet, in quanto è il sistema di monitoraggio utilizzato dall'azienda NoemaLife, ma in un secondo momento il tutto è stato integrato in Zabbix ottenendo gli stessi ed identici risultati, modificando in modo molto lieve il codice sorgente del server Web e dei service che, in prima battuta, erano stati realizzati appositamente per Skynet.

Nel capitolo successivo, che parla delle scelte implementative fatte per svolgere tale lavoro, permetterà di capire quanto lievi risultano essere le modifiche da fare, mostrando come il progetto, in toto, risulta essere solido e abbastanza generalizzato da poter essere utilizzato anche su altri sistemi di monitoraggio.

# 5 – Implementazione

In questo capitolo, verranno affrontate le scelte implementative che sono state fatte per la realizzazione dell'intero progetto, gli strumenti utilizzati, la realizzazione dello stesso ed, infine, i risultati che si sono ottenuti.

La prima cosa importante da dire è che sono state create tre macchine virtuali su cui mettere in esecuzione Skynet, il server Web realizzato e Zabbix.

Per quanto riguarda Skynet e il server Web, il sistema operativo installato è un Oracle Linux Server versione 6.3. Nel primo caso, non si può parlare di scelta in quanto la macchina virtuale su cui si trova il sistema di monitoraggio aziendale mi è stata fornita dall'azienda già funzionante, quindi, si tratta della stessa versione che è in uso. Per quanto riguarda il server Web, invece, poiché inizialmente doveva interagire con Skynet, è stato scelto lo stesso sistema operativo proprio per non incontrare problematiche che non riguardano la realizzazione del server, ma di altra natura, come pacchetti di base differenti che possono risultare incompatibili.

Per quanto riguarda Zabbix, invece, si è deciso di utilizzare il sistema operativo CentOS 7. La scelta è ricaduta sul fatto che l'ultima versione stabile di Zabbix è la 3.2 (esiste anche la versione 3.4, ma al momento non è stabile ed è ancora in fase di sviluppo); tale versione del sistema di monitoraggio, chiaramente, presenta come requisito un sistema operativo appartenente alla famiglia RedHat Enterprise Linux, dalla versione 7 in su.

Parlato del punto di partenza è possibile analizzare gli strumenti che sono stati utilizzati per la realizzazione dei service che eseguono su host monitorati in Skynet e in Zabbix ed, infine, gli strumenti utilizzati per la realizzazione del server Web.

## 5.1 – Gli strumenti

Per la realizzazione dei service che devono eseguire sugli host monitorati da Skynet sono stati realizzati diversi script nel linguaggio di programmazione Perl. La scelta di questo linguaggio deriva semplicemente dal fatto che coloro che hanno creato Skynet e, successivamente, coloro che l'hanno utilizzato ritenevano questo linguaggio più consono allo svolgimento delle loro attività. Si può ricordare, infatti, che alla base di Skynet vi è Nagios, sistema di monitoraggio che esegue plugin che possono essere scritti in Perl, Shell, Bash e altri linguaggi, ma come detto in precedenza, tale sistema di monitoraggio mi è stato fornito già pronto all'uso con un certo numero di service già installati e scritti in questo linguaggio.

Per rimanere in linea al lavoro già svolto su Skynet, lo stesso linguaggio è stato utilizzato anche per realizzare script esterni in Zabbix. Questa scelta è dovuta anche al fatto di voler dimostrare come semplici e rapide modifiche permettono di passare da un sistema di monitoraggio all'altro.

Tutt'altro discorso, invece, deve essere fatto per quanto riguarda il server Web. Come prima cosa, tale server è stato realizzato a partire da zero, quindi non vi erano vincoli nella scelta del linguaggio di programmazione da utilizzare per la sua realizzazione. In questa circostanza il discorso da fare risulta essere molto più lungo e complesso.

Partiamo col dire che il server Web è stato realizzato con Node.js.

Node.js è un Open Source cross-platform running environment per lo sviluppo di applicazioni server-side e di rete. È una piattaforma costruita su Google Chrome JavaScript Engine (V8 Engine) in cui le applicazioni sono scritte in JavaScript e possono essere eseguite all'interno di Node.js su diversi sistemi operativi come OS X, Windows e Linux.

Node.js è un modello event-driven e non-blocking I/O che lo rende leggero ed efficiente. In questo caso, tale modello prevede che un programma rimane in attesa di specifici eventi ai quali il sistema reagisce in un certo modo. Il tutto avviene in modo asincrono per garantire una maggiore efficienza di elaborazione.

Vediamo alcune caratteristiche che potrebbero rendere Node.js la prima scelta quando si vogliono realizzare particolari applicazioni:

- **Asincronicità e modello event-driven**: tutte le API della libreria Node.js sono asincrone e non bloccanti; di conseguenza, quando si realizza un server basato su questa piattaforma, tale server non attenderà mai il ritorno dei dati

dall'esecuzione delle API. Il server si sposta sull'API successiva dopo la chiamata e un meccanismo di notifiche di eventi aiuta il server a fornire le risposte alle funzioni che sono state invocate;

- **Velocità di esecuzione**: poiché tale piattaforma è costruita su Google Chrome V8 JavaScript Engine, la libreria di Node.js risulta essere molto veloce nell'esecuzione del codice;
- **Modello Single-Threaded, ma alta scalabilità**: Node.js utilizza un modello single-threaded con un event looping. Il meccanismo degli eventi aiuta il server a rispondere in modo non bloccante rendendolo altamente scalabile, in opposizione a server tradizionali che creano un numero limitato di thread per gestire le richieste. Node.js utilizza un programma single threaded e lo stesso programma può fornire il servizio ad un grandissimo numero di richieste esattamente come un server tradizionale, ad esempio Apache HTTP server;
- **Mancanza di buffering**: le applicazioni Node.js non bufferizzano alcun tipo di dato perché restituiscono i risultati alle richieste effettuate attraverso chunk;
- **Licenza**: Node.js è stato rilasciato sotto la licenza del MIT (Massachusetts Institute of Technology).

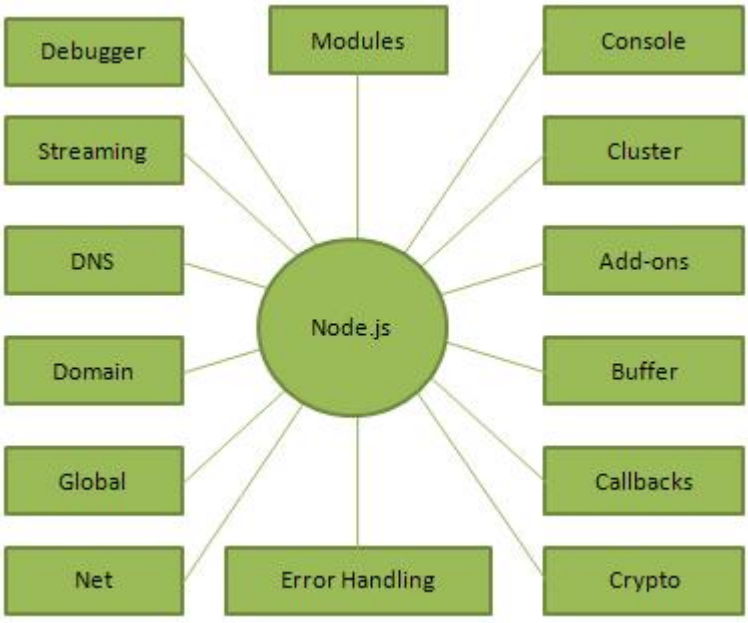



*Figura 56: Concetti fondamentali di Node.js*

Dalla Figura 56, è possibile notare quali sono i concetti fondamentali per Node.js. Due in particolare, però, risultano essere la base principale quando si vuole realizzare un'applicazione server-side con questa piattaforma:

- **Moduli**: il sistema base di Node.js mette a disposizione un numero limitato di metodi utilizzabili direttamente all'interno del codice. Per espandere le funzionalità è necessario integrare dei moduli, ciascuno dei quali aggiunge una classe o singoli metodi utilizzabili all'interno del codice. Per poter importare i moduli viene utilizzato il metodo *require*, mentre nel caso in cui si voglia creare un modulo da poter utilizzare nel codice è necessario "esportare" il modulo stesso e questo viene fatto attraverso il metodo *exports*;
- **Callback**: le callback sono funzioni che permettono all'applicazione realizzata di poter agire in modalità asincrona. Tale funzione, solitamente, viene passata come parametro di ingresso alla funzione chiamante; in questo modo la chiamante può realizzare un compito specifico (quello svolto dalla callback). È possibile riferirsi alle callback anche come funzioni chiamate dal sistema operativo con lo scopo di gestire particolare eventi (click con il mouse) e quindi consentire ad un programma, di livello più basso, di richiamare una funzione o servizio ad un livello più alto.

Un altro aspetto molto importante di Node.js è che si basa su un pacchetto, chiamato npm, che rappresenta il più grande ecosistema di librerie Open Source nel mondo.

Npm rappresenta il package manager di default per Node.js, infatti, npm viene incluso automaticamente quando Node.js viene installato. Npm consiste in una riga di comando che interagisce con un registro remoto; questo consente agli utenti di utilizzare e distribuire moduli JavaScript che sono disponibili nel registro. I pacchetti presenti nel registro sono nel formato CommonJS e ciascuno di loro include un file di metadati nel formato JSON. Il registro npm non prevede un processo di valutazione dei moduli realizzati, il che significa che i pacchetti presenti possono essere di bassa qualità, insicuri e dannosi; tuttavia, gli amministratori del server npm sono molto competenti e quindi sono in grado di eliminare pacchetti malevoli ed impedire ad utenti con cattive intenzioni di poter caricare ancora moduli nel registro.

Come si può capire da tutto questo, quindi, quando si va a scrivere un'applicazione Node.js e si necessita di moduli per raggiungere certi obiettivi, è necessario dover installare sulla propria macchina il modulo di interesse e poi importarlo nel codice che si sta scrivendo. Questo può essere fatto attraverso il comando *npm install nome-modulo*. Vediamo adesso quali sono i moduli basilari che vengono praticamente sempre usati quando si realizza un'applicazione Node.js:

- **Globals**: è un modulo atipico in quanto non rappresenta una vera e propria libreria, ma una serie di API incluse nel namespace globale dell'applicazione e quindi richiamabile direttamente. Tra le funzioni di questo pseudo-modulo si possono ricordare:
  - *require* che permette di includere nel codice scritto moduli aggiuntivi;
  - *exports* che permette di esportare moduli creati dall'utente;
  - *setTimeout* e *setInterval* che consentono di lavorare con i timer.

  Oltre a queste funzioni, tale modulo fornisce degli oggetti fondamentali come *console* che permette di accedere allo standard input/output e allo standard error del processo, *_filename* che fornisce, a livello runtime, il nome del file avviato e *_dirname* che ne fornisce la cartella corrente;
- **http**: questo modulo include una serie di oggetti che permettono di utilizzare il protocollo senza problemi. Inoltre, contiene la classe bootstrap http.Server che permette di avviare il server Web. Questo modulo rappresenta il punto di partenza senza il quale il server non può essere creato;
- **url**: questo modulo permette di creare, trasformare e manipolare gli url sotto forma di oggetti o stringhe. Di fondamentale importanza è il metodo *parse* che trasforma un url in un oggetto;
- **path**: questo modulo permette di lavorare con i percorsi reali della macchina sulla quale è avviata l'applicazione. Fornisce diversi metodi che consentono di navigare fra le cartelle e montare path complessi a partire da più stringhe. Questo modulo è molto importante per quelle applicazioni che lavorano con il filesystem della macchina;
- **fs (File System)**: questo modulo, a stretto contatto con il precedente, permette di lavorare con il filesystem della macchina, eseguendo tutte le operazioni tipiche come copia, rinominazione, cancellazione, lettura e scrittura di file e cartelle;
- **util**: questo modulo include una serie di funzionalità di utilità in ottica di introspezione delle variabili. Oltre ad una serie di funzioni logiche come *isArray* o *isDate* possiede anche la *format* per la formattazione delle stringhe a partire dal placeholder e la *debug* e la *log* come funzioni per il monitoraggio del flusso;

- **querystring**: questo modulo permette di creare una stringa di valori concatenati dal carattere & da un oggetto e viceversa. La funzione *querystring.stringify* permette di creare una stringa a partire da un oggetto, mentre la funzione *querystring.parse* permette di creare un oggetto a partire dalla stringa;
- **net**: questo modulo consente agli sviluppatori di implementare applicazioni client-server a prescindere dal protocollo HTTP, ma sfruttando le socket come strumenti di più basso livello e lavorare direttamente con le connessioni.

Tutti questi moduli citati rappresentano la base per la creazione di un'applicazione Node.js, ma come detto in precedenza, ci sono altri moduli che possono essere scaricati attraverso npm oppure c'è la possibilità di creare dei moduli personalizzati che vengono prima esportati e successivamente importati nel proprio codice main.

Ultimo aspetto importante per comprendere gli strumenti utilizzati per la realizzazione del progetto è che tutti i database utilizzati risultano essere di test, quindi non sono quelli realmente utilizzati dall'azienda. Lo stesso discorso si può fare per il SysAid CMDB, nel senso che è stato utilizzato quello di test su cui l'azienda effettua gli esperimenti ed, infine, le macchine che sono state monitorate sia con Skynet che con Zabbix sono server che non vengono utilizzati più dall'azienda.

## 5.2 – Struttura del progetto

In questo paragrafo verrà spiegato come è strutturato il progetto in termini di script realizzati per i service e moduli creati per il server Web.

Per quanto riguarda il server Web, sono stati creati diversi moduli che possono essere divisi in base al lavoro svolto. Per ottenere le informazioni di monitoraggio si ha:

- **adt.js**: modulo per ottenere i middleware dal database Galileo Oracle;
- **database.js**: modulo per ottenere i database con cui la macchina, su cui esegue il server Web, può collegarsi;
- **host.js**: modulo per ottenere la lista degli host correntemente attivi sul sistema di monitoraggio;
- **host_dnweb.js**: modulo per ottenere la lista di host correntemente attivi e su cui si trova il prodotto DNWeb;
- **host_halia.js**: modulo per ottenere la lista di host correntemente attivi e su cui si trova il prodotto Halia;
- **host_galileo.js**: modulo per ottenere la lista di host correntemente attivi e su cui si trova il prodotto Galileo;
- **services_picasso.js**: modulo per ottenere la lista degli host correntemente attivi e su cui esegue il service HSO JBOSS PICASSO.

Poi ci sono quei moduli che vengono utilizzati per controllare quali sono le informazioni di monitoraggio già presenti all'interno del SysAid CMDB:

- **nagios_id.js**: questo modulo permette di ottenere la lista degli host che sono già memorizzati all'interno del CMDB;
- **nagios_id_database.js**: questo modulo permette di ottenere la lista dei database che sono già memorizzati all'interno del CMDB;
- **nagios_id_dnweb.js**: questo modulo permette di ottenere la lista degli host DNWeb che sono già memorizzati all'interno del CMDB;
- **nagios_id_halia.js**: questo modulo permette di ottenere la lista degli host Halia che sono già memorizzati all'interno del CMDB;
- **nagios_id_galileo.js**: questo modulo permette di ottenere la lista degli host Galileo che sono già memorizzati all'interno del CMDB;

- **nagios_id_middleware.js**: questo modulo permette di ottenere la lista dei middleware e la lista degli host su cui esegue il service HSO JBOSS PICASSO che sono già memorizzati all'interno del CMDB. Questo modulo viene utilizzato per entrambi i controlli in quanto nel CMDB hanno gli stessi *ciType* e *ciSubType*.

Di seguito, ci sono quei moduli che si occupano di confrontare i dati monitorati con quelli all'interno del CMDB e di restituire quelli che non sono presenti:

- **check_adt.js**: restituisce i middleware;
- **check_database.js**: restituisce i database;
- **check_dnweb_host.js**: restituisce gli host DNWeb;
- **check_halia_host.js**: restituisce gli host Halia;
- **check_galileo_host.js**: restituisce gli host Galileo;
- **check_host.js**: restituisce tutti gli host correntemente attivi;
- **check_picasso.js**: restituisce gli host su cui è attivo il service HSO JBOSS PICASSO.

In seguito, ci sono i moduli che permettono di interagire con SysAid:

- **login.js**: tale modulo permette di effettuare il login presso il Web Service di SysAid in modo da poter effettuare le operazioni;
- **executeSelectQuery_type_database.js**: tale modulo restituisce la lista dei CIId presenti nel CMDB che hanno come *ciType* il valore Database e il *ciSubType* Oracle, ovvero, il tipo di database utilizzato;
- **executeSelectQuery_type_dnweb.js**: tale modulo restituisce la lista dei CIId presenti nel CMDB che hanno come *ciType* il valore NL-Product e come *cySubType* DNWeb;
- **executeSelectQuery_type_halia.js**: tale modulo restituisce la lista dei CIId presenti nel CMDB che hanno come *ciType* il valore NL-Product e come *cySubType* Halia;
- **executeSelectQuery_type_galileo.js**: tale modulo restituisce la lista dei CIId presenti nel CMDB che hanno come *ciType* il valore NL-Product e come *cySubType* Galileo;

- **executeSelectQuery_type_middleware.js**: tale modulo restituisce la lista dei CIId presenti nel CMDB che hanno come *ciType* il valore Middleware e come *cySubType* Picasso;
- **executeSelectQuery_type_server.js**: tale modulo restituisce la lista dei CIId presenti nel CMDB che hanno come *ciType* il valore Server. Il *cySubType* in questo caso non viene specificato in quanto non serve;
- **loadByStringId.js**: questo modulo restituisce tutte le informazioni per ogni singolo CIId che gli viene passato; infatti, questo modulo solitamente viene utilizzato dopo una executeSelectQuery;
- **save_adt.js**: tale modulo consente la scrittura dei middleware all'interno del CMDB;
- **save_database.js**: tale modulo consente la scrittura dei database all'interno del CMDB;
- **save_dnweb.js**: tale modulo consente la scrittura degli host DNWeb all'interno del CMDB;
- **save_halia.js**: tale modulo consente la scrittura degli host Halia all'interno del CMDB;
- **save_galileo.js**: tale modulo consente la scrittura degli host Galileo all'interno del CMDB;
- **save_host.js**: tale modulo consente la scrittura di tutti gli host attivi sul sistema di monitoraggio all'interno del CMDB;
- **save_picasso.js**: tale modulo consente la scrittura dei service HSO JBOSS PICASSO con l'identificativo della macchina su cui è attivo all'interno del CMDB;
- **main.js**: è il modulo che viene eseguito per far funzionare il server Web;
- **config.csv**: è il file di configurazione che contiene le informazioni necessarie al server Web per svolgere i suoi compiti.

Per quanto riguarda i service in Skynet, invece, gli script realizzati sono:

- **check_remote_ssh.pl**: tale script consente di ottenere il numero di processori, il modello, la ram, il sistema operativo e l'hostname di una qualsiasi macchina Linux;

- **check_wmi2.pl**: tale script consente di ottenere il numero di processori, il modello, la ram, il sistema operativo e l'hostname di una qualsiasi macchina Windows;
- **check.pl**: tale script permette la realizzazione di tutti gli altri service;
- **notify_by_sysaid_application.pl**: tale script viene utilizzato per consentire la scrittura dei service all'interno del CMDB.

A ciascuno script, infine, è associato un file di configurazione:

- **host_sw.csv**: file di configurazione dello script check.pl;
- **host_ssh.csv**: file di configurazione dello script check_remote_ssh.pl;
- **host_wmi.csv**: file di configurazione dello script check_wmi2.pl;
- **sysaid_integration.cfg**: file di configurazione dello script notify_by_sysaid_application.pl.

Per quanto riguarda gli item in Zabbix, invece, questi sono stati raggruppati tutti all'interno di un unico script, in quanto lo scopo è verificare il funzionamento all'interno di un nuovo sistema di monitoraggio:

- **new_check.pl**: tale script permette la realizzazione di tutti gli item corrispondenti ai service di Skynet;
- **hosts.csv**: file di configurazione dello script new_check.pl.

## 5.3 – Realizzazione

In questo paragrafo verrà spiegato come il progetto è stato realizzato tenendo conto anche di parti di codice sorgente significativi.

### 5.3.1 – Server Web

Per quanto riguarda la realizzazione del server Web, bisogna considerare, in primo luogo, il file di configurazione *config.csv*. Tale file è strutturato nel seguente modo:

*SERVER,ip,port*

*ORACLE,username,password,ip/SID*

*SKYNET,ip,username,password,database,port*

*SYSAID,accountId,username,password*

*DATABASE,username,password*

*ZABBIX,ip,username,password,database,port.*

Come si può notare, si hanno diversi campi di configurazione ciascuno dei quali ha i proprio parametri. Ogni campo viene utilizzato in base alla funzionalità che si vuole svolgere con l'eccezione del primo che ha il compito di fornire i parametri per l'avvio del server; infatti, SERVER è la parola chiave che serve per individuare il campo che si vuole leggere, ip e port rappresentano l'indirizzo IP e la porta su cui il server rimane in ascolto. Per poter accedere al server Web, si può aprire un qualunque browser e inserire l'URL: http://ip_server:port_server; in questa circostanza si aprirà la pagina iniziale che permetterà all'utente di interagire con il server Web.

Secondo file molto importante è il *main.js*, in quanto rappresenta la logica applicativa del server e senza questo non potrà funzionare nulla.

La parte più significativa del *main.js* è la seguente:

```
var http=require('http');

var server = http.createServer(function(req,res){

        ...

})

server.listen(port, ip)
```

La prima cosa da fare è importare il modulo http che contiene diversi oggetti che permettono l'utilizzo del protocollo HTTP. Attraverso questo modulo, si è in grado di creare il server e, successivamente, di mettere il server in ascolto su una determinata porta e con un determinato indirizzo IP. Questa parte è fondamentale in quanto senza il server non esisterebbe. La porta e l'indirizzo IP del server vengono letti dal file di configurazione, pertanto un altro modulo da importare è *fs* che permette di lavorare con il filesystem della macchina, quindi consente di effettuare la lettura del file di configurazione per estrapolare l'indirizzo e la porta del server.

All'interno del blocco precedente, si troverà:

```
switch(req.url){
        case '/':
                ...
        case '/hosts':
                ...
        case '/services':
                ...
        case '/adt':
                ...
        case '/product':
                ...
        case 'database':
                ...
}
```

In questo caso, si va ad analizzare l'URL presente all'interno della richiesta HTTP. Nel primo caso, ci si riferisce alla pagina iniziale del server Web all'interno della quale si trovano i diversi controlli che possono essere effettuati. Ogni tasto Check è collegato ad un'action e a un metodo come segue:

```
res.write('<form enctype="application/x-www-form-urlencoded" action="/hosts" method="post">');
```

*res.write('<input type="submit" name="check" value="Check"/><br><br>');*

Questo piccolo pezzo di codice permette di capire che quando viene cliccato il tasto Check relativo agli host (per gli altri tasti basta cambiare l'action), all'interno della richiesta HTTP ci sarà l'URL *http://ip_server:port/hosts*. Questo rimanderà al *case '/hosts'* all'interno della quale verranno effettuate le operazioni. Al suo interno si troverà:

```
if(req.method == 'POST'){
    req.on('data', function(chunk){
        var p=new Promise(function(resolve, reject){
            resolve(chunk.toString());
        })
    });
    req.on('end',function(){
        p.then(function(mio){
            var click=mio.split("=");
            if (click[0] == "check"){
                ...
            }
            elsif(click[0] == "sync"){
                ...
            }
        })
    })
}
else {
    restituisce una pagina Web all'interno del quale c'è scritto Method not
    supported
}
```

In questa porzione di codice, si va a verificare prima se il metodo della richiesta è POST, perché se così non fosse allora verrà mostrata una pagina Web con scritto "Method not supported". L'oggetto req (Request) implementa l'interfaccia ReadableStream, pertanto il flusso dei dati può essere ascoltato o convogliato ovunque. In questo caso, si è in grado di catturare i giusti dati ascoltando gli eventi di stream 'data' e 'end'. Attraverso *req.on('data', function(chunck))* si è in grado di capire la richiesta effettuata, infatti, il chunk restituito sarà del tipo:

*check=Check → name=value*

oppure in caso di sincronizzazione:

*sync=Sync*.

Un aspetto importante e del tutto nuovo è rappresentato dalla Promise. Questa rappresenta il risultato di un'operazione asincrona, infatti, è in grado di catturare il risultato di tale operazione e di poterla riutilizzare come e quando si vuole. Per poterla utilizzare è necessario importare il corrispondente modulo attraverso *require('promise')*. È possibile notare come, quando viene definita una promise, vi è una funzione di callback che presenta due parametri: resolve e reject. Il primo è lo stato di una promise rappresentante un'operazione avvenuta con successo (in altre parole consente di catturare il risultato); il secondo è lo stato della promise rappresentate il fallimento dell'operazione. Attraverso la promise, viene catturato il chunk in modo da poterlo utilizzare nel blocco successivo, ovvero, *req.on('end', function())*. Questo termina l'invio della richiesta e, di conseguenza, al suo interno è possibile manipolarla. Attraverso la promise, viene restituito il chunk di cui viene estratta la prima parte, ovvero il *name*, per capire quale operazione eseguire, cioè se la richiesta fatta consiste nel fare un controllo oppure nel fare una scrittura all'interno del CMDB. Ultima cosa importante è che all'interno del blocco check si implementerà la logica per effettuare i controlli e si creerà la pagina Web che mostrerà il tutto e aggiungerà il tasto Sync per consentire la scrittura dei dati, ricordando che se non ci sono dati da scrivere verrà restituita la lista vuota e l'unica azione possibile sarà quella di tornare alla pagina iniziale. Al suo interno verranno importati tutti i moduli check realizzati, in quanto il loro compito è quello di restituire il risultato finale. All'interno del blocco Sync, invece, è contenuta la logica di implementazione per la scrittura dei dati. All'interno di questo blocco verranno importanti tutti i moduli save realizzati che consentiranno di effettuare la scrittura e di ottenere i CIId dei CI creati.

I blocchi di codice riportati rappresentano le parti fondamentali del file main.js; inoltre, anche se è stato preso in considerazione il controllo relativo agli host attivi sul sistema di monitoraggio, per tutti gli altri casi il discorso è esattamente lo stesso.

Dopo aver discusso dei file di "partenza" per la creazione ed esecuzione del server Web, bisogna analizzare i singoli moduli creati per capire come avviene il controllo e la scrittura dei dati.

I primi moduli da prendere in considerazione sono quelli che permettono di ottenere le informazioni di monitoraggio da Skynet, Zabbix e dai diversi database.

Per quanto riguarda il modulo *host.js* la prima cosa importante da dire è si tratta di un modulo user-define così come lo sono tutti gli altri.

Per creare un modulo personalizzato bisogna definire una funzione che include tutto il codice e poi effettuare l'esportazione del tutto per consentire il suo riutilizzo in altri moduli:

```
var hosts=function hosts(){

        ...

}

exports.hosts=hosts
```

Come si può vedere dal codice, per la creazione di un modulo personalizzato, non bisogna fare altro che definire la funzione all'interno della quale ci sarà la sua logica implementativa e poi esportare il tutto per consentirne l'utilizzo. Per importare il modulo:

```
var host=require('/root/node_js/host.js');

var h=host.hosts();
```

la prima cosa da fare è importare il modulo creato attraverso il metodo *require* specificando il path completo. Fatto questo, bisogna invocare la funzione realizzata all'interno del modulo.

All'interno della funzione, la prima cosa da fare è effettuare la lettura del file di configurazione per ottenere i parametri di connessione al database di Skynet o Zabbix:

```
var fs=require('fs');

var content =fs.readFileSync('/root/node_js' + process.argv[2], 'utf8');
```

```
var rows = content.split("\n");
for(var i in rows){
    var params=rows[i].split(",");
    if(params[0] == "SKYNET"){
        host=params[1];
        user=params[2];
        pwd=params[3];
        db=params[4];
        port=params[5];
    }
}
```

Questo blocco consente la lettura del file di configurazione e di estrapolare i parametri che sono di interesse. Prima cosa da fare è importare il modulo *fs* per poter effettuare operazioni di lettura. A questo punto, viene invocata la funzione *readFileSync* che consente di effettuare la lettura del file in modo sincrono; tale funzione prende in ingresso il path del file di configurazione e la sua codifica. In questo caso, il file viene passato attraverso la riga di comando pertanto viene utilizzato il *process.argv[2]*. Effettuata la lettura, bisogna considerare il campo che contiene i parametri relativi a Skynet o Zabbix in modo da ottenere indirizzo IP, username, password, database e porta. Il prossimo passo da fare è il seguente:

```
var mysql=require('mysql');
var connection=mysql.createConnection({
    host        : host,
    user        : user,
    password    : pwd,
    database    : db,
    port        : port
});
connection.connect();
```

*connection.query('select name, alias, address from hosts;', function(err, rows, fields)*
*{*

*connection.destroy();*

*if (err) return reject(err);*

*resolve(rows);*

*})*

Come si può vedere dal blocco del codice, la prima cosa da fare è importare il modulo *mysql* che permette di interagire con database di questo tipo tenendo presente che i database di Skynet e Zabbix sono MySQL. Successivamente, viene creata la connessione, viene effettuata la connessione, viene eseguita la query e viene distrutta la connessione. Come si può notare sono presenti resolve e reject; questo perché per ogni modulo è stata utilizzata la promise. Questa è risultata fondamentale per gestire l'asincronismo a cui Node.js sottopone i suoi sviluppatori. Il codice, sopra mostrato, si riferisce al caso in cui si vuole interagire con Skynet; se si volesse interagire con Zabbix, invece, non bisogna fare altro che sostituire la parola chiave SKYNET con ZABBIX e modificare la query in:

*select h.name as name, h.description as alias, i.ip as address from hosts h, interface i where h.hostid=i.hostid and h.status=0;*

il cambiamento della query è necessario in quanto ogni database avrà le sue tabelle e ciascuna di esse avrà le sue colonne con i dati formattati in un determinato modo.

Il modulo successivo è il *services_picasso.js*. Per la definizione del modulo si avrà:

*var service=function services(){*

*...*

*}*

*exports.service=service.*

In questa circostanza vengono fatte esattamente le stesse cose del modulo host.js, compresa la lettura dello stesso campo del file di configurazione. L'unica differenza è rappresentata dalla query che viene eseguita.

Se si decide di operare con Skynet si avrà che:

*params[0] == "SKYNET"*

e la query da eseguire è:

*select a.name as service, c.host_id as host_id, c.name as host from services_name a inner join services b on b.servicename_id=a.servicename_id inner join hosts c on b.host_id=c.host_id where a.command_line= "check_jmx4perl_config!ARG1!ARG2! ARG3!ARG4".*

Nel caso si decide di lavorare con Zabbix, invece:

*params[0] == "ZABBIX"*

e la query da eseguire sarà:

*select i.name as service, h.name as host, h.hostid as host_id from items i, hosts h where i.hostid=h.hostid and h.status=0 and i.key_ like \'%check_jmx4perl%\'.*

Per quanto riguarda la creazione del modulo *adt.js*, questo rappresenta un caso particolare in quanto al suo interno sono presenti due funzioni; pertanto si avrà:

```
var ids=function getId(){
        ...
}
var parameters = function getParameters(id){
        ...
}
exports.ids=ids;
exports.parameters=parameters;
```

Come si può notare dalla porzione di codice, vengono definite due funzione che entrambe devono essere esportate per consentirne l'utilizzo in altri moduli. La prima funzione ha il compito di restituire gli id dei service sender, mentre la seconda funzione restituisce i collegamenti fra i service sender e quelli receiver e di conseguenza i middleware.

Prima della definizione delle due funzioni, viene effettuata la lettura del file di configurazione. In questo caso, come campo del file, viene passato Oracle:

*params[0] == "ORACLE"*

questo per il semplice motivo che le informazioni di interesse sono contenute all'interno di questo database; di conseguenza non vi è differenza fra Skynet e Zabbix proprio perché il monitoraggio avviene al di fuori dei due sistemi.

All'interno di ciascuna funzione, viene creata una connessione al database, viene effettuata la connessione, eseguita la query ed, infine, la connessione viene distrutta. La funzione *getParameters* dipende dall'altra funzione in quanto prende come parametro di ingresso gli id restituiti dalla *getId*.

La query che viene effettuata nella *getId* è la seguente:

*select id, msg_tp_code, app_code from P4_CH where active=1 and ch_tp_code='SND' and msg_tp_code <> 'ANYHL7V2' order by id asc;*

quella eseguita nella *getParameters(id)* è:

*conn.execute("select app_code from p4_ch where id in (select dst_id from p4_ch_map where src_id in (select dst_id from p4_ch_map where* ***src_id =:nm****))",* ***[id[0]]****, function(err,res){*

...

*}).*

In questo caso è stato riportata tutta la funzione e non solo la query in quanto vi è un aspetto particolare che prende il nome di *Bind Variables* (in grassetto). La Bind Variables è uno dei più importanti concetti per quanto riguarda la connessione a database Oracle, in quanto migliora radicalmente le performance delle query effettuate sul database. Nel caso classico, quando si esegue una stessa istruzione SQL più di una volta, ma con parametri differenti nella clausola where, si forza il database ad analizzare la query ogni volta che questa viene eseguita; con le Bind Variables cambia completamente l'approccio perché permettono di scrivere un'istruzione SQL che accetta parametri in fase di esecuzione.

Per quanto riguarda i moduli *host_galileo.js*, *host_halia.js* e *host_dnweb.js*, questi possono essere considerati insieme in quanto le modifiche sono veramente lievi.

La loro creazione prevede la seguente funzione:

*var hosts=function hosts(){*

...

*}*

*exports.hosts=hosts;*

all'interno di questa funzione, viene effettuata la lettura del file di configurazione per estrarre i parametri di accesso al database di Skynet, quindi si avrà:

*params[0] == "SKYNET"*

successivamente, verrà effettuata la connessione al database, come visto nel modulo *host.js* e *services_picasso.js* e le query da effettuare saranno:

- per il modulo ***host_galileo.js***: *select name from hosts where hostextinfo_id = (select hostextinfo_id from extended_host_info_templates where name="galileo-application");*
- per il modulo ***host_halia.js***: *select name from hosts where hostextinfo_id = (select hostextinfo_id from extended_host_info_templates where name="halia-application");*
- per il modulo ***host_dnweb.js***: *select name from hosts where hostextinfo_id = (select hostextinfo_id from extended_host_info_templates where name="dnweb-application")*.

Per quanto riguarda Zabbix, invece:

*params[0] == "ZABBIX"*

mentre le query saranno:

- per il modulo ***host_galileo.js***: *select h.name as name from hosts h, groups g, hosts_groups hg where hg.hostid=h.hostid and hg.groupid=g.groupid and h.status=0 and g.name=\'galileo-server\';*
- per il modulo ***host_halia.js*** : *select h.name as name from hosts h, groups g, hosts_groups hg where hg.hostid=h.hostid and hg.groupid=g.groupid and h.status=0 and g.name=\'halia-server\';*
- per il modulo ***host_dnweb.js***: *select h.name as name from hosts h, groups g, hosts_groups hg where hg.hostid=h.hostid and hg.groupid=g.groupid and h.status=0 and g.name=\'dnweb-server\'.*

Ultimo modulo di monitoraggio rimasto è il *database.js*. Per la definizione di questo modulo si ha:

*var db=function getDb(){*

*...*

*}*

*exports.db=db.*

In questa circostanza, viene effettuata una doppia lettura di file: il primo file letto è quello di configurazione per ottenere le informazioni necessarie per accedere al database; in questo caso, quando si esegue la lettura si avrà:

*params[0] == "DATABASE";*

la seconda lettura che viene effettuata è quella del file */etc/tnsnames.ora* ricordando che questo modulo ha il compito di restituire tutti quei database con cui la macchina, su cui esegue il server Web, può collegarsi.

All'interno del file *tnsnames.ora* ci saranno diversi database a cui ci si può collegare; pertanto il file viene letto, parsato in modo da ottenere le informazioni necessarie per ciascun database e poi viene eseguita la connessione per estrarne la versione.

Per ogni database individuato nel file:

```
db.connection({
        user            : user,
        password        :pwd,
        connectString   : cons
}, function (err, conn){
        conn.execute("select * from v$instance", function(err, res){
                ...
        })
})
```

In questo caso, ci si collega al database per estrarne la versione.

Terminati i moduli per il monitoraggio di parametri di interesse, si considerano quelli che permettono di interagire con SysAid e in particolar modo con il CMDB.

Il modulo di partenza è rappresentato da *login.js* che permette all'utente di loggarsi presso SysAid e accedere alle diverse funzionalità messe a disposizione. In questo caso, si va a leggere il file di configurazione per estrarre i parametri relativi al campo SYSAID, ovvero accountId, username e password. Fatto questo, si esegue il login

che restituisce il sessionID per poter sfruttare le API messe a disposizione. La prima cosa da fare è definire la funzione per la realizzazione del modulo:

```
var login=function log (client){
    ...
}
exports.login=login.
```

Come si può notare, alla funzione viene passato, come parametro di ingresso, il SOAP client che consente di poter eseguire le operazioni.

All'interno della funzione si troverà:

```
var args={
    accountId: accId,
    userName: user,
    password: pwd
};
client.login(args, function(err, sId){
    ...
});
```

Vengono definiti gli argomenti necessari per effettuare il login, parametri che vengono estratti dal file di configurazione. Successivamente, viene fatto il login che restituisce il sessionId (sID) per poter accedere alle funzioni.

Per quanto riguarda i moduli *executeSelectQuery_type_server.js, executeSelectQuery_type_database.js, executeSelectQuery_type_dnweb.js, executeSelectQuery_type_galileo.js, executeSelectQuery_type_halia.js* e *executeSelectQuery_type_middleware.js*, questi eseguono le stesse operazioni, ma cambia la condizione di ricerca. Per la definizione di ciascun modulo viene definita la funzione:

```
var esq=function executeSelectQuery(client, sId){
    ...
}
```

*exports.esq=esq*.

Alla funzione vengono passati come parametri di ingresso il SOAP client che consente di effettuare le operazioni e il sessionId che consente il login all'interno di SysAid.

All'interno della funzione vengono definiti gli argomenti per consentire l'esecuzione della funzione:

```
var args={
        sessionId: sId,
        apiSysObj: {
                $attributes : {
                        "xsi:type" : "tns:apiCI"
                }
        },
        condition: varia a seconda del modulo
}
client.executeSelectQuery(args, function(err,res){
        ...
})
```

In questa porzione di codice, si può constatare come vengono definiti gli argomenti necessari per effettuare la seguente funzione, ricordando che la executeSelectQuery permette di ottenere la lista di CIId di tutti quei CI che soddisfano la condizione. Questa varia da modulo a modulo:

- per il modulo ***executeSelectQuery_type_server.js*** la condizione sarà: *condition: "company=" + company + "and ci_type=\'102\'"*;
- per il modulo ***executeSelectQuery_type_database.js*** la condizione sarà: *condition: "company=" + company + "and ci_type=\'104\' and ci_sub_type=\'133\'"*;

- per il modulo ***executeSelectQuery_type_dnweb.js*** la condizione sarà: *condition: "company=" + company + "and ci_type=\'100\' and ci_sub_type=\'122\'"*;
- per il modulo ***executeSelectQuery_type_galileo.js*** la condizione sarà: *condition: "company=" + company + "and ci_type=\'100\' and ci_sub_type=\'119\'"*;
- per il modulo ***executeSelectQuery_type_halia.js*** la condizione sarà: *condition: "company=" + company + "and ci_type=\'100\'and ci_sub_type=\'117\'"*;
- per il modulo ***executeSelectQuery_type_middleware.js*** la condizione sarà: *condition: "company=" + company + "and ci_type=\'105\' and ci_sub_type=\'210\'"*.

Altro modulo per l'interazione con SysAid è *loadByStringId* che permette di ottenere il contenuto completo di un CI. Per definire tale modulo, la funzione è la seguente:

```
var lbsi=function loadByStringId(client, sId, CIId){
    ...
}
exports.lbsi=lbsi.
```

Questa funzione ha come parametri di ingresso il SOAP client che consente di effettuare le operazione, il sessionId per il login e il CIId che rappresenta l'identificatore del CI si cui si vogliono ottenere le informazioni. In questo caso si ha:

```
var args = {
    sessionId: sId,
    apiSysObj : {
        $attributes : {
            "xsi:type" : "tns:apiCI"
        }
    },
    id: CIId
};
```

*client.loadByStringId(args,function(err,res){*

*...*

*})*

Definiti gli argomenti, questi vengono passati alla funzione eseguita dal SOAP client fornendo tutte le informazioni relativi al CI.

Ultimi moduli per l'interazione con SysAid sono *sava_adt.js, save_database.js, save_dnweb.js, save_galileo.js, save_halia.js, save_host.js* e *save_picasso.js*. Tutti i moduli esposti si differiscono per i parametri che vanno a scrivere all'interno del CMDB, ma dal punto di vista del codice sono simili. Per la definizione dei moduli si ha:

*var save = function save (parameters){*

*...*

*}*

*exports.save=save*.

In questo caso, alla funzione vengono passati dei parametri di ingresso che cambiano in base al modulo a cui si fa riferimento. Successivamente, bisogna importare il modulo soap attraverso la *require*. Anche se sono stati definiti diversi moduli che permettono di interagire con SysAid, questa è la prima volta che viene richiesto il modulo soap. Il motivo è di facile comprensione in quanto i moduli a loro volta vengono utilizzati da altri moduli che avranno il modulo già importato, quindi, non sempre è necessario importarlo perché potrebbe trovarsi altrove.

Detto questo: si avrà:

*var soap=require('soap');*

*var url='http://awssysaidtest.noemalife.loc:80/services/SysaidApiService?wsdl';*

*var wsdlOptions = {*

*...*

*};*

*soap.createClient(url, wsdlOptions, function(err, client) {*

*var promlog=log.login(client);*

```
promlog.then(function(res){
        var args = {
                ...
        }
}
client.save(args,function(err, res){
})
```

Per effettuare la scrittura all'interno del CMDB, devono essere definiti le wsdlOptions. Queste vengono utilizzate per consentire la creazione del SOAP client, all'interno del quale viene effettuata un'operazione di login per poter accedere a SysAid, vengono definiti gli argomenti necessari ed, infine, viene effettuata la scrittura. Per comprendere al meglio informazioni relative al concetto SOAP nel mondo Node.js è possibile consultare la documentazione[6].

Per ogni modulo realizzato, le informazioni che cambiano sono i parametri passati in ingresso alla funzione per la creazione del modulo e quelli definiti all'interno di *args*. In questo caso si ha:

- per il modulo ***save_adt.js***: i parametri di ingresso sono il nome del CI, un campo personalizzato chiamato nagios_id e il campo notes. Per quanto il campo *args* al suo interno si troveranno i parametri di ingresso, *ciType: 105, ciSubType: 210, status: 1 e company: company;*
- per il modulo ***save_database.js***: i parametri di ingresso sono il nome del CI, versione del database, porta e SID. Quelli presenti nel campo *args* sono quelli appena elencati con l'aggiunta di *ciType: 104, ciSubType: 133, status: 1 e company: company*;
- per il modulo ***save_dnweb.js***: il parametro di ingresso è il nome del CI, mentre in *args* oltre ad inserire il nome, viene inserito anche il *ciType: 100, ciSubType: 122, status: 1, company: company*;
- per il modulo ***save_halia.js***: il parametro di ingresso è il nome del CI, mentre in *args* oltre ad inserire il nome, viene inserito anche il *ciType: 100, ciSubType: 117, status: 1, company: company*;

- per il modulo ***save_galileo.js***: il parametro di ingresso è il nome del CI, mentre in *args* oltre ad inserire il nome, viene inserito anche il *ciType: 100, ciSubType: 119, status: 1, company: company*;
- per il modulo ***save_host.js***: i parametri di ingresso sono il nome del CI, alias e indirizzo IP, mentre in *args* oltre ad inserire i parametri elencati, viene inserito anche il *ciType: 102, ciSubType: 159, status: 1, company: company*;
- per il modulo ***save_picasso.js***: i parametri di ingresso sono il nome del CI e il nagios_id, mentre in *args* oltre ad inserire i parametri elencati, viene inserito anche il *ciType: 105, ciSubType: 210, status: 1, company: company.*

Per quanto riguarda i moduli che consentono di verificare quali informazioni di monitoraggio sono presenti nel CMDB, questi sono molto simili fra loro, ma importano moduli creati differenti. La funzione che permette di definire ciascun modulo è:

*var nagios_id=function nagios_id(){*

    *...*

*}*

*exports.nagios_id=nagios_id*.

Il nagios_id è un parametro creato dall'azienda che lo considera come un identificatore unico per ciascun componente registrato nel CMDB. Le operazioni svolte all'interno di questa funzione sono:

*soap.createClient(url, wsdlOptions, function(err,client){*

    *...*

*})*

Questa operazione permette di creare il SOAP client, il quale eseguirà le diverse funzionalità. All'interno di questa funzione, verrà fatta l'operazione di login, la executeSelectQuery per ottenere la lista di CIId che hanno determinati parametri e la loadByStringId per ottenere il contenuto di ciascun CI. Del contenuto, viene restituito solo il campo nagios_id (identificato con custTextFieald10 nella terminologia di SysAid) che permette di effettuare i confronti e sapere cosa non è presente all'interno del CMDB. In questa circostanza, i moduli hanno in comune l'importazione del modulo *login.js* e *loadByStringId*, mentre si differenziano per il modulo *executeSelectQuery*; infatti:

- per il modulo ***nagios_id.js*** viene importato il modulo *executeSelectQuery_type_server.js*;
- per il modulo ***nagios_id_database.js*** viene importato il modulo *executeSelectQuery_type_database.js*;
- per il modulo ***nagios_id_dnweb.js*** viene importato il modulo *executeSelectQuery_type_dnweb.js*;
- per il modulo ***nagios_id_galileo.js*** viene importato il modulo *executeSelectQuery_type_galileo.js*;
- per il modulo ***nagios_id_halia.js*** viene importato il modulo *executeSelectQuery_type_halia.js*;
- per il modulo ***nagios_id_middleware.js*** viene importato il modulo *executeSelectQuery_type_middleware.js.*

Gli ultimi moduli rimasti sono quelli che si occupano di prendere le informazioni di monitoraggio e quelle provenienti dal CMDB e confrontarli per sapere quali possono essere memorizzati. Per la definizione di ciascuno modulo, viene creata la seguente funzione:

```
var check=function check(){
    ...
}
exports.check=check.
```

All'interno di questa funzione vengono confrontati i nagios_id per sapere quali dover scrivere. Anche in questa circostanza, i diversi moduli di check si differenziano per l'importazione dei moduli creati, infatti:

- per il modulo ***check_host.js*** vengono importati i moduli *nagios_id.js* e *host.js*;
- per il modulo ***check_adt.js*** vengono importati i moduli *nagios_id_middleware.js* e *adt.js*;
- per il modulo ***check_database.js*** vengono importati i moduli *nagios_id_database.js* e *database.js*;
- per il modulo ***check_dnweb_host.js*** vengono importati i moduli *nagios_id_dnweb.js* e *host_dnweb.js*;

- per il modulo ***check_galileo_host.js*** vengono importati i moduli *nagios_id_galileo.js* e *host_galileo.js*;
- per il modulo ***check_host_halia.js*** vengono importati i moduli *nagios_id_halia.js* e *host_halia.js*;
- per il modulo ***check_picasso.js*** vengono importati i moduli *nagios_id_middleware.js* e *services_picasso.js*.

### 5.3.2 – Service Skynet e Item Zabbix

Per quanto riguarda la realizzazione dei service in Skynet, chiamati item nella terminologia di Zabbix, sono stati creati diversi script in linguaggio Perl.

Il primo file che si può prendere in considerazione è il *check_remote_ssh.pl* che permette di realizzare service che hanno il compito di monitorare le informazioni di base di una macchina Linux:

1. HSO OPERATING SYSTEM REMOTE SSH CPU NUMBER;
2. HSO OPERATING SYSTEM REMOTE SSH CPU MODEL;
3. HSO OPERATING SYSTEM REMOTE SSH RAM;
4. HSO OPERATING SYSTEM REMOTE SSH OS;
5. HSO OPERATING SYSTEM REMOTE SSH HOSTNAME.

Tutti questi service si ottengono più o meno allo stesso modo, l'unica cosa che cambia è il comando che viene passato; infatti, per il primo service il command è il seguente:

*my $command=qq{cat /proc/cpuinfo | grep -c processor};*

per il secondo:

*my $command=qq{cat /proc/cpuinfo | grep "model name" | cut -d: -f2};*

per il terzo:

*my $command=qq{cat /proc/meminfo | grep MemTotal | rev | cut -d" " -f2 | rev};*

per il quarto:

*my $command=qq{lsb_release -a | grep Description | cut -d":" -f2 | tr -d "\t"};*

per il quinto:

*my $command=qq{hostname}.*

Una volta definiti i diversi comandi da eseguire per ciascun service, non bisogna fare altro che passarlo alla funzione *exec_ssh_cmd* che ha il compito di eseguire il comando creato sulla macchina monitorata. Infatti, si avrà:

*my %command_result = exec_ssh_cmd($configuration{$opt_H}{IP}, $configuration{$opt_H}{US},$configuration{$opt_H}{PW},$command);*

I parametri IP, US (username) e PW (password) vengono letti dal file di configurazione *hosts_ssh.csv* e si riferiscono all'indirizzo della macchina di cui si vogliono monitorare certi parametri, lo username e la password per accedere alla macchina stessa. La funzione *exec_ssh_cmd* è realizzata in questo modo:

*my $ssh = Net::SSH::Perl→new($l_host);*

creazione dell'istanza SSH per poter usufruire di questo servizio;

*$ssh->login($l_user,$l_pass) or die qq{CRITICAL - Unable to login to $l_host (using $l_user/$l_pass)\n};*

viene effettuato il login per poter avere l'accesso alla macchina;

*my @retval = $ssh→cmd($l_cmd);*

esecuzione del comando realizzato ottenendo come risultato un array di parametri. Tale array viene elaborato per poter ottenere, alla fine, l'informazione di interesse.

Ultimo aspetto importante per tutti i service è la formattazione dell'output. Quando un service esegue su un host monitorato da Skynet, l'output deve restituire lo stato del service oltre che il risultato; pertanto, bisogna formattare l'output in modo da consentire a Nagios di poterlo elaborare:

```
if($command_result{EXT} != 0 || $result eq ""){
        $output=q{CRITICAL};
}
else{
        $output=q{OK};
}
```

in questo caso si controlla il risultato restituito dal comando specifico di ciascun service settando l'output in maniera appropriata. Fatto questo è possibile restituire il

tutto in modo da consentire al sistema di monitoraggio la corretta elaborazione. Per il primo service si avrà:

*print qq{$output - CPU NUMBER: $result | 'CPU NUMBER'=$result;\n};*

per il secondo:

*print qq{$output – CPU MODEL: @array | 'CPU MODEL $n'=$model[$#model];\n};*

per il terzo*:*

*print qq{$output - RAM SIZE: $res $unit | 'SIZE'=$res $unit;\n};*

per il quarto:

*print qq{$output - OS: $result | 'OS'=$result;\n};*

ed, infine, per il quinto:

*print qq{$output - HOSTNAME: $result | 'HOSTNAME'=$result;\n}*.

Un altro script realizzato è quello che prende il nome di *check_wmi2.pl* che permette di realizzare service che hanno il compito di monitorare le informazioni di base di una macchina Windows:

1. HSO OPERATING SYSTEM OS;
2. HSO OPERATING SYSTEM CPU NUMBER;
3. HSO OPERATING SYSTEM CPU MODEL;
4. HSO OPERATING SYSTEM RAM;
5. HSO OPERATING SYSTEM HOSTNAME.

Anche in questa circostanza, i service si ottengono tutti più o meno allo stesso modo, ciò che cambia è il comando che viene passato per ottenere una specifica informazione. Per il primo service si ha:

*my $l_query = qq/select Name from Win32_OperatingSystem/;*

per il secondo:

*my $l_query=qq{select NumberOfCores, NumberOfLogicalProcessors from Win32_Processor};*

per il terzo:

*my $l_query=qq{select name from Win32_Processor};*

per il quarto:

*my $l_query=qq{select MaxCapacity from Win32_PhysicalMemoryArray};*

ed, infine, per il quinto:

*my $l_query=qq{select name from Win32_ComputerSystem}*.

Una volta che sono stati definiti i comandi da eseguire, ciascuno di questo, per ogni service, verrà passato ad una funzione che prende il nome di *wmi_cmd*. Tale funzione ha il compito di collegarsi alla macchina Windows monitorata e di eseguire il comando in modo da estrarre le informazioni richieste. Tale funzione prende come parametri di ingresso username, password, hostname e query, di cui i primi tre vengono letti dal file di configurazione *hosts_wmi.csv*, mentre la query è il comando che viene realizzato per ogni specifico servizio. Tale funzione è molto semplice, infatti:

*my $cmd = qq{wmic -U $l_user\%$l_pass //$l_host "$l_query"};*

*my $output = `$cmd 2>&1`;*

come si può notare non fa altro che utilizzare wmic per poter eseguire il comando ed, infine, restituisce il risultato.

Una volta che si sono ottenute le informazioni di interesse, bisogna formattare l'output in modo da consentire al sistema di monitoraggio di analizzalo. Per il primo service si avrà:

*print qq{$output - OS VERSION: $res |'OS'=$res;\n};*

per il secondo:

*print qq{$output - CPU PHYSICAL: $num[$#num-1] CPU LOGICAL: $num[$#num] | 'CPU PHYSICAL'=$num[$#num-1]; 'CPU LOGICAL'=$num[$#num]\n};*

per il terzo:

*print qq{$output - CPU MODEL: $name[$#name] | 'CPU MODEL'=$name[$#name];\n};*

per il quarto:

*print qq{$output - RAM SIZE: $res $unit | 'RAM SIZE'=$res $unit;\n};*

ed, infine, per il quinto:

*print qq{$output - HOSTNAME: $array[$#array] | 'HOSTNAME'=$array[$#array];\n}.*

Altro script realizzato è quello che prende il nome di *check.pl* che permette di realizzare una molteplicità di service: quelli che permettono di arricchire le informazioni del CI contenuto all'interno del CMDB, quelli che permettono di estrarre informazioni di monitoraggio provenienti da JavaMelody ed, infine, quelli che permettono di estrarre utili informazioni dal prodotto Galileo e Halia. Una cosa importante da dire è che i service, che permettono di ottenere indicatori dell'uso della soluzione informatizzata NoemaLife per svolgere le principali operazioni cliniche e diagnostiche, sono stati ottenuti attraverso un plugin di Nagios chiamato *check_oracle_health* scaricato da internet. Questo script, quindi, non è stato realizzato, ma si è cercato di capirne il funzionamento; infatti, questo plugin permette di controllare diversi parametri di un database Oracle ed è molto utilizzato dall'azienda.

Ritornando allo script *check.pl*, vediamo come è stato realizzato e quali service permette di creare.

Il service GALILEO VERSION permette di ottenere la versione del prodotto Galileo installato sulla macchina che si sta monitorando. Per ottenerlo è stata realizzata una funzione chiamata *find_version* che come parametri di ingresso prende l'indirizzo IP della macchina, la porta e il file generato dalla macchina quando il prodotto viene installato e/o aggiornato. Tale funzione ottiene la versione in questo modo:

*my $url="http://$ip:$port/$file";*

*my c=get($url).*

Ottenuto il file, in seguito a delle elaborazioni, si ottiene la versione che poi viene scompattata in HighVersion, MainVersion, Patch e Hotfix. All'interno del main, nella sezione relativa a questo service, si avrà la formattazione dell'output per consentire a Nagios di elaborare il risultato:

*print qq{$output - VERSION GALILEO $res{"HighVersion"}.$res{"MainVersion"}. $res{"Patch"}.$res{"Hotfix"} | 'HighVersion'=$res{"HighVersion"}; 'MainVersion'=$res{"MainVersion"}; 'Patch'=$res{"Patch"}; 'Hotfix'=$res{"Hotfix"};\n};*

ed, infine, la scrittura all'interno del CMDB viene fatta in questo modo:

*`perl /usr/local/nagios/libexec/notify_by_sysaid_application.pl -H $opt_H -m $output -w "$output - VERSION GALILEO $res{"HighVersion"}. $res{"MainVersion"}.$res{"Patch"}.$res{"Hotfix"}" -r "$opt_r"`.*

Come si può notare, viene citato un altro script realizzato che prende il nome di *notify_by_sysaid_application.pl* che ha il compito di effettuare la scrittura. La sua realizzazione verrà affrontata più avanti.

Il service GALILEO MEMORY permette di ottenere particolari parametri di "memoria" del prodotto Galileo installato sulla macchina che si sta monitorando. Per questo service è stata realizzata una funzione che prende il nome di *find_mempar* che prende in ingresso l'indirizzo IP della macchina, la porta, lo username e la password. Tale funzione estrae i parametri di interesse in questo modo:

*my $req=HTTP::Request->new(GET => "http://$ip:$port/webmed/monitoring?jmxValue=java.lang:type=Runtime.InputArguments");*

quindi viene fatta una richiesta HTTP. Successivamente, si passano i parametri di autorizzazione per l'accesso:

*$req→authorization_basic($user,$pwd);*

ed, infine, si ottengono i parametri:

*my $ua=LWP::UserAgent->new;*

*my $res=$ua→request($req);*

quindi, viene prima definito un nuovo userAgent e poi viene estratta la risposta dalla richiesta effettuata.

Se la risposta ha avuto successo:

*if ($res→ is_success){*

*elaborazione del risultato con estrazione dei parametri di memoria*

*}.*

Se tutto va per il verso giusto, è possibile estrarre ed elaborare il risultato.

All'interno del main, nella sezione relativa a questo service, viene formattato l'output da restituire al sistema di monitoraggio per consentirne l'analisi:

*print qq{$output - Xms=$result[0]MB Xmx=$result[1]MB XX:MaxPermSize=$result[2]MB | 'Xms'=$result[0]MB; 'Xmx'=$result[1]MB; 'XX:MaxPermSize'=$result[2]MB;\n};*

ed, infine, viene effettuata la scrittura all'interno del CMDB:

*`perl /usr/local/nagios/libexec/notify_by_sysaid_application.pl -H $opt_H -m $output -w "$output - Xms=$result[0]MB Xmx=$result[1]MB XX:MaxPermSize=$result[2]MB" -r "$opt_r"`.*

Il service HALIA VERSION permette di ottenere la versione di un altro prodotto aziendale. Questo viene fatto attraverso la funzione *find_version_halia* che prende come parametro di ingresso il nome del database da cui estrarre le informazioni. Questa funzione estrae la versione del prodotto attraverso una particolare funzione chiamata *sqlrsh*:

*my $name_version=`/usr/local/firstworks/bin/sqlrsh -id $db -command \"select * from v_sk_halia_vers;\"`.*

Il comando *sqlrsh* è un tool interattivo per eseguire query verso un database molto simile a sqlplus, psql, mysql e molti altri ancora. Attraverso questo comando, si è in grado di effettuare una query all'interno del database ottenendo immediatamente il risultato senza dovervi accedere esplicitamente. Ottenuta la versione del prodotto, questa viene scompattata in HighVersione, MainVersion, Patch e Hotfix.

All'interno del main, nella sezione relativa a questo service, viene formattato l'output per l'elaborazione da parte di Nagios:

*print qq{$output - VERSION HALIA $result{"HighVersion"}.$result{"MainVersion"}.$result{"Patch"}.$result{"Hotfix"} | 'HighVersion'=$result{"HighVersion"}; 'MainVersion'=$result{"MainVersion"}; 'Patch'=$result{"Patch"}; 'Hotfix'=$result{"Hotfix"};\n};*

e, successivamente, viene effettuata la scrittura presso il CMDB:

*`perl /usr/local/nagios/libexec/notify_by_sysaid_application.pl -H $opt_H -m $output -w "$output - VERSION HALIA $result{"HighVersion"}.$result{"MainVersion"}.$result{"Patch"}.$result{"Hotfix"}" -r "$opt_r"`.*

Il service DEVICE CONFIGURATION HALIA restituisce la lista dei dispositivi medicali collegati al prodotto Halia. A tale scopo, è stata realizzata la funzione

*find_devices_config* che prende in ingresso il nome del database. Tale funzione consente di estrarre i dispositivi attraverso una query verso il database Halia:

*my $l_list=`/usr/local/firstworks/bin/sqlrsh -id $db -command \"select dc.device_description from haldev.devices_conf dc;\"`.*

Ottenuti i parametri di interesse, viene formattato l'output per Nagios:

*print qq{$output - LIST DEVICES:$res | value$n=$result[$n-1];\n};*

ed, infine, viene effettuata la scrittura presso il CMDB:

*`perl /usr/local/nagios/libexec/notify_by_sysaid_application.pl -H $opt_H -m $output -w "$output - LIST DEVICES:$res" -r "$opt_r"`.*

I service AMBIENTE JAVA DNWEB, APACHE TOMCAT DNWEB e OE LIS DNWEB ottengono le informazioni di interesse attraverso la stessa funzione chiamata *find_par_dnweb*, che prende in ingresso l'indirizzo IP della macchina monitorata e la porta. Tale funzione estrae le informazioni attraverso un URL:

*open my $input, "-|", "wget -O - http://$ip:$port/servlet/it.dianoema.dnweb.dnlis.servlets.About 2>/dev/null";*

*while (<$input>){*

*estrazione ed elaborazione dei parametri*

*}*

In questo caso, attraverso delle condizioni if vengono gestiti i diversi service. Poiché si parla di versione, queste vengono scompattate in HighVersion, MainVersion, Patch e Hotfix. Ottenuto ciò che si desidera, si passa alla formattazione dell'output per consentire a Nagios di elaborare i dati. Per il service OE LIS DNWEB:

*print qq{$output - VERSION OE LIS DNWEB $result[0]{"HighVersion"}.$result[0]{"MainVersion"}.$result[0]{"Patch"} | 'HighVersion'=$result[0]{"HighVersion"}; 'MainVersion'=$result[0]{"MainVersion"}; 'Patch'=$result[0]{"Patch"};\n};*

e la scrittura:

*`perl /usr/local/nagios/libexec/notify_by_sysaid_application.pl -H $opt_H -m $output -w "$output - - VERSION OE LIS DNWEB $result[0]{"HighVersion"}.$result[0]{"MainVersion"}.$result[0]{"Patch"}" -r "$opt_r"`;*

per il service APACHE TOMCAT DNWEB:

*print qq{$output - VERSION APACHE TOMCAT $result[1]{"HighVersion"}. $result[1]{"MainVersion"}.$result[1]{"Patch"} | 'HighVersion'=$result[1] {"HighVersion"}; 'MainVersion'=$result[1]{"MainVersion"}; 'Patch'=$result[1] {"Patch"};\n};*

e la scrittura:

*`perl /usr/local/nagios/libexec/notify_by_sysaid_application.pl -H $opt_H -m $output -w "$output - VERSION APACHE TOMCAT $result[1]{"HighVersion"}. $result[1]{"MainVersion"}.$result[1]{"Patch"}" -r "$opt_r"`;*

per il service AMBIENTE JAVA DNWEB:

*print qq{$output - VERSION AMBIENTE JAVA $result[2] | 'Version'=$result[2];\n};*

per la scrittura:

*`perl /usr/local/nagios/libexec/notify_by_sysaid_application.pl -H $opt_H -m $output -w "$output - VERSION AMBIENTE JAVA $result[2]" -r "$opt_r"`.*

Ultimo service che viene utilizzato per il popolamento del CMDB è il PEOPLE VERSION che consente di ottenere la versione di un altro prodotto NoemaLife chiamato People. Per tale service è stata realizzata una funzione chiamata *find_people* che prende in ingresso l'indirizzo IP e la porta della macchina monitorata. Per ottenere la versione:

*my $url="http://$ip:$port/people/People.jnlp";*

*my $c = get($url);*

Grazie a questo blocco, si ottiene la versione del prodotto che successivamente viene scompattata in HighVersion, MainVersion, Patch e Hotfix. All'interno del main, nella sezione relativa a questo service, viene formattato l'output per Nagios:

*print qq{$output - VERSION PEOPLE $res{"HighVersion"}.$res{"MainVersion"}. $res{"Patch"}.$res{"Hotfix"} | 'HighVersion'=$res{"HighVersion"}; 'MainVersion'=$res{"MainVersion"}; 'Patch'=$res{"Patch"}; 'Hotfix'=$res{"Hotfix"};\n};*

ed, infine, viene fatta la scrittura all'interno del CMDB:

*`perl /usr/local/nagios/libexec/notify_by_sysaid_application.pl -H $opt_H -m $output -w "$output - VERSION PEOPLE $res{"HighVersion"}. $res{"MainVersion"}.$res{"Patch"}.$res{"Hotfix"}" -r "$opt_r"`.*

Con PEOPLE VERSION sono terminati i service che vanno a popolare il CMDB, quindi si può passare ad analizzare il codice per la creazione dei service che hanno solo lo scopo di monitoraggio.

Consideriamo i service che permettono di ottenere le informazioni attraverso il monitoraggio con JavaMelody.

ACTIVE CONNECTIONS, ACTIVE THREADS, CPU, USED CONNECTIONS e USED MEMORY sono ottenuti attraverso la stessa funzione che prende il nome di *find_parameters*. Tale funzione prende in ingresso l'indirizzo IP, la porta, username, password e il parametro che si intende intende cercare (usedMemory se si vuole cercare la memoria utilizzata, cpu per la cpu utilizzata, activeThreads per vedere quanti thread attivi ci sono, activeConnections per sapere il numero delle connessione attive e usedConnections per sapere il numero di connessioni utilizzate). Tale funzione prevede:

*my $req=HTTP::Request->new(GET => "http://$ip:$port/webmed/monitoring?part=lastValue&graph=usedMemory,cpu,activeThreads,activeConnections,usedConnections");*

Viene effettuata una richiesta HTTP per ottenere tutti i parametri di interesse;

*$req→authorization_basic($user,$password);*

Vengono forniti username e password per consentire l'accesso ai dati;

*$ua=LWP::UserAgent→new;*

*$res=$ua→request($req);*

*if ($res→is_success){*

*estrazione ed elaborazioni dei dati*

*}*

Viene creato un nuovo userAgent e poi viene estratta la risposta dalla richiesta effettuata. Se tutto avviene con successo è possibile accedere ai dati ed elaborarli. Successivamente, l'output viene formattato per consentire a Nagios di analizzarli:

*print qq{$output - PARAMETER $opt_a: $result | $opt_a=$result;$opt_w; $opt_c;\n}*.

I service GInPatientMR e GUtil vengono utilizzati per estrarre la versione dei due micro-moduli del prodotto Galileo: Entrambi sono ottenuti attraverso la realizzazione

della funzione *find_modvers* che prende come parametri di ingresso il nome del database e il nome del modulo di cui si vuole conoscere la versione. La funzione presenta:

*my $name_version=`/usr/local/firstworks/bin/sqlrsh -id $db -command \"select * from (select pack_name,pack_version from noematica.modules_classes_version t order by t.last_update DESC) where pack_name=\'$mod\'and rownum = 1\"| grep $mod`;*

Questo pezzo di codice permette di estrarre la versione del modulo specificato, $mod, effettuando una query sul database passato come parametro $db. In seguito, bisogna effettuare la formattazione dell'output per consentire a Nagios di elaborare i dati:

*print qq{$output - MODULE $token[0] $token[$#token] | $token[0]=$token[$#token];\n}.*

Il service GALILEO VERSION DB consente di estrarre la versione del database Galileo che si trova su una determinata macchina. A tal scopo, è stata realizzata una funzione chiamata *find_db_version* che prende come parametro di ingresso il nome del database. Per poter ottenere la versione, la funzione presenta:

*my $name_version=`/usr/local/firstworks/bin/sqlrsh -id $db -command \"select t.patch_level||'.'||t.patch_level_minor as db_version from med.tk_patches t where t.system_name = 'med';\"`;*

Successivamente, si va a formattare l'output per consentire a Nagios di analizzare il risultato:

*print qq{$output - VERSION DB GALILEO $result | VERSION_DB_GALILEO=$result;\n};*

L'ultimo service rimasto è il BUS che consente di ottenere quelli oggetti scritti in Java che si trovano sul JBoss del prodotto Halia. A tal scopo, è stata creata una funzione chiamata *find_bus* che prende in ingresso il nome del database di Halia. Per ottenere i bus, la funzione presenta:

*my $bus=`/usr/local/firstworks/bin/sqlrsh -id $db -command \"select lis_id,lis_description from LIS_CONF order by 1;\"`;*

ovvero, viene eseguita una particolare query sul database per ottenere i parametri di interesse. Successivamente, viene formattato l'output per consentire a Nagios di elaborare i risultati:

*print qq{$output - @result | BUS$n=$arr[0]-$arr[1];\n}.*

Tutti questi service vengono realizzati attraverso lo script *check.pl* che presenta diverse funzioni che permettono di ottenere dati monitorati desiderati.

Adesso, si passa all'analisi dello script realizzato per effettuare la scrittura presso il CMDB, ovvero, *notify_by_sysaid_application.pl*. Per poter memorizzare i dati monitorati sono state realizzate diverse funzioni.

La prima funzione è quella che prende il nome di *login*. Tale metodo consente all'utente di loggarsi presso SysAid e di accedere alle diverse API fornite per interagire con il CMDB. Tale funzione prende come parametri di ingresso username, password, accountId, soap endpoint e soap uri. Questi ultimi due parametri vengono definiti come costanti in quanto saranno sempre uguali:

*my $endpoint = qq{http://$o_server:$o_port/$o_uri?wsdl};*

*my $uri = q{urn:SysaidApiService};*

I parametri presenti nell'endpoint, *$o_server, $o_port* e *$o_uri*, vengono letti dal file di configurazione */etc/sysaid_integration.cfg*.

Per il funzionamento del metodo *login*, vengono definiti il metodo SOAP:

*my $l_soap_method =SOAP::Data→name(q{tns:login});*

che permette di capire cosa si vuole fare; i parametri di login:

```
my @l_soap_login_param=(
        SOAP::Data->type("")->name( accountId => $l_accountid ),
        SOAP::Data->type("")->name( userName => $l_user ),
        SOAP::Data->type("")->name( password => $l_pass )
);
```

per poter accedere a SysAid. A questo punto viene fatta la richiesta SOAP al Web Service di SysAid:

*my $l_soap_req = SOAP::Lite->on_fault(sub { my($soap, $res) = @_; log_sysaid("CANNOT CONNECT TO SYSAID $res->faultstring \n"), ref $res ? $res->faultstring : $soap->transport->status, "\n"; })->uri($l_soap_uri) ->proxy($l_soap_endpoint, keep_alive =>1, timeout =>30, proxy=> ['http' => $k_soap_proxy]);*

A questo punto si va a serializzare le strutture dati al SOAP package:

*my $l_serializer = $l_soap_req→serializer();*

Si va a registrare un namespace globale con il SOAP Envelope:

*$l_serializer->register_ns( 'http://api.ilient.com/', 'tns' );*

Viene invocato il metodo definito con i parametri di accesso:

*my $l_result = $l_soap_req->call( $l_soap_method => @l_soap_login_param );*

e da questo si ottiene il sessionId che permette di accedere alle API SysAid per interagire con il Web Service ed effettuare le operazione sul CMDB.

Altra funzione importante è quella chiamata *ci_content* che ha il compito di fornire il contenuto di ciascun CI. I parametri di ingresso di questo metodo sono il sessionId fornito dalla funzione *login*, soap endpoint, soap uri e il CIId che identifica il CI di cui si vogliono avere le informazioni. A questo punto bisogna definire il metodo:

*my $l_soap_method =SOAP::Data→name(q{tns:loadByStringId});*

i parametri da fornire al metodo per eseguire l'operazione richiesta:

```
my @l_param=(
        SOAP::Data->type("")->name( sessionId => $id ),
        SOAP::Data->type("tns:apiCI")->name(q{apiSysObj} => q{}),
        SOAP::Data->type("")->name( id => $ci),
);
```

Arrivati a questo punto, viene fatta la richiesta SOAP, vengono serializzate le strutture dati, viene registrato il namespace globale con il SOAP Envelope ed, infine, invocato il metodo con i parametri; tutto esattamente come nella funzione *login*.

Un'altra funzione molto importante è quella che prende il nome di *modify_ci*. Tale metodo consente di creare e/o modificare i valori che costituiscono un CI. I parametri di ingresso sono il sessionId, soap endpoint, soap uri, il CIId ed, infine, un parametro che rappresenta l'output di Nagios. Quest'ultimo è stato inserito per consentire ai service di poter andare a popolare il CMDB.

Anche in questo caso, viene definito il metodo da invocare:

*my $l_soap_method =SOAP::Data→name(q{tns:save});*

e i parametri da utilizzare per l'esecuzione del metodo; trattandosi di una scrittura, questi rappresentano i dati che si vanno a memorizzare nel CMDB:

```
my @l_param=(
        SOAP::Data->type("")->name( sessionId => $id ),
        SOAP::Data->type("tns:apiCI")->name( q{apiSysObj} => (
                \SOAP::Data→value(
                        SOAP::Data->type("xsd:string")->name( ciName => $res->result->{'ciName'}),,
                        ...
                )
        )
)
```

In questo blocco di codice, si può capire che il primo parametro è il sessionId, senza il quale non si può effettuare nessuna interazione con SysAid, il secondo, invece, contiene una molteplicità di parametri che verranno scritti nel CMDB. In questa circostanza, è stato riportato solo il ciName per mostrare la struttura di come un parametro deve essere definito, ma in realtà ce ne sono moltissimi che possono essere visualizzati accedendo al codice sorgente.

Fatto questo, viene fatta la richiesta SOAP di scrittura, vengono serializzate le strutture dati, viene registrato il namespace globale con il SOAP Envelope, viene invocato il metodo con i corrispondenti parametri ed, infine, viene restituito il CIId corrispondente al CI che è stato creato o modificato.

Ultima funzione di questo script è quella che prende il nome di *obtain_ci*. Tale funzione consente di ottenere tutti i CIId di interesse in base alla condizione che gli viene passata. Come parametri di ingresso presenta il sessionId, id della company, hostname, soap endpoint, soap uri e il ci_sub_type. La prima cosa da fare è definire il metodo da eseguire:

```
my $l_soap_method=SOAP::Data→name(q{tns:executeSelectQuery});
```

la condizione che deve essere soddisfatta:

```
my $l_cust_list =qq{ company = $id_company AND ci_sub_type = $ci_sub_type AND ci_cust_text_10 like '%$l_hostname%'};
```

in questo caso, bisogna sottolineare come il parametro ci_cust_text_10 rappresenta in CMDB quello che più volte è stato definito come nagios_id. Successivamente, bisogna specificare i parametri necessari al metodo per eseguire:

```
my @l_soap_req_tk_param =(
        SOAP::Data->type("")->name( sessionId => $sessionid),
        SOAP::Data->type("tns:apiCI")->name(q{apiSysObj} => q{}),
        SOAP::Data->type("")->name( condition => $l_cust_list)
);
```

Successivamente, bisogna fare la richiesta SOAP, serializzare le strutture dati, registrare il namespace globale con il SOAP Envelope ed, infine, invocare il metodo con i parametri specificati ottenendo il risultato, ovvero, la lista di CIId che soddisfano la condizione *condition*.

Questo era l'ultimo metodo fondamentale presente nello script *notify_by_sysaid_application.pl*. Questi metodi, combinati fra di loro, consentono ai service definiti di andare a popolare i campi che caratterizzano un CI in base alle necessità dell'utente.

Tutti gli script definiti in precedenza consentono la realizzazione di service che possono essere configurati in Skynet. Vediamo adesso lo script utilizzato per la configurazione degli item in Zabbix che corrispondono ai service di Skynet.

Una prima cosa importante da dire è che, mentre per Skynet si avevano vincoli richiesti dall'azienda e, di conseguenza, sono stati realizzati diversi script per i service, in questo caso, essendo una semplice integrazione in Zabbix, è stato realizzato un solo script che contiene tutte le funzioni necessarie per ottenere i dati monitorati. Tale script prende il nome di *new_check.pl*.

Sostanzialmente, non ci sono grandissime differenze rispetto alle funzioni presenti negli script utilizzati per Skynet, però, ci sono aspetti importanti che devono essere considerati.

La prima cosa importante è che non è assolutamente richiesta la formattazione dell'output, questo perché mentre in Nagios il plugin/script deve restituire lo stato del service o dell'host, in Zabbix viene restituito solo il risultato, in quanto si possono creare i trigger per ciascun item o per ciascun host che permette di monitorare il cambiamento di stato. Detto questo, si può affermare che, per ogni funzione, non

bisogna fare altro che far restituire il risultato che, nel main, verrà mostrato attraverso una print.

Il secondo aspetto importante è rappresentato dal fatto che per accedere ai database non è stata utilizzata la funzione *sqlrsh*, ma *sqlplus*; pertanto consideriamo le funzioni che ne fanno uso e analizziamo le modifiche.

Per gli item GInPatientMR e GUtil, viene utilizzata la funzione *find_module_version*. Questa, per ottenere la versione dei moduli, effettua una query sul database non più attraverso *sqlrsh*, ma attraverso *sqlplus*, quindi si avrà:

*my $version=`echo "select * from (select pack_name,pack_version from noematica.modules_classes_version t order by t.last_update DESC) where pack_name=\'$module_name\' and rownum = 1;" | /usr/lib/oracle/11.2/client64/bin/sqlplus $user\/$pwd\@$ip:$port\/$sid | tail -4 | head -1`;*

dove i parametri user, pwd, ip, port e sid sono parametri di ingresso alla funzione.

Altro item che fa uso di *sqlplus* è il Database Version, che corrisponde al GALILEO VERSION DB di Skynet. In questo caso, si ha la funzione *find_db_version* che prende come parametri di ingresso indirizzo IP, porta, username, password e SID del database. La versione sarà ottenuta attraverso:

*my $version=`echo "select t.patch_level||'.'||t.patch_level_minor as db_version from med.tk_patches t where t.system_name = 'med';" | /usr/lib/oracle/11.2/client64/bin/sqlplus $user\/$pwd\@$ip:$port\/$sid | tail -3 | head -1`;*

Anche l'item Halia Version accede al database attraverso la funzione *find_halia_version* che prende, come parametri di ingresso, indirizzo IP, porta, username, password e SID del database. La versione sarà ottenuta attraverso:

*my $version=`echo "select * from v_sk_halia_vers;" | /usr/lib/oracle/11.2/client64/bin/sqlplus $user\/$pwd\@$ip:$port\/$sid | tail -3 | head -1`;*

L'item Device Configuration accede al database per ottenere la lista di dispositivi medicali collegati ad Halia. Si basa sulla funzione *find_dev_config* che prende, come parametri di ingresso, indirizzo IP, porta, username, password e SID del database. La lista dei dispositivi sarà ottenuta attraverso:

*my $dev=`echo "select dc.device_description from haldev.devices_conf dc;" | /usr/lib/oracle/11.2/client64/bin/sqlplus $user\/$pwd\@$ip:$port\/$sid | tail -25 | head -21 | sed 's/DEVICE_DESCRIPTION//g' | sed 's/-//g' | tr -s "\n"`;*

Ultimo item è il Bus che consente di ottenere gli oggetti Java che si trovano sul JBoss di Halia. Si basa sulla funzione *find_bus* che prende, come parametri di ingresso, indirizzo IP, porta, username, password e SID del database. La lista dei bus sarà ottenuta attraverso:

*my $bus=`echo "select lis_id,lis_description from LIS_CONF order by 1;" | /usr/lib/oracle/11.2/client64/bin/sqlplus $user\/$pwd\@$ip:$port\/$sid | tail -5 | head -3 | tr -d "\t" | cut -d' ' -f1,6`;*

Altro aspetto importante in Zabbix riguarda quei servizi che hanno il compito di ottenere i dati monitorati di macchine Linux. Mentre nel caso di Skynet, si utilizza una funzione per l'utilizzo di ssh, in questo caso viene usato il metodo *sshpass*. Per l'item Processor Number si utilizza:

*$result=`/usr/bin/sshpass -p \"$opt_P\" ssh -o StrictHostKeyChecking=no root\@$opt_H 'cat /proc/cpuinfo | grep -c processor'`;*

dove il parametro opt_P rappresenta la password, mentre il parametro opt_H rappresenta l'indirizzo IP;

per l'item Processor Model:

*$result=`sshpass -p \"$opt_P\" ssh -o StrictHostKeyChecking=no root\@$opt_H 'cat /proc/cpuinfo | grep \"model name\" | cut -d: -f2 | tr -s \" \" | sed \"s/ //\"'`;*

per l'item Hostname:

*$result=`sshpass -p \"$opt_P\" ssh -o StrictHostKeyChecking=no root\@$opt_H 'hostname'`;*

per l'item Operating System:

*$result=`sshpass -p \"$opt_P\" ssh -o StrictHostKeyChecking=no root\@$opt_H 'lsb_release -a | grep Description | cut -d":" -f2 | tr -d "\t"'`;*

ed, infine, per l'item Ram:

*$result=`sshpass -p \"$opt_P\" ssh -o StrictHostKeyChecking=no root\@$opt_H 'cat /proc/meminfo | grep MemTotal | rev | cut -d" " -f2 | rev'`.*

Ultimo aspetto importante in Zabbix è che, mentre in Skynet per la realizzazione dei servizi per scopi statistici sono stati realizzati attraverso uno script predisposto, in questo caso, invece, è stato necessario creare una funzione che prende il nome di *find_new_parameters*. Tale funzione prende in ingresso indirizzo IP della macchina, porta, username, password, il sid del database ed un parametro che serve per distinguere i diversi dati monitorati. Tale funzione si basa su *sqlplus*, in quanto tali dati sono presenti all'interno del database.

Per l'item Cartelle Cliniche si ha:

*$value=`echo "select * from v_sk_med_rec;" | /usr/lib/oracle/11.2/client64/bin/sqlplus $user\/$pwd\@$ip:$port\/$sid | tail -3 | head -1 | rev | cut -d " " -f1 | rev`;*

per Lettere di Dimissioni:

*$value=`echo "select * from v_sk_res_let;" | /usr/lib/oracle/11.2/client64/bin/sqlplus $user\/$pwd\@$ip:$port\/$sid | tail -3 | head -1 | rev | cut -d " " -f1 | rev`;*

per Referti di Consulenza:

*$value=`echo "select * from v_sk_cons_ref;" | /usr/lib/oracle/11.2/client64/bin/sqlplus $user\/$pwd\@$ip:$port\/$sid | tail -3 | head -1 | rev | cut -d " " -f1 | rev`;*

per Richieste di Anatomia Patologica:

*$value=`echo "select * from v_sk_pat_req;" | /usr/lib/oracle/11.2/client64/bin/sqlplus $user\/$pwd\@$ip:$port\/$sid | tail -3 | head -1 | rev | cut -d " " -f1 | rev`;*

per Richieste di Consulenza:

*$value=`echo "select * from v_sk_adv_req;" | /usr/lib/oracle/11.2/client64/bin/sqlplus $user\/$pwd\@$ip:$port\/$sid | tail -3 | head -1 | rev | cut -d " " -f1 | rev`;*

per Richieste di Laboratorio:

*$value=`echo "select * from v_sk_lab_req;" | /usr/lib/oracle/11.2/client64/bin/sqlplus $user\/$pwd\@$ip:$port\/$sid | tail -3 | head -1 | rev | cut -d " " -f1 | rev`;*

ed, infine, per Richieste di Radiologia:

*$value=`echo "select * from v_sk_radio_req;" | /usr/lib/oracle/11.2/client64/bin/sqlplus $user\/$pwd\@$ip:$port\/$sid | tail -3 | head -1 | rev | cut -d " " -f1 | rev`.*

Con questo è terminata la realizzazione del progetto, in termini di codice, mostrando come l'integrazione con Zabbix può avvenire in modo semplice e veloce senza dover riscrivere righe di codice particolarmente complesse.

## 5.4 – Risultati sperimentali

In questo paragrafo si parlerà dei risultati sperimentali che si sono ottenuti con il server Web e con i service/item che sono stati realizzati e delle loro performance, in termini di tempo.

Per quanto riguarda il server Web, ci sono diversi controlli che possono essere effettuati a discrezione dell'utente. Si parte con il primo controllo:

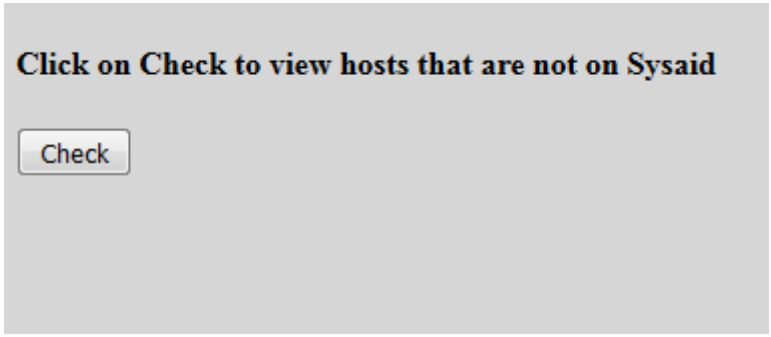



*Figura 57: Sezione relativa agli host attivi*

Cliccando sul tasto Check della Figura 57, verranno mostrati tutti gli host che sono correntemente attivi e monitorati attraverso il sistema di monitoraggio. In questa prima circostanza, si considera Skynet:


*Figura 58: Lista degli host attivi su Skynet*

La Figura 58 mostra come il server Web restituisce la lista degli host correntemente attivi su Skynet. Inoltre, si può notare che solo dopo aver effettuato il Check, compare il tasto che permette di fare la sincronizzazione e, di conseguenza, la scrittura dei dati presso il SysAid CMDB. Altro aspetto importante è rappresentato

dal link “return to check” che permette di tornare alla pagina iniziale del server per effettuare altri controlli.

Il tempo impiegato da server per effettuare questo controllo è il seguente:


*Figura 59: Tempo per ottenere la lista degli host*

Dalla Figura 59, si può constatare come il tempo impiegato dal server per ottenere la lista degli host monitorati con Skynet è relativamente basso, ovvero, circa 3,5 secondi. È chiaro che i tempi non sono fissi e possono variare in base al numero di host presenti, ma anche in base al tempo impiegato da SysAid per fornire la risposta. L’aspetto positivo, però, è che i tempi non variano di molto a discapito di un controllo che deve essere fatto manualmente; infatti, un utente dovrebbe controllare quali host sono monitorati e quali sono già presenti nel CMDB, lavoro non di poco conto.

Analizzando la schermata di Skynet, invece, si può notare come gli host effettivamente monitorati sono quelli restituiti dal server:

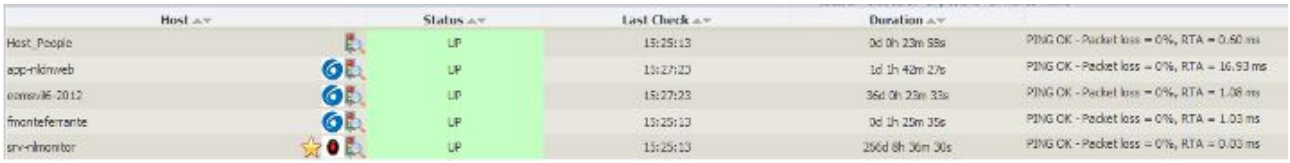



*Figura 60: Host monitorati da Skynet*

Successivamente, si passa alla scrittura dei dati cliccando sul tasto Sync della Figura 58, ottenendo come risultato:

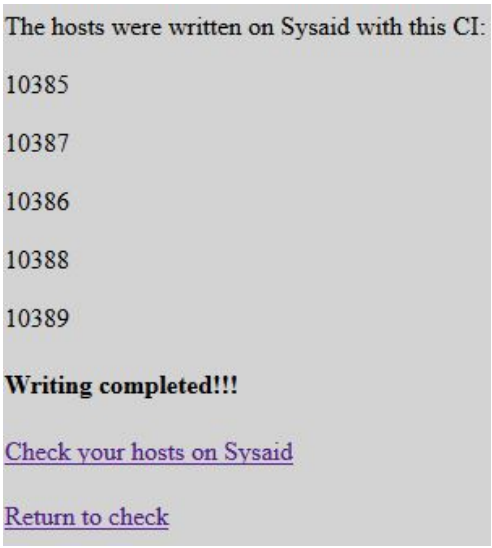



*Figura 61: Lista dei CIId corrispondenti agli host*

La Figura 61 mostra la lista dei CIId che corrispondono agli host che sono stati scritti all'interno del CMDB. Altro aspetto importante è che, oltre al link che rimanda alla pagina iniziale per effettuare altri controlli, ve ne è un altro "Check your hosts on Sysaid" che rimanda alla pagina di SysAid per verificare che a scrittura sia avvenuta con successo. Il tempo impiegato dal server per effettuare la scrittura è il seguente:

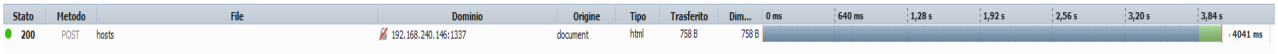



*Figura 62: Tempo di scrittura degli host*

Dalla Figura 62, si può constatare che il tempo impiegato dal server per effettuare la scrittura all'interno del CMDB è di circa 4 secondi.

Volendo tirare le somme, si potrebbe affermare che il tempo impiegato per controllare e memorizzare gli host correntemente monitorati da Skynet è di circa 8 secondi; un ottimo tempo se si va a considerare il fatto che il tutto doveva essere fatto manualmente.

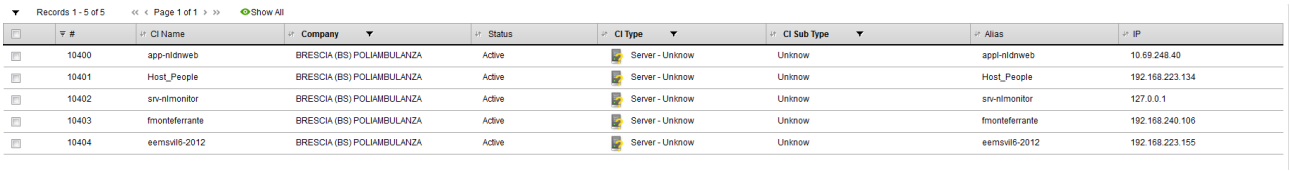



*Figura 63: Lista degli host su SysAid*

La Figura 63 mostra come la scrittura sia avvenuta con successo memorizzando oltre al nome delle macchine anche il suo alias e il suo indirizzo IP; queste rappresentano le informazioni iniziali che vengono memorizzate attraverso il server Web. Una volta terminata la scrittura, se si decide di riprovare a fare tale controllo, verrà restituita una lista vuota in quanto non ci sono altri host che devono essere memorizzati; è chiaro, però, che se si aggiungono altri host su Skynet, allora il server restituirà la lista con gli host mancanti su SysAid.

Adesso si deve prendere in considerazione Zabbix, come sistema di monitoraggio, per dimostrare come l'integrazione sia avvenuta con successo, ricordando che inizialmente il server Web era stato realizzato per Skynet. La lista degli host presenti su Zabbix è la seguente:


*Figura 64: Host monitorati con Zabbix*

Tendenzialmente, gli host monitorati con Zabbix sono gli stessi di quelli monitorati con Skynet con l'aggiunta di qualcun altro. Se si va a cliccare sul Check nella sezione per il controllo degli host, il risultato è il seguente:


*Figura 66: Lista di host monitorati con Zabbix parte1*

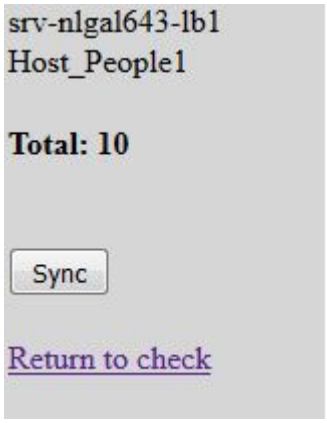



*Figura 65: Lista di host monitorati con Zabbix*

Dalle Figure 65 e 66 è possibile notare come il controllo sia andato a buon fine. Il tempo impiegato dal server per effettuare il controllo è il seguente:


*Figura 67: Tempo per ottenere la lista degli host*

Dalla Figura 67, si può constatare come il tempo impiegato dal server per controllare gli host che sono correntemente attivi e monitorati è di circa 4 secondi. Anche in questo caso il tempo può essere considerato accettabile.

Nel momento in cui si decide di fare la scrittura, si clicca sul tasto Sync e i risultati sono i seguenti:

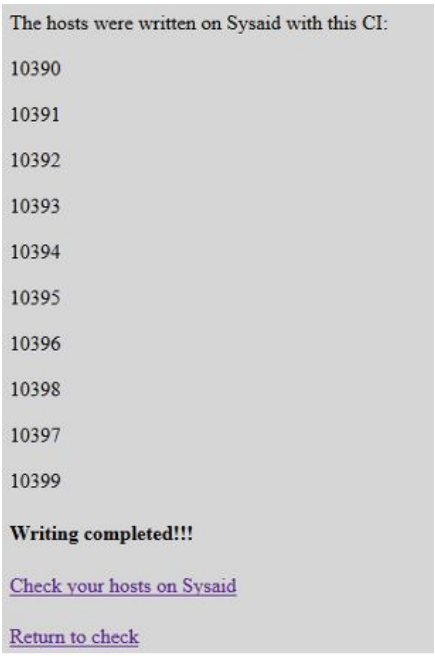



*Figura 68: Lista CIId corrispondenti agli host scritti nel CMDB*

Il tempo di scrittura impiegato dal server è il seguente:


*Figura 69: Tempo impiegato per la scrittura*

Dalla Figura 69, si può notare come il tempo impiegato per effettuare la scrittura presso il CMDB è di circa 4,5 secondi; quindi sommandoli ai 4,5 secondi per effettuare il controllo si può affermare che il tutto viene svolto in circa 9 secondi. Ancora una volta il tempo risulta essere ottimo e accettabile. In seguito alla scrittura, nel CMDB si avrà:

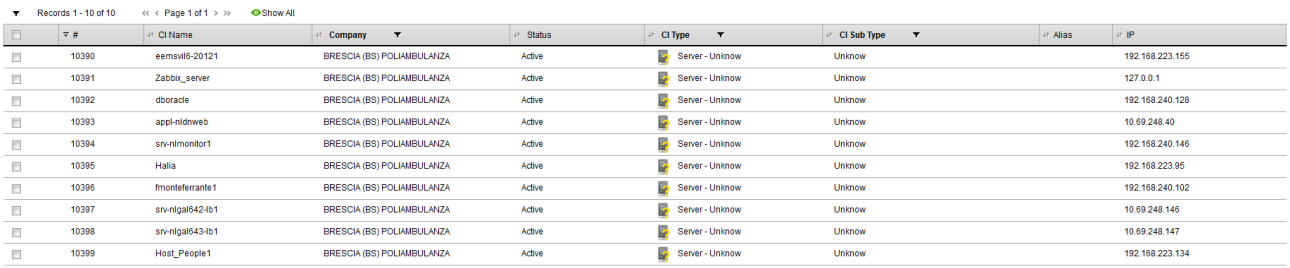



*Figura 70: Host monitorati da Zabbix sul CMDB*

Dalla Figura 70, si può notare come la scrittura sia avvenuta con successo memorizzando oltre al nome delle macchine anche il suo alias e il suo indirizzo IP; queste rappresentano le informazioni iniziali che vengono memorizzate attraverso il server Web.

Si passa al secondo controllo:

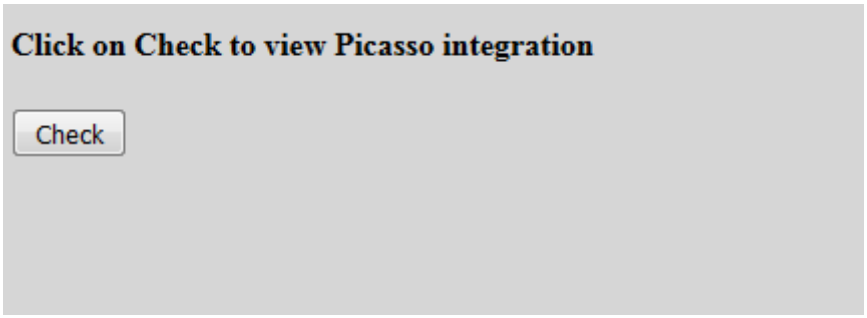



*Figura 71: Sezione relativa agli host contenenti il service HSO JBOSS PICASSO*

Cliccando sul tasto Check della Figura 71, verranno mostrati tutti i service HSO JBOSS PICASSO che sono presenti sugli host attivi e monitorati attraverso il sistema di monitoraggio. In questa prima circostanza, si considera Skynet:

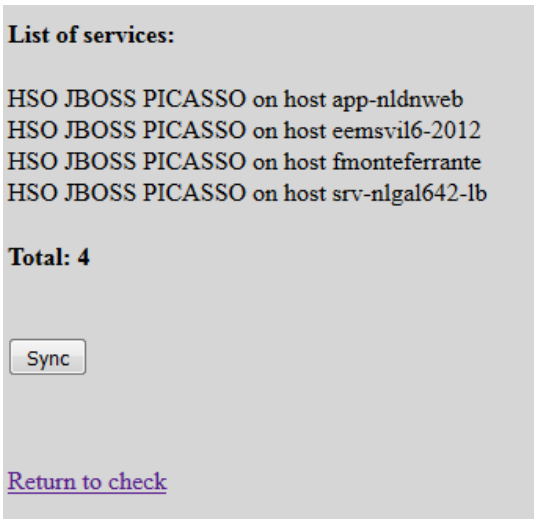



*Figura 72: Risultato del controllo*

Dalla Figura 72, si può notare su quali host il service è presente. Il tempo impiegato dal server per effettuare il controllo è il seguente:


*Figura 73: Tempo impiegato per il controllo*

Dalla Figura 73, si può notare come il tempo impiegato è di circa 2 secondi, assolutamente accettabile. Analizzando la schermata di Skynet, si può notare come il server ha fornito risultati giusti:

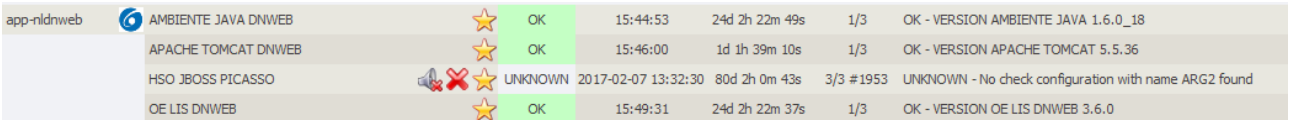
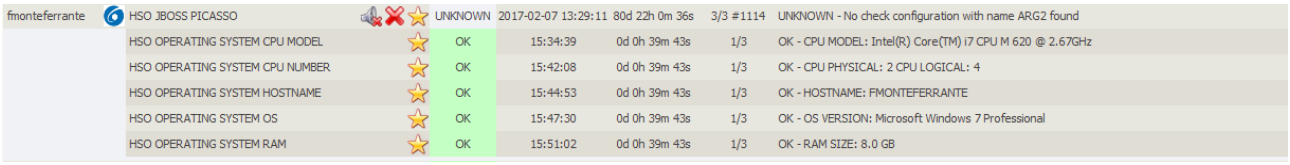



*Figura 74: Presenza del service sull'host app-nldnweb*

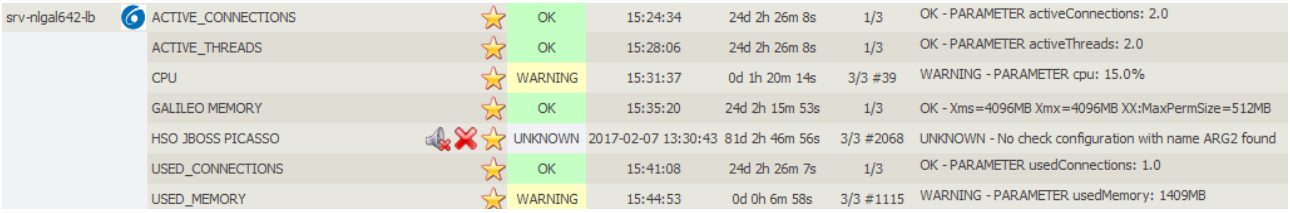
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| eemsvil6-2012 | ACTIVE_CONNECTIONS | OK | 15:23:03 | 24d 2h 26m 8s | 1/3 | OK - PARAMETER activeConnections: 0.0 |
| | ACTIVE_THREADS | OK | 15:26:35 | 24d 2h 26m 7s | 1/3 | OK - PARAMETER activeThreads: 0.0 |
| | BUS | OK | 15:30:07 | 24d 2h 26m 8s | 1/3 | OK - 1-LIS1 2-LIS2 3-SUNQUEST |
| | CARTELLE CLINICHE | OK | 15:33:38 | 24d 2h 27m 42s | 1/3 | OK - value: 0 |
| | CPU | OK | 15:39:26 | 24d 2h 26m 8s | 1/3 | OK - PARAMETER cpu: 0.0% |
| | DEVICES CONFIGURATION HALIA | OK | 15:44:53 | 24d 2h 18m 51s | 1/3 | OK - LIST DEVICES:VISTA#Cobas8000#Modular Lab1#Modular Lab2#XE2100#SiemensVista1-Lab1#SiemensVista2-Lab1#SiemensVista1-Lab2#SiemensVista2-Lab2#Navios 1#Navios 2#Navios 3#Navios 4# |
| | GALILEO MEMORY | OK | 15:46:30 | 24d 2h 17m 52s | 1/3 | OK - Xms=1024MB Xmx=1024MB XX:MaxPermSize=512MB |
| | GInPatientMR | OK | 15:50:02 | 24d 2h 26m 8s | 1/3 | OK - MODULE GInPatientMR 1.5.2-b927 |
| | GUtil | OK | 15:23:33 | 24d 2h 26m 8s | 1/3 | OK - MODULE GUtil 1.6.0-b654 |
| | HALIA VERSION | OK | 15:27:05 | 24d 2h 16m 52s | 1/3 | OK - VERSION HALIA 2.4.0.1 |
| | HSO JBOSS PICASSO | UNKNOWN | 2017-02-07 13:30:43 | 77d 4h 24m 15s | 3/3 #1263 | UNKNOWN - No check configuration with name ARG2 found |
| | LETTERE DI DIMISSIONI | OK | 15:30:37 | 24d 2h 26m 7s | 1/3 | OK - value: 0 |
| | REFERTI DI CONSULENZA | OK | 15:34:09 | 24d 2h 26m 7s | 1/3 | OK - value: 0 |
| | RICHIESTE DI ANATOMIA PATOLOGICA | OK | 15:39:57 | 24d 2h 26m 7s | 1/3 | OK - value: 0 |
| | RICHIESTE DI CONSULENZA | OK | 15:44:53 | 24d 2h 27m 42s | 1/3 | OK - value: 0 |
| | RICHIESTE DI LABORATORIO | OK | 15:47:00 | 24d 2h 26m 7s | 1/3 | OK - value: 0 |
| | RICHIESTE DI RADIOLOGIA | OK | 15:50:32 | 24d 2h 26m 7s | 1/3 | OK - value: 0 |
| | USED_CONNECTIONS | OK | 15:24:04 | 24d 2h 26m 8s | 1/3 | OK - PARAMETER usedConnections: 0.0 |
| | USED_MEMORY | OK | 15:27:35 | 24d 2h 26m 8s | 1/3 | OK - PARAMETER usedMemory: 814MB |
| | VERSION DB GALILEO | OK | 15:31:07 | 24d 2h 26m 8s | 1/3 | OK - VERSION DB GALILEO 51.1 |

*Figura 75: Presenza del service sull'host eemsvil6-2012*



| | | | | | | |
|---|---|---|---|---|---|---|
| fmonteferrante | HSO JBOSS PICASSO | UNKNOWN | 2017-02-07 13:29:11 | 80d 22h 0m 36s | 3/3 #1114 | UNKNOWN - No check configuration with name ARG2 found |
| | HSO OPERATING SYSTEM CPU MODEL | OK | 15:34:39 | 0d 0h 39m 43s | 1/3 | OK - CPU MODEL: Intel(R) Core(TM) i7 CPU M 620 @ 2.67GHz |
| | HSO OPERATING SYSTEM CPU NUMBER | OK | 15:42:08 | 0d 0h 39m 43s | 1/3 | OK - CPU PHYSICAL: 2 CPU LOGICAL: 4 |
| | HSO OPERATING SYSTEM HOSTNAME | OK | 15:44:53 | 0d 0h 39m 43s | 1/3 | OK - HOSTNAME: FMONTEFERRANTE |
| | HSO OPERATING SYSTEM OS | OK | 15:47:30 | 0d 0h 39m 43s | 1/3 | OK - OS VERSION: Microsoft Windows 7 Professional |
| | HSO OPERATING SYSTEM RAM | OK | 15:51:02 | 0d 0h 39m 43s | 1/3 | OK - RAM SIZE: 8.0 GB |

*Figura 76: Presenza del service sull'host fmonteferrante*



| | | | | | | |
|---|---|---|---|---|---|---|
| srv-nlgal642-lb | ACTIVE_CONNECTIONS | OK | 15:24:34 | 24d 2h 26m 8s | 1/3 | OK - PARAMETER activeConnections: 2.0 |
| | ACTIVE_THREADS | OK | 15:28:06 | 24d 2h 26m 8s | 1/3 | OK - PARAMETER activeThreads: 2.0 |
| | CPU | WARNING | 15:31:37 | 0d 1h 20m 14s | 3/3 #39 | WARNING - PARAMETER cpu: 15.0% |
| | GALILEO MEMORY | OK | 15:35:20 | 24d 2h 15m 53s | 1/3 | OK - Xms=4096MB Xmx=4096MB XX:MaxPermSize=512MB |
| | HSO JBOSS PICASSO | UNKNOWN | 2017-02-07 13:30:43 | 81d 2h 46m 56s | 3/3 #2068 | UNKNOWN - No check configuration with name ARG2 found |
| | USED_CONNECTIONS | OK | 15:41:08 | 24d 2h 26m 7s | 1/3 | OK - PARAMETER usedConnections: 1.0 |
| | USED_MEMORY | WARNING | 15:44:53 | 0d 0h 6m 58s | 3/3 #1115 | WARNING - PARAMETER usedMemory: 1409MB |

*Figura 77: Presenza del service sull'host srv-nlgal642-lb*

Come si può notare dalle figure sopra elencate il server ha fornito i risultati giusti. Nel momento in cui si vuole effettuare la scrittura non bisogna fare altro che avviare tale processo andando a cliccare sul tasto Sync.

I risultati sono i seguenti:

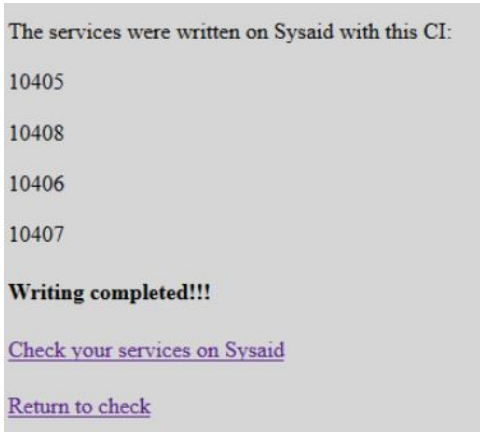



*Figura 78: Lista CIId corrispondenti ai service scritti nel CMDB*

Nella Figura 78, si può notare la lista dei CIId corrispondenti ai CI che sono stati scritti all'interno del CMDB.

Andando a controllore all'interno del CMDB, si avrà:

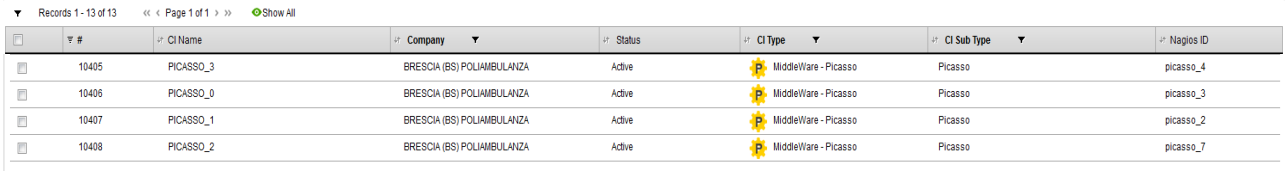



*Figura 79: Scrittura dei service*

Dalla Figura 79, si può notare come la scrittura sia avvenuta con successo. Inoltre, si può osservare un nuovo campo che prende il nome di Nagios_id che contiene una label, picasso, seguito dall'identificativo su cui tale service esegue.

Il tempo impiegato dal server per effettuare la scrittura è il seguente:

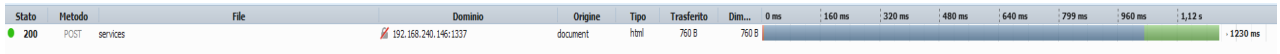



*Figura 80: Tempo di scrittura*

Come si può constatare il tempo di scrittura impiegato dal server Web è di circa 1 secondo. I tempi possono variare in seguito a diversi fattori, come la connessione che si ha a disposizione oppure i tempi di risposta da parte di SysAid, ciononostante risultano essere soddisfacenti.

Anche in questo caso, terminata la scrittura, se viene effettuato un nuovo controllo viene restituita una lista vuota, ma nel momento in cui il service HSO JBOSS PICASSO viene fatto girare su un altro host, allora il server mostrerà la lista con i loro nomi.

Per quanto riguarda Zabbix, invece, gli item HSO JBOSS PICASSO eseguono sulle seguenti macchine:

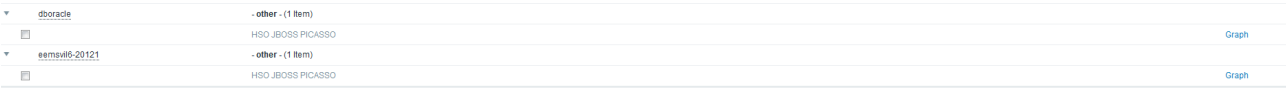



*Figura 81: Host Zabbix su cui esegue l'item HSO JBOSS PICASSO*

Nel momento in cui viene effettuato il controllo, il server restituirà la seguente lista:


*Figura 82: Controllo Zabbix*

Si può notare dalle due figure precedenti come il controllo del server sia andato a buon fine. Il tempo impiegato per effettuare questo controllo è il seguente:


*Figura 83: Tempo per il controllo*

In questa circostanza, il tempo impiegato per verificare su quali macchine girasse un determinato servizio è pari a circa 2 secondi, ancora una volta accettabile.

Per poter effettuare la scrittura bisogna cliccare sul tasto Sync e il risultato è il seguente:

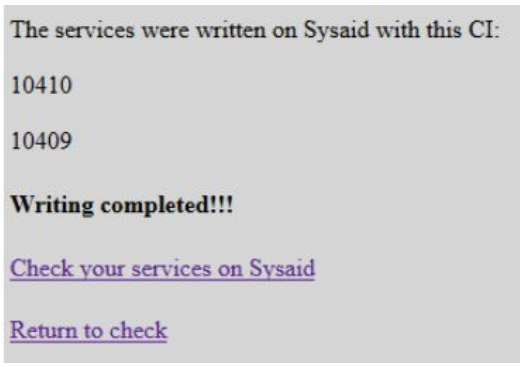



*Figura 84: Lista dei CIId*

Per verificare se la scrittura è avvenuta con successo, si verifica la lista dei CI contenuti all'interno del CMDB, attraverso l'interfaccia Web di SysAid:

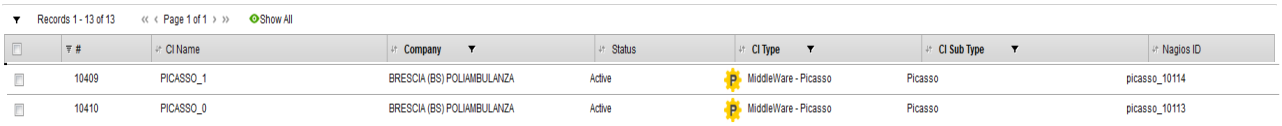



*Figura 85: Lista su SysAid*

Come si può constatare non ci sono stati problemi relativamente alla scrittura; inoltre, il tempo impiegato per farla è la seguente:

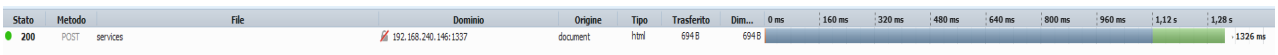



*Figura 86: Tempo di scrittura*

Dalla Figura 86, si può notare come la scrittura sia avvenuta in circa 1,5 secondo, ancora una volta tempo ottimo ed accettabile; se lo sommassimo con il tempo impiegato per il controllo il tempo totale impiegato dal server per svolgere completamente il lavoro sarebbe di circa 3 secondi.

Il prossimo controllo consente di individuare i middleware attraverso il database Galileo_Oracle pertanto non sarà necessario alcun tipo di distinzione fra Skynet e Zabbix:

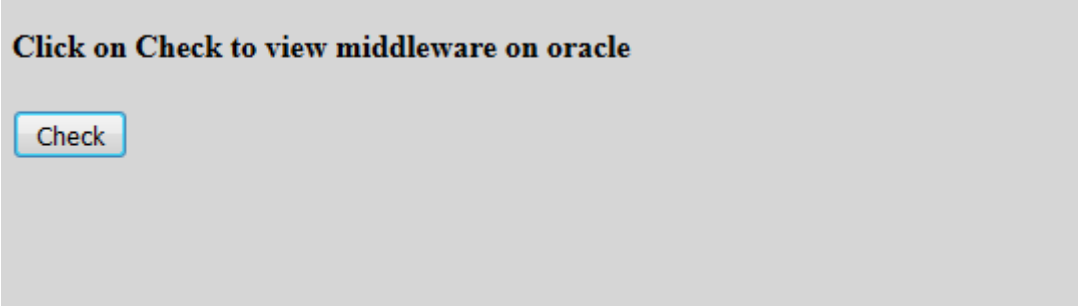



*Figura 87: Controllo middlware*

Cliccando sul tasto Check della Figura 87, verranno mostrati tutti i collegamenti fra service sender e service receiver. Il risultato è il seguente:

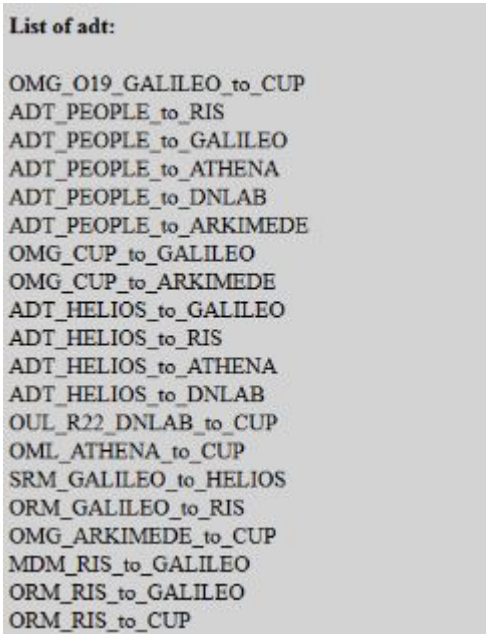



*Figura 88: Lista middleware parte1*

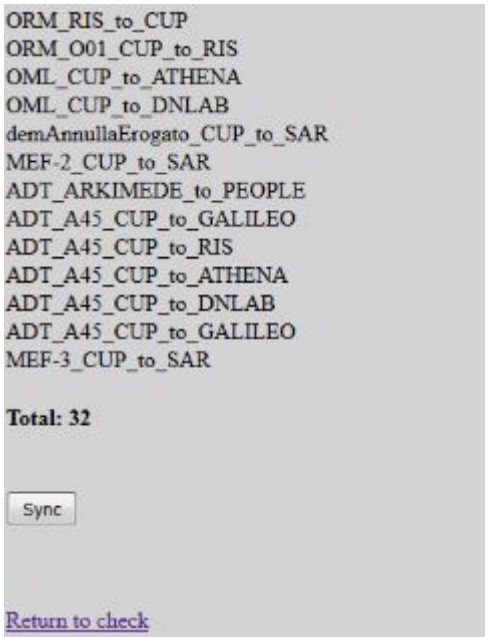



*Figura 89: Lista middleware parte2*

Le due figure mostrano tutti i risultati ottenuti che dovranno, in seguito, essere scritti sul CMDB. Il tempo impiegato dal server per effettuare tale controllo è il seguente:


*Figura 90: Tempo per il controllo*

Dalla Figura 90, si può constatare con il tempo impiegato per effettuare questo controllo è di circa 4 secondi, sempre accettabile per velocizzare e automatizzare il processo di controllo e scrittura all'interno del CMDB.

Per effettuare la scrittura basta cliccare sul tasto Sync. In questa circostanza, non viene riportato l'output in quanto risulterebbe molto lunga, ma si può notare come è avvenuta la scrittura all'interno del CMDB, attraverso l'interfaccia SysAid:

| # | CI Name | Company | Status | CI Type | CI Sub Type | Nagios ID | Notes |
|---|---|---|---|---|---|---|---|
| 10411 | ADT PEOPLE to ATHENA | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | ADT_PEOPLE_to_ATHENA | 21 |
| 10412 | ADT PEOPLE to DNLAB | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | ADT_PEOPLE_to_DNLAB | 21 |
| 10413 | ADT PEOPLE to RIS | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | ADT_PEOPLE_to_RIS | 21 |
| 10414 | ADT PEOPLE to GALILEO | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | ADT_PEOPLE_to_GALILEO | 21 |
| 10415 | OMG O19 GALILEO to CUP | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | OMG_O19_GALILEO_to_CUP | 29 |
| 10416 | OMG CUP to ARKIMEDE | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | OMG_CUP_to_ARKIMEDE | 32 |
| 10417 | OMG CUP to GALILEO | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | OMG_CUP_to_GALILEO | 32 |
| 10418 | ORM GALILEO to RIS | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | ORM_GALILEO_to_RIS | 68 |
| 10419 | OUL R22 DNLAB to CUP | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | OUL_R22_DNLAB_to_CUP | 43 |
| 10420 | ADT PEOPLE to ARKIMEDE | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | ADT_PEOPLE_to_ARKIMEDE | 21 |
| 10421 | SRM GALILEO to HELIOS | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | SRM_GALILEO_to_HELIOS | 54 |
| 10422 | OML ATHENA to CUP | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | OML_ATHENA_to_CUP | 71 |
| 10423 | OMG ARKIMEDE to CUP | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | OMG_ARKIMEDE_to_CUP | 94 |
| 10424 | MDM RIS to GALILEO | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | MDM_RIS_to_GALILEO | 77 |
| 10425 | ORM RIS to CUP | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | ORM_RIS_to_CUP | 83 |
| 10426 | ORM RIS to GALILEO | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | ORM_RIS_to_GALILEO | 83 |
| 10427 | ADT HELIOS to ATHENA | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | ADT_HELIOS_to_ATHENA | 50 |
| 10428 | ADT HELIOS to DNLAB | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | ADT_HELIOS_to_DNLAB | 50 |
| 10429 | ADT HELIOS to RIS | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | ADT_HELIOS_to_RIS | 50 |
| 10430 | ADT HELIOS to GALILEO | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | ADT_HELIOS_to_GALILEO | 50 |
| 10431 | OML CUP to ATHENA | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | OML_CUP_to_ATHENA | 122 |
| 10432 | ADT ARKIMEDE to PEOPLE | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | ADT_ARKIMEDE_to_PEOPLE | 90 |
| 10433 | ORM O01 CUP to RIS | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | ORM_O01_CUP_to_RIS | 128 |
| 10434 | MEF-2 CUP to SAR | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | MEF-2_CUP_to_SAR | 141 |
| 10435 | demAnnullaErogato CUP to SAR | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | demAnnullaErogato_CUP_to_SAR | 147 |
| 10436 | OML CUP to DNLAB | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | OML_CUP_to_DNLAB | 125 |
| 10437 | MEF-3 CUP to SAR | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | MEF-3_CUP_to_SAR | 144 |
| 10438 | ADT A45 CUP to GALILEO | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | ADT_A45_CUP_to_GALILEO | 201 |
| 10439 | ADT A45 CUP to ATHENA | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | ADT_A45_CUP_to_ATHENA | 241 |
| 10440 | ADT A45 CUP to RIS | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | ADT_A45_CUP_to_RIS | 241 |
| 10441 | ADT A45 CUP to DNLAB | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | ADT_A45_CUP_to_DNLAB | 241 |
| 10442 | ADT A45 CUP to GALILEO | BRESCIA (BS) POLIAMBULANZA | Active | MiddleWare - Picasso | Picasso | ADT_A45_CUP_to_GALILEO | 241 |

*Figura 91: Scrittura middleware*

Dalla Figura 91, si può osservare come la scrittura sia avvenuta con successo. Il tempo impiegato per questa operazione è il seguente:

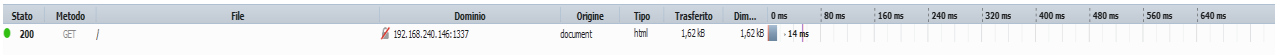



*Figura 92: Tempo di scrittura*

In questa circostanza, anche se può sembrare assurda visto la mole di servizi da scrivere, il tempo impiegato è di circa 14 millisecondi. Questo permette di capire come i tempi variano sempre e ancora una volta sono accettabilissimi.

Terminata la scrittura, se si effettua nuovamente un controllo del genere, viene restituita la lista vuota; nel momento in cui vengono aggiunte nuove informazioni all'interno del database, allora il server sarà in grado di restituire una nuova lista con nuovi risultati.

Si passa al quarto controllo, che in realtà ne comprende tre, infatti:


*Figura 93: Sezione relativa agli che appartengono ad una determinata categoria*

Come si può notare dalla Figura 93, in questa sezione si può scegliere quale controllo effettuare in base alle proprie esigenze. In questo caso, sono stati riuniti in quanto il loro compito è lo stesso, ma cambia solo la categoria a cui si riferiscono. Si considera come sistema di monitoraggio Skynet e si analizzano i controlli in modo parallelo.

Se viene cliccato il tasto Check Galileo il risultato è il seguente:

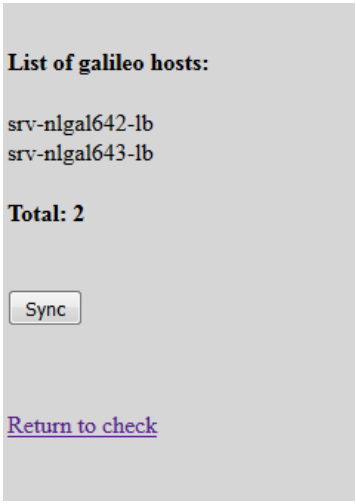



*Figura 94: Lista degli host con categoria Galileo*

Questa rappresenta la lista degli host attivi che appartengono alla categoria Galileo. Il tempo impiegato per questo controllo è il seguente:


*Figura 95: Tempo di controllo*

In questo caso, il tempo impiegato per effettuare il controllo è di circa 800 millisecondi, che risulta essere assolutamente accettabile.

Se viene cliccato il tasto Check Halia il risultato è il seguente:

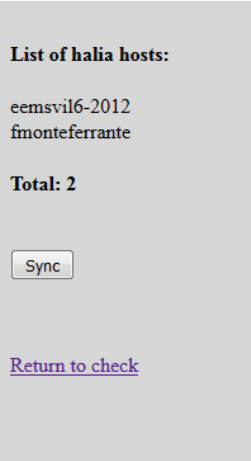



*Figura 96: Lista degli host con categoria Halia*

Questa rappresenta la lista degli host attivi che appartengono alla categoria Halia. Il tempo impiegato per questo controllo è il seguente:

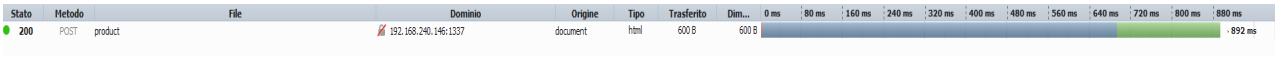



*Figura 97: Tempo di controllo*

Anche in questo caso, il tempo impiegato per effettuare il controllo è di circa 800 millisecondi, che risulta essere assolutamente accettabile.

Se viene cliccato il tasto Check Dnweb il risultato è il seguente:

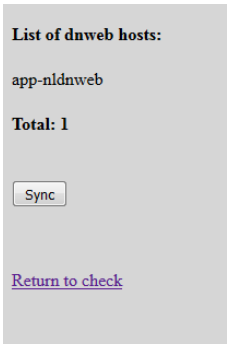
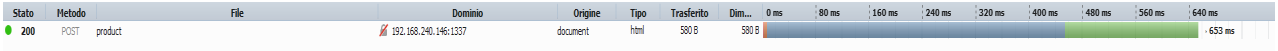



*Figura 98: Lista di host con categoria Dnweb*

Questa rappresenta la lista degli host attivi che appartengono alla categoria Dnweb. Il tempo impiegato per questo controllo è il seguente:




In questo caso, il tempo impiegato per effettuare il controllo è di circa 600 millisecondi, che risulta essere assolutamente accettabile.

Una volta effettuati i controlli, si passa alla scrittura dei dati all'interno del CMDB cliccando il corrispondente tasto Sync.

Per quanto riguarda la categoria Galileo:

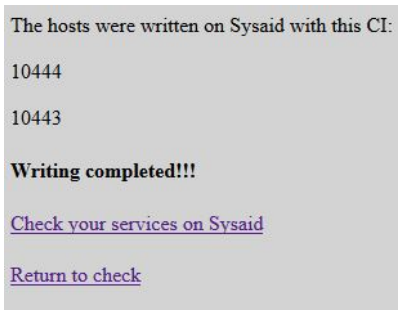



*Figura 99: Scrittura per categoria Galileo*

Viene restituita la lista dei CIId corrispondenti ai CI che sono stati creati e memorizzati all'interno del CMDB. Il tempo di scrittura è il seguente:

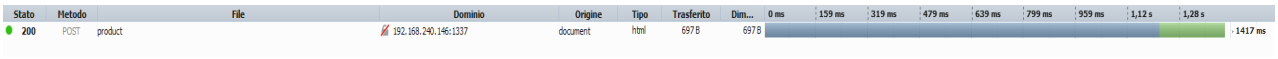



*Figura 100: Tempo di scrittura*

Si può notare dalla Figura 100 come il tempo risulta essere circa 1,5 secondi, che ancora una volta è accettabile. Andando a controllare l'interfaccia di SysAid si avrà:

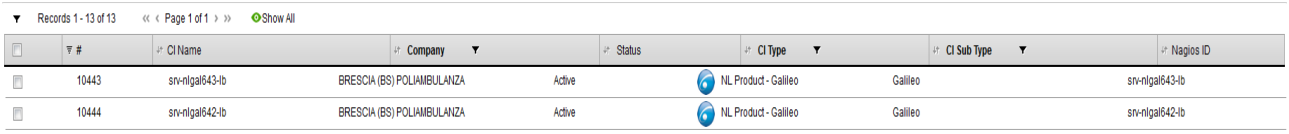



*Figura 101: Interfaccia web SysAid*

In questo caso, si può notare come la scrittura è avvenuta con successo.

Per quanto riguarda la categoria Halia:

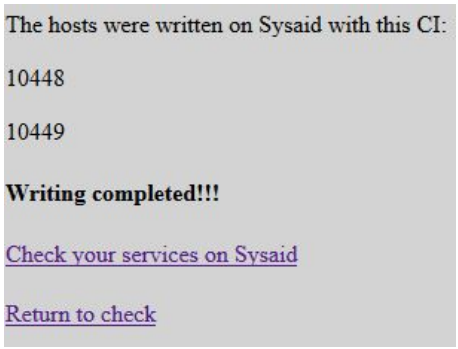



*Figura 102: Scrittura per categoria Halia*

Viene restituita la lista dei CIId corrispondenti ai CI che sono stati creati e memorizzati all'interno del CMDB. Il tempo di scrittura è il seguente:

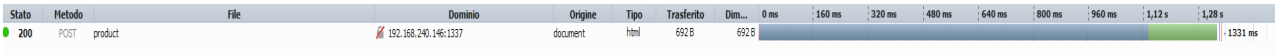



*Figura 103: Tempo di scrittura*

Si può notare dalla Figura 103 come il tempo risulta essere circa 1,5 secondi, che ancora una volta è accettabile. Andando a controllare l'interfaccia di SysAid si avrà:

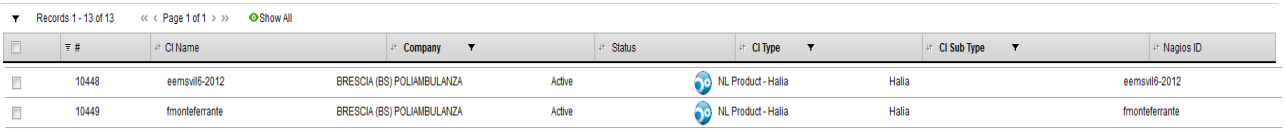



*Figura 104: Interfaccia web SysAid*

Anche in questo caso, si può notare come la scrittura è avvenuta con successo.

Per quanto riguarda l'ultima categoria, ovvero, Dnweb:

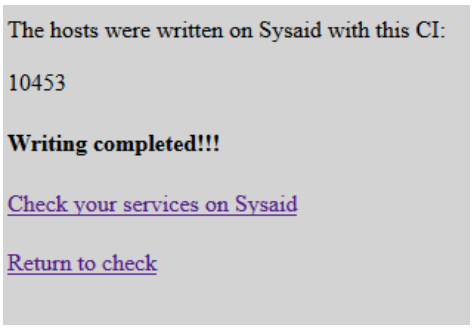



*Figura 105: Scrittura per categoria Dnweb*

Viene restituita la lista dei CIId corrispondenti ai CI che sono stati creati e memorizzati all'interno del CMDB. Il tempo di scrittura è il seguente:

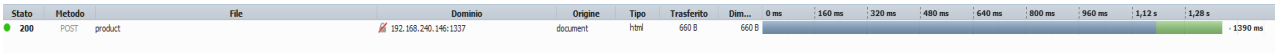



*Figura 106: Tempo scrittura*

Si può notare dalla Figura 106 come il tempo risulta essere circa 1,5 secondi, che ancora una volta è accettabile. Andando a controllare l'interfaccia di SysAid si avrà:

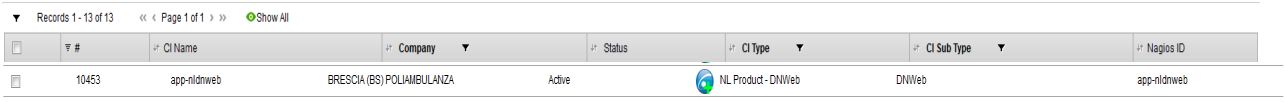



*Figura 107: Interfaccia web SysAid*

Anche in questo caso, si può notare come la scrittura è avvenuta con successo.

Nel momento che si decide di effettuare nuovamente i controlli dopo la scrittura viene restituita la lista vuota.

Questa situazione si verifica perché non si sono più host da memorizzare in quanto già presenti. Nel momento in cui si va a creare un nuovo host con una certa categoria, allora il server fornirà una lista con le nuove macchine.

Dopo aver affrontato il discorso di questi controlli in ambito Skynet, bisogna passare al mondo Zabbix.

Se viene cliccato il tasto Check Galileo il risultato è il seguente:

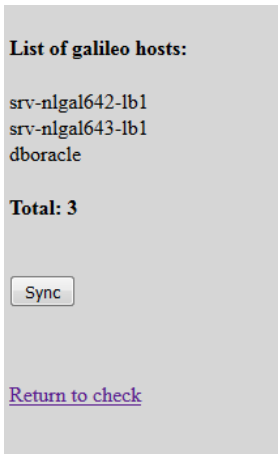



*Figura 108: Lista di host Galileo*

Questa rappresenta la lista degli host attivi che appartengono alla categoria Galileo. Il tempo impiegato per questo controllo è il seguente:

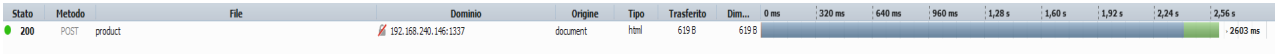



*Figura 109: Tempo di controllo*

In questo caso, il tempo impiegato per effettuare il controllo è di circa 2,5 secondi, che risulta essere accettabile.

Se viene cliccato il tasto Check Halia il risultato è il seguente:


*Figura 110: Lista di host Halia*

Questa rappresenta la lista degli host attivi che appartengono alla categoria Halia. Il tempo impiegato per questo controllo è il seguente:

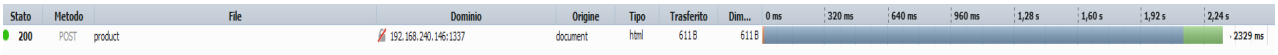



*Figura 111: Tempo di controllo*

Anche in questo caso, il tempo impiegato per effettuare il controllo è di circa 2,5 secondi, che risulta essere accettabile.

Se viene cliccato il tasto Check Dnweb il risultato è il seguente:


*Figura 112: Lista di host Dnweb*

Questa rappresenta la lista degli host attivi che appartengono alla categoria Dnweb. Il tempo impiegato per questo controllo è il seguente:


*Figura 113: Tempo di controllo*

Anche in questo caso, il tempo impiegato per effettuare il controllo è di circa 2,5 secondi, che risulta essere accettabile.

Effettuati i controlli, è possibile effettuare la scrittura attraverso il tasto Sync. Per quanto riguarda la categoria Galileo:

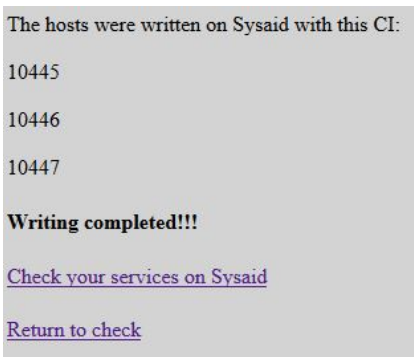



*Figura 114: Lista CIId Galileo*

Viene restituita la lista dei CIId corrispondenti ai CI che sono stati creati e memorizzati all'interno del CMDB. Il tempo di scrittura è il seguente:


*Figura 115: Tempo di scrittura*

Si può notare dalla Figura 115 come il tempo risulta essere circa 1,5 secondi, che ancora una volta è accettabile. Andando a controllare l'interfaccia di SysAid si avrà:

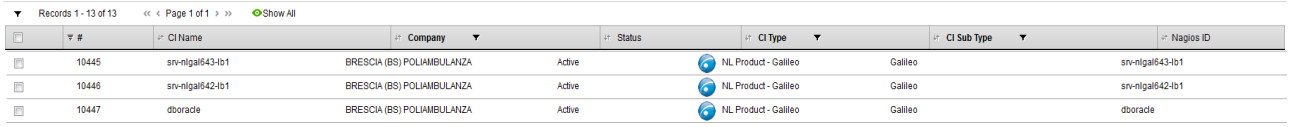



*Figura 116: Interfaccia web SysAid*

In questo caso, si può notare come la scrittura è avvenuta con successo.

Per quanto riguarda la categoria Halia:

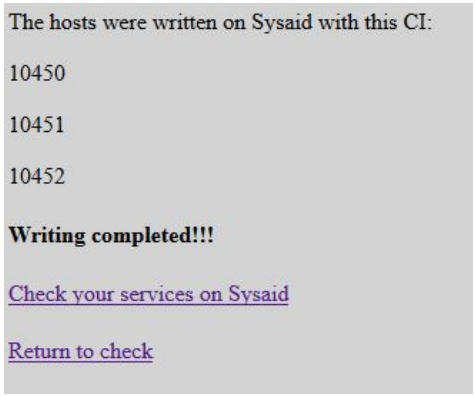



*Figura 117: Lista di CIId Halia*

Viene restituita la lista dei CIId corrispondenti ai CI che sono stati creati e memorizzati all'interno del CMDB. Il tempo di scrittura è il seguente:

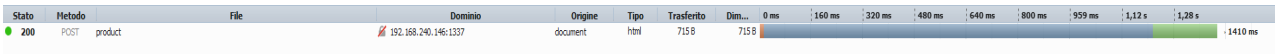



*Figura 118: Tempo di scrittura*

Si può notare dalla Figura 118 come il tempo risulta essere circa 1,5 secondi, che ancora una volta è accettabile. Andando a controllare l'interfaccia di SysAid si avrà:

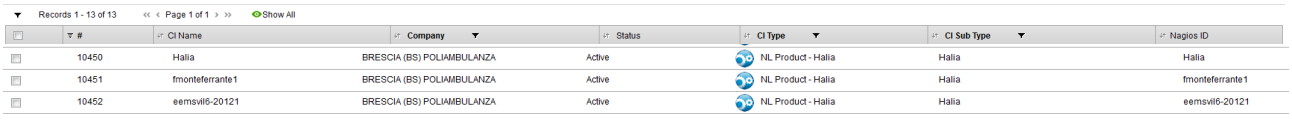



*Figura 119: Interfaccia web SysAid*

Anche in questo caso, si può notare come la scrittura è avvenuta con successo.

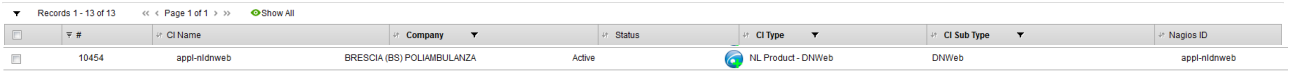
Per quanto riguarda la categoria Dnweb:


*Figura 120: List CIId Dnweb*

Viene restituita la lista dei CIId corrispondenti ai CI che sono stati creati e memorizzati all'interno del CMDB. Il tempo di scrittura è il seguente:


*Figura 121: Tempo di scrittura*

Si può notare dalla Figura 121 come il tempo risulta essere circa 1,5 secondi, che ancora una volta è accettabile. Andando a controllare l'interfaccia di SysAid si avrà:




Anche in questo caso, si può notare come la scrittura è avvenuta con successo.

Per quanto riguarda il server Web, è rimasto l'ultimo controllo, ovvero, verificare quali sono i database con cui la macchina, su cui il server esegue, può interagire. In questa circostanza, quindi, non c'è assolutamente bisogna di fare distinzione fra Skynet e Zabbix, in quanto non hanno nulla a che fare con questo controllo.

Il punto di partenza è rappresentato dalla schermata seguente:

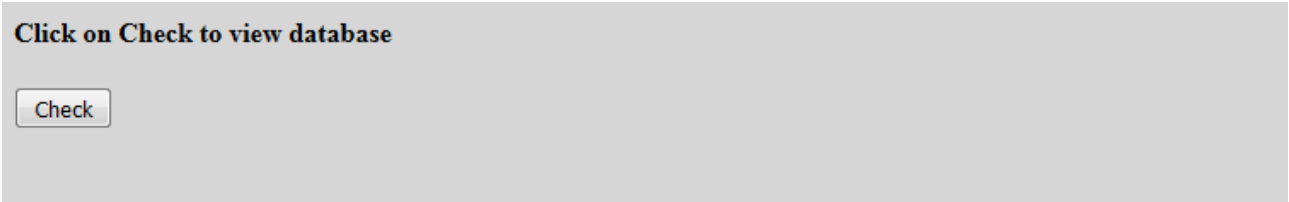



*Figura 122: Sezione relativa al controllo di database*

Cliccando sul tasto Check della Figura 122, verrà restituita la lista di tutti i database con cui la macchina può collegarsi:

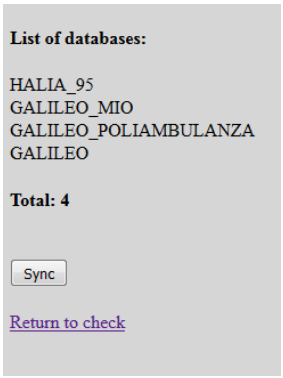



*Figura 123: Lista di database*

Si può constatare come è stata restituita la lista dei nomi di tutti i database con cui il server può interagire. Il tempo impiegato per fare questo controllo è il seguente:


*Figura 124: Tempo di controllo*

Dalla Figura 124, si può notare come il tempo impiegato è di circa 1,5 secondi che risulta essere accettabile. Una volta effettuato il controllo, si può passare alla fase di scrittura. Questa avviene cliccando sul tasto Sync:

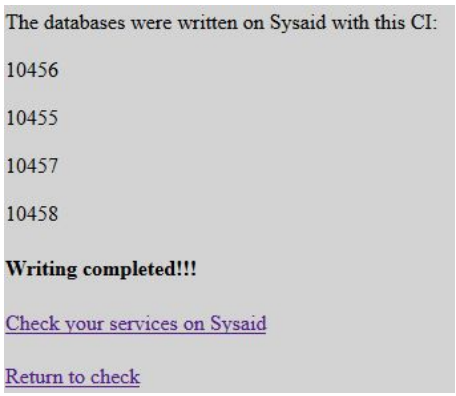



*Figura 125: Lista CIId corrispondenti ai database memorizzati*

In questo caso, viene restituita la lista di CIId corrispondenti ai CI creati e memorizzati all'interno del CMDB. Il tempo impiegato per fare la scrittura è il seguente:

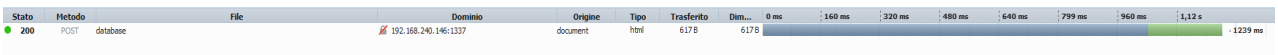



*Figura 126: Tempo di scrittura*

Dalla Figura 126, s può notare che il tempo impiegato è di circa 1,5 secondi che risulta essere assolutamente accettabile. Per avere la conferma della scrittura, si verifica l'interfaccia Web di SysAid:

| # | CI Name | Company | Status | CI Type | CI Sub Type | Nagios ID | SID | DB Version | TCP/IP Port |
|---|---|---|---|---|---|---|---|---|---|
| 10455 | HALIA_95 | BRESCIA (BS) POLIAMBULANZA | Active | Data Base - Oracle | Oracle | HALIA_95 | hal | 11.1.0.7.0 | 1521 |
| 10456 | GALILEO_MIO | BRESCIA (BS) POLIAMBULANZA | Active | Data Base - Oracle | Oracle | GALILEO_MIO | XE | 11.2.0.2.0 | 1521 |
| 10457 | GALILEO_POLIAMBULANZA | BRESCIA (BS) POLIAMBULANZA | Active | Data Base - Oracle | Oracle | GALILEO_POLIAMBULANZA | WBMD | 10.2.0.5.0 | 1521 |
| 10458 | GALILEO | BRESCIA (BS) POLIAMBULANZA | Active | Data Base - Oracle | Oracle | GALILEO | wbmd | 11.2.0.4.0 | 1521 |

*Figura 127: Interfaccia web SysAid*

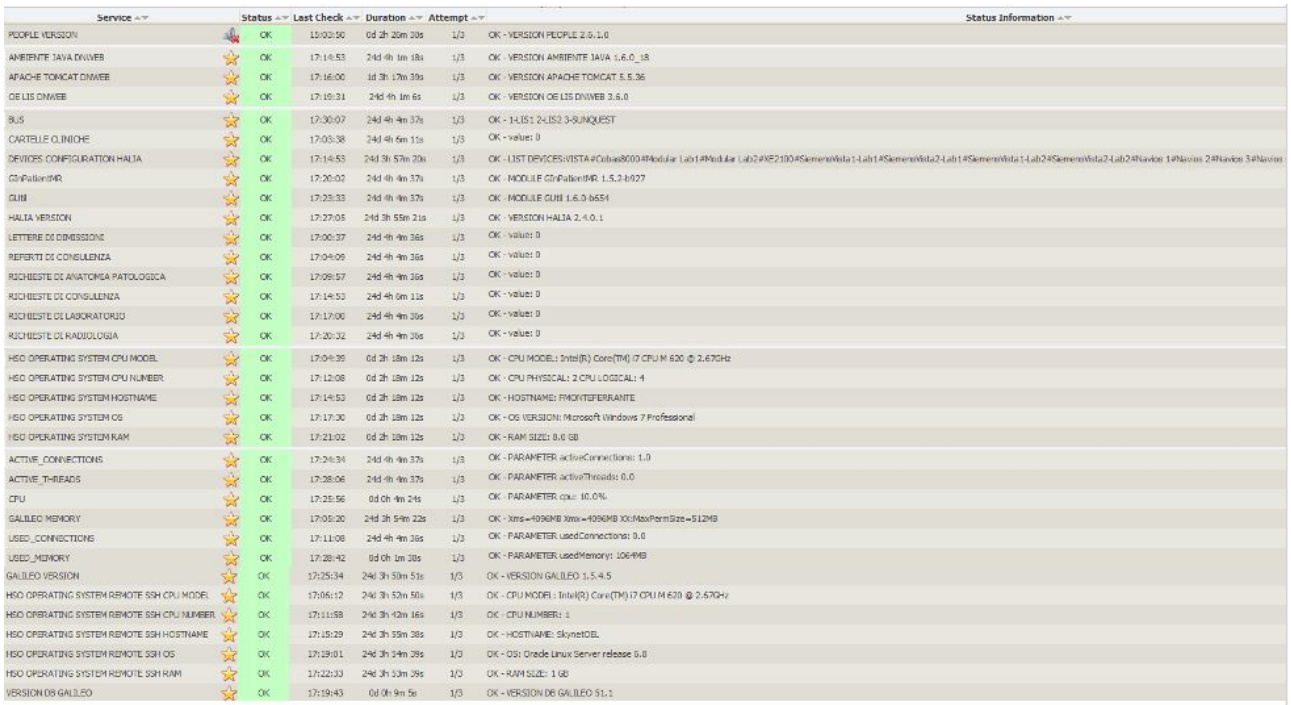
Si può osservare come la scrittura sia avvenuta con successo.

Nel momento in cui si decide di effettuare nuovamente il controllo, non essendoci altri database oltre a quelli già memorizzati, viene restituita una lista vuota. Nel momento in cui viene aggiunto un ulteriore database all'interno del corrispondente file di configurazione, tnsnames.ora, allora il server mostrerà quei database che risultano essere "nuovi".

Questi sono tutti i risultati che si possono ottenere con il server Web realizzato; è chiaro che può essere esteso per consentire altri controlli che sono di interesse e scrivere, all'interno del CMDB, tutti i prodotti che si vogliono monitorare. I tempi, in linea di massima, risultano essere eccellenti; infatti, si varia dai millisecondi a circa 5 secondi per controllo. Se si vuole effettuare l'intera procedura, cioè controllo e scrittura, il tempo massimo potrebbe essere di circa 10 secondi. Questo risulta essere un ottimo tempo considerando che tutto il lavoro deve essere fatto manualmente da un operatore del customer service.

Adesso si può affrontare il discorso dei service/item realizzati in Skynet e Zabbix. Tutti questi risultano essere perfettamente funzionanti fornendo le informazioni richieste.

Per quanto riguarda Skynet si ha:



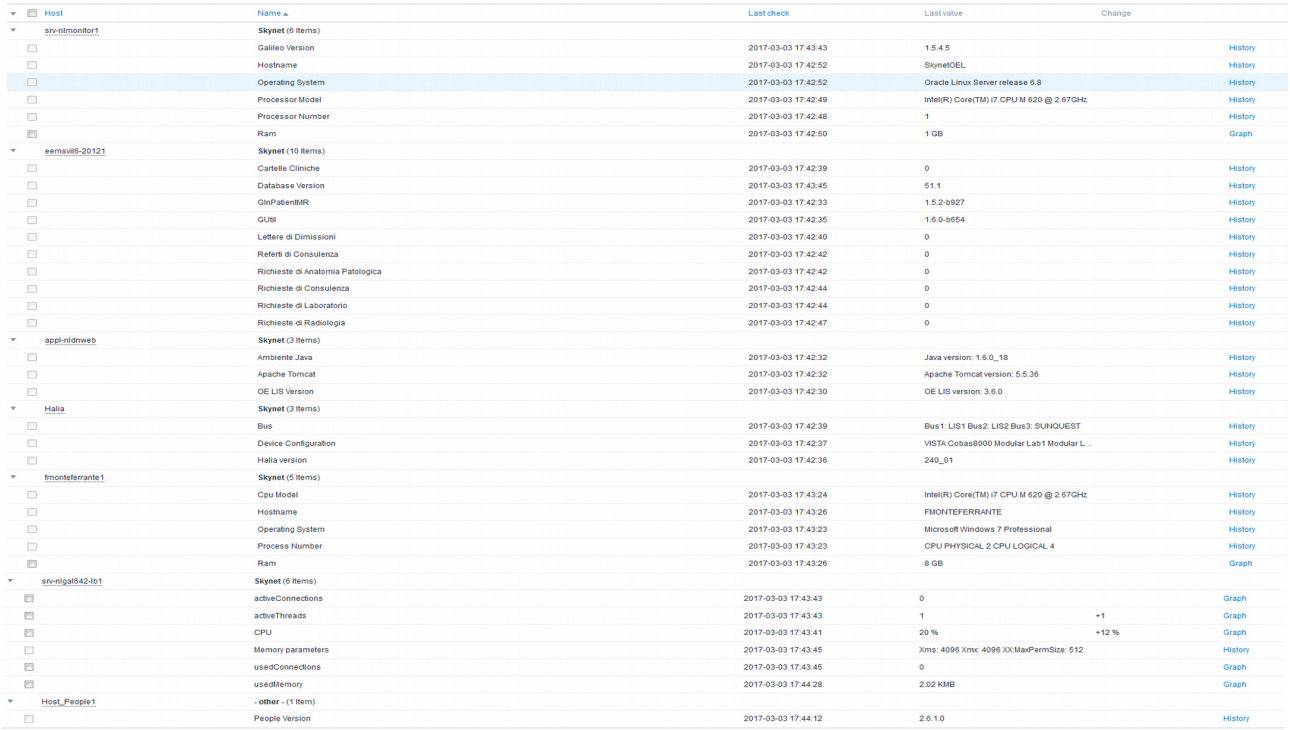
| Service | Status | Last Check | Duration | Attempt | Status Information |
|---|---|---|---|---|---|
| PEOPLE VERSION | OK | 10:03:50 | 0d 2h 20m 30s | 1/3 | OK - VERSION PEOPLE 2.6.1.0 |
| AMBIENTE JAVA DNWEB | OK | 17:14:53 | 24d 4h 1m 18s | 1/3 | OK - VERSION AMBIENTE JAVA 1.6.0_18 |
| APACHE TOMCAT DNWEB | OK | 17:16:00 | 1d 3h 17m 39s | 1/3 | OK - VERSION APACHE TOMCAT 5.5.36 |
| OE LIS DNWEB | OK | 17:19:31 | 24d 4h 1m 6s | 1/3 | OK - VERSION OE LIS DNWEB 3.6.0 |
| BUS | OK | 17:30:07 | 24d 4h 4m 37s | 1/3 | OK - 1-LIS1 2-LIS2 3-SUNQUEST |
| CARTELLE CLINICHE | OK | 17:03:38 | 24d 4h 6m 11s | 1/3 | OK - value: 0 |
| DEVICES CONFIGURATION HALIA | OK | 17:14:53 | 24d 3h 57m 20s | 1/3 | OK - LIST DEVICES:VISTA#Cobas8000#Modular Lab1#Modular Lab2#XE2100#Siemens Vista1-Lab1#Siemens Vista2-Lab1#Siemens Vista1-Lab2#Siemens Vista2-Lab2#Navios 1#Navios 2#Navios 3#Navios |
| GDnPatientMR | OK | 17:20:02 | 24d 4h 4m 37s | 1/3 | OK - MODULE GDnPatientMR 1.5.2-b027 |
| GUNI | OK | 17:23:33 | 24d 4h 4m 37s | 1/3 | OK - MODULE GUNI 1.6.0-b554 |
| HALIA VERSION | OK | 17:27:05 | 24d 3h 59m 21s | 1/3 | OK - VERSION HALIA 2.4.0.1 |
| LETTERE DI DIMISSIONI | OK | 17:00:37 | 24d 4h 4m 36s | 1/3 | OK - value: 0 |
| REFERTI DI CONSULENZA | OK | 17:04:09 | 24d 4h 4m 36s | 1/3 | OK - value: 0 |
| RICHIESTE DI ANATOMIA PATOLOGICA | OK | 17:09:57 | 24d 4h 4m 36s | 1/3 | OK - value: 0 |
| RICHIESTE DI CONSULENZA | OK | 17:14:53 | 24d 4h 6m 11s | 1/3 | OK - value: 0 |
| RICHIESTE DI LABORATORIO | OK | 17:17:00 | 24d 4h 4m 36s | 1/3 | OK - value: 0 |
| RICHIESTE DI RADIOLOGIA | OK | 17:20:32 | 24d 4h 4m 36s | 1/3 | OK - value: 0 |
| HSO OPERATING SYSTEM CPU MODEL | OK | 17:04:39 | 0d 2h 18m 12s | 1/3 | OK - CPU MODEL: Intel(R) Core(TM) i7 CPU M 620 @ 2.67GHz |
| HSO OPERATING SYSTEM CPU NUMBER | OK | 17:12:08 | 0d 2h 18m 12s | 1/3 | OK - CPU PHYSICAL: 2 CPU LOGICAL: 4 |
| HSO OPERATING SYSTEM HOSTNAME | OK | 17:14:53 | 0d 2h 18m 12s | 1/3 | OK - HOSTNAME: FMONTEFERRANTE |
| HSO OPERATING SYSTEM OS | OK | 17:17:30 | 0d 2h 18m 12s | 1/3 | OK - OS VERSION: Microsoft Windows 7 Professional |
| HSO OPERATING SYSTEM RAM | OK | 17:21:02 | 0d 2h 18m 12s | 1/3 | OK - RAM SIZE: 8.0 GB |
| ACTIVE_CONNECTIONS | OK | 17:24:34 | 24d 4h 4m 37s | 1/3 | OK - PARAMETER activeConnections: 1.0 |
| ACTIVE_THREADS | OK | 17:28:06 | 24d 4h 4m 37s | 1/3 | OK - PARAMETER activeThreads: 0.0 |
| CPU | OK | 17:25:56 | 0d 0h 4m 24s | 1/3 | OK - PARAMETER cpu: 10.0% |
| GALILEO MEMORY | OK | 17:05:20 | 24d 3h 54m 22s | 1/3 | OK - Xms=4096MB Xmx=4096MB XX:MaxPermSize=512MB |
| USED_CONNECTIONS | OK | 17:11:08 | 24d 4h 4m 36s | 1/3 | OK - PARAMETER usedConnections: 0.0 |
| USED_MEMORY | OK | 17:28:42 | 0d 0h 1m 38s | 1/3 | OK - PARAMETER usedMemory: 1064MB |
| GALILEO VERSION | OK | 17:25:34 | 24d 3h 50m 51s | 1/3 | OK - VERSION GALILEO 1.5.4.5 |
| HSO OPERATING SYSTEM REMOTE SSH CPU MODEL | OK | 17:06:12 | 24d 3h 52m 50s | 1/3 | OK - CPU MODEL: Intel(R) Core(TM) i7 CPU M 620 @ 2.67GHz |
| HSO OPERATING SYSTEM REMOTE SSH CPU NUMBER | OK | 17:11:58 | 24d 3h 42m 16s | 1/3 | OK - CPU NUMBER: 1 |
| HSO OPERATING SYSTEM REMOTE SSH HOSTNAME | OK | 17:15:29 | 24d 3h 55m 38s | 1/3 | OK - HOSTNAME: SkynetOEL |
| HSO OPERATING SYSTEM REMOTE SSH OS | OK | 17:19:01 | 24d 3h 54m 39s | 1/3 | OK - OS: Oracle Linux Server release 5.8 |
| HSO OPERATING SYSTEM REMOTE SSH RAM | OK | 17:22:33 | 24d 3h 53m 39s | 1/3 | OK - RAM SIZE: 1 GB |
| VERSION DB GALILEO | OK | 17:19:43 | 0d 0h 9m 5s | 1/3 | OK - VERSION DB GALILEO 51.1 |

*Figura 128: Service Skynet realizzati*

Dalla Figura 128, si può osservare come tutti i service, in Skynet, funzionano in modo corretto e permettono di ottenere le informazione desiderate.

È possibile, inoltre, notare come l'output, che deve essere correttamente formattato, venga mostrato sotto forma di barra colorata di verde; infatti, in base alla realizzazione dello script e al modo in cui si è deciso di gestire lo stato di un service, Nagios analizzerà l'output e restituirà il risultato esattamente come mostrato.

Per quanto riguarda Zabbix, invece:



| Host | Name | Last check | Last value | Change | |
|---|---|---|---|---|---|
| srv-nlmonitor1 | **Skynet** (6 Items) | | | | |
| | Galileo Version | 2017-03-03 17:43:43 | 1.5.4.5 | | History |
| | Hostname | 2017-03-03 17:42:52 | SkynetOEL | | History |
| | Operating System | 2017-03-03 17:42:52 | Oracle Linux Server release 6.8 | | History |
| | Processor Model | 2017-03-03 17:42:49 | Intel(R) Core(TM) i7 CPU M 620 @ 2.67GHz | | History |
| | Processor Number | 2017-03-03 17:42:48 | 1 | | History |
| | Ram | 2017-03-03 17:42:50 | 1 GB | | Graph |
| eemsvil6-20121 | **Skynet** (10 Items) | | | | |
| | Cartelle Cliniche | 2017-03-03 17:42:39 | 0 | | History |
| | Database Version | 2017-03-03 17:43:45 | 51.1 | | History |
| | GInPatientMR | 2017-03-03 17:42:33 | 1.5.2-b927 | | History |
| | GUtil | 2017-03-03 17:42:35 | 1.6.0-b654 | | History |
| | Lettere di Dimissioni | 2017-03-03 17:42:40 | 0 | | History |
| | Referti di Consulenza | 2017-03-03 17:42:42 | 0 | | History |
| | Richieste di Anatomia Patologica | 2017-03-03 17:42:42 | 0 | | History |
| | Richieste di Consulenza | 2017-03-03 17:42:44 | 0 | | History |
| | Richieste di Laboratorio | 2017-03-03 17:42:44 | 0 | | History |
| | Richieste di Radiologia | 2017-03-03 17:42:47 | 0 | | History |
| appl-nldnweb | **Skynet** (3 Items) | | | | |
| | Ambiente Java | 2017-03-03 17:42:32 | Java version: 1.6.0_18 | | History |
| | Apache Tomcat | 2017-03-03 17:42:32 | Apache Tomcat version: 5.5.36 | | History |
| | OE LIS Version | 2017-03-03 17:42:30 | OE LIS version: 3.6.0 | | History |
| Halia | **Skynet** (3 Items) | | | | |
| | Bus | 2017-03-03 17:42:39 | Bus1: LIS1 Bus2: LIS2 Bus3: SUNQUEST | | History |
| | Device Configuration | 2017-03-03 17:42:37 | VISTA Cobas8000 Modular Lab1 Modular L... | | History |
| | Halia version | 2017-03-03 17:42:36 | 240_01 | | History |
| fmonteferrante1 | **Skynet** (5 Items) | | | | |
| | Cpu Model | 2017-03-03 17:43:24 | Intel(R) Core(TM) i7 CPU M 620 @ 2.67GHz | | History |
| | Hostname | 2017-03-03 17:43:26 | FMONTEFERRANTE | | History |
| | Operating System | 2017-03-03 17:43:23 | Microsoft Windows 7 Professional | | History |
| | Process Number | 2017-03-03 17:43:23 | CPU PHYSICAL:2 CPU LOGICAL:4 | | History |
| | Ram | 2017-03-03 17:43:26 | 8 GB | | Graph |
| srv-nlgal642-lb1 | **Skynet** (6 Items) | | | | |
| | activeConnections | 2017-03-03 17:43:43 | 0 | | Graph |
| | activeThreads | 2017-03-03 17:43:43 | 1 | +1 | Graph |
| | CPU | 2017-03-03 17:43:41 | 20 % | +12 % | Graph |
| | Memory parameters | 2017-03-03 17:43:45 | Xms: 4096 Xmx: 4096 XX:MaxPermSize: 512 | | History |
| | usedConnections | 2017-03-03 17:43:45 | 0 | | Graph |
| | usedMemory | 2017-03-03 17:44:28 | 2.02 KMB | | Graph |
| Host_People1 | - **other** - (1 Item) | | | | |
| | People Version | 2017-03-03 17:44:12 | 2.6.1.0 | | History |

*Figura 129: Item Zabbix realizzati*

Anche in questo caso, si può notare come gli item sono tutti funzionanti e restituiscono gli stessi risultati forniti dai service presenti in Skynet. L'aspetto interessante in questo caso è che viene fornito solo il risultato dell'item senza interessarsi troppo allo stato dell'item, in quanto questo verrà gestito attraverso la realizzazione dei trigger.

Altro aspetto importante da considerare, per quanto riguarda i risultati sperimentali, è il popolamento che alcuni service/item effettuano su alcuni campi del CMDB.

Per il service GALILEO VERSION, questo esegue sulla macchina srv-nlmonitor pertanto andrà a modificare il corrispondente campo:

| # | CI Name | Company | Status | CI Type | CI Sub Type | High Version | Main Version | Patch | Hotfix |
|---|---|---|---|---|---|---|---|---|---|
| 557 | Galileo 1.4 | BRESCIA (BS) POLIAMBULANZA | Active | NL Product - Galileo | Galileo | 1 | 4 | 4 | 0 |
| 5772 | Nodo 1 Galileo | BRESCIA (BS) POLIAMBULANZA | Active | NL Product - Galileo | Galileo | 1 | 5 | 4 | 5 |
| 5773 | Nodo 2 Galileo | BRESCIA (BS) POLIAMBULANZA | Active | NL Product - Galileo | Galileo | 1 | 5 | 4 | 0 |
| 8176 | Nodo 3 Galileo | BRESCIA (BS) POLIAMBULANZA | Active | NL Product - Galileo | Galileo | 1 | 5 | 4 | 0 |
| 10443 | srv-nlgal643-lb | BRESCIA (BS) POLIAMBULANZA | Active | NL Product - Galileo | Galileo | 0 | 0 | 0 | 0 |
| 10444 | srv-nlgal642-lb | BRESCIA (BS) POLIAMBULANZA | Active | NL Product - Galileo | Galileo | 0 | 0 | 0 | 0 |
| 10445 | srv-nlgal643-lb1 | BRESCIA (BS) POLIAMBULANZA | Active | NL Product - Galileo | Galileo | 0 | 0 | 0 | 0 |
| 10446 | srv-nlgal642-lb1 | BRESCIA (BS) POLIAMBULANZA | Active | NL Product - Galileo | Galileo | 0 | 0 | 0 | 0 |
| 10447 | dboracle | BRESCIA (BS) POLIAMBULANZA | Active | NL Product - Galileo | Galileo | 0 | 0 | 0 | 0 |
| 10459 | srv-nlmonitor | BRESCIA (BS) POLIAMBULANZA | Active | NL Product - Galileo | Galileo | 1 | 5 | 4 | 5 |

*Figura 130: Galileo Version*

Come si può notare dalla Figura 130, la macchina srv-nlmonitor presenta i parametri:

- HighVersion pari a 1;
- MainVersion pari a 5;

- Patch pari a 4;
- Hotfix pari a 5.

Questi parametri messi insieme forniscono la versione 1.5.4.5 che è esattamente quella fornita dal service/item GALILEO VERSION; infatti, il service quando esegue controlla che la versione sia diversa e se così fosse la va a modificare in base al nuovo valore ottenuto.

Per il service GALILEO MEMORY si ha:

| # | CI Name | Company | Status | CI Type | CI Sub Type | High Version | Main Version | Patch | Hotfix | CI Cust Int 11 | CI Cust Int 12 | CI Cust Int 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 557 | Galileo 1.4 | BRESCIA (BS) | Active | NL Product - Galileo | Galileo | 1 | 4 | 4 | 0 | 0 | 0 | 0 |
| 5772 | Nodo 1 Galileo | BRESCIA (BS) | Active | NL Product - Galileo | Galileo | 1 | 5 | 4 | 5 | 0 | 0 | 0 |
| 5773 | Nodo 2 Galileo | BRESCIA (BS) | Active | NL Product - Galileo | Galileo | 1 | 5 | 4 | 0 | 1024 | 1024 | 512 |
| 8176 | Nodo 3 Galileo | BRESCIA (BS) | Active | NL Product - Galileo | Galileo | 1 | 5 | 4 | 0 | 0 | 0 | 0 |
| 10443 | srv-nlgal643-lb | BRESCIA (BS) | Active | NL Product - Galileo | Galileo | 0 | 0 | 0 | 0 | 4096 | 4096 | 512 |
| 10444 | srv-nlgal642-lb | BRESCIA (BS) | Active | NL Product - Galileo | Galileo | 0 | 0 | 0 | 0 | 4096 | 4096 | 512 |
| 10445 | srv-nlgal643-lb1 | BRESCIA (BS) | Active | NL Product - Galileo | Galileo | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10446 | srv-nlgal642-lb1 | BRESCIA (BS) | Active | NL Product - Galileo | Galileo | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10447 | dboracle | BRESCIA (BS) | Active | NL Product - Galileo | Galileo | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10459 | srv-nlmonitor | BRESCIA (BS) | Active | NL Product - Galileo | Galileo | 1 | 5 | 4 | 5 | 0 | 0 | 0 |

*Figura 131: Galileo Memory*

Questo service/item, invece, esegue sulla macchina srv-nlgal642-lb, di conseguenza, dalla Figura 131, si possono notare i campi:

- CI Cust Int 11 corrisponde a Xms e ha valore 4096;
- CI Cust Int 12 corrisponde a Xmx e ha valore pari a 4096;
- CI Cust Int 13 corrisponde a XX:MaxPermSize e ha valore pari a 512.

Anche in questo caso, la scrittura da parte del service è avvenuta con successo, quindi si controlla se questi parametri sono uguali o meno a quelli ottenuti in seguito all'esecuzione del service e se così non fosse viene effettuata la scrittura.

Per il service/item HALIA VERSION:

| # | CI Name | Company | Status | CI Type | CI Sub Type | High Version | Main Version | Patch | Hotfix |
|---|---|---|---|---|---|---|---|---|---|
| 543 | Halia | BRESCIA (BS) POLIAMBULANZA | Active | NL Product - Halia | Halia | 2 | 1 | 0 | 4939 |
| 10448 | eemsvil6-2012 | BRESCIA (BS) POLIAMBULANZA | Active | NL Product - Halia | Halia | 2 | 4 | 0 | 1 |
| 10449 | fmonteferrante | BRESCIA (BS) POLIAMBULANZA | Active | NL Product - Halia | Halia | 0 | 0 | 0 | 0 |
| 10450 | Halia | BRESCIA (BS) POLIAMBULANZA | Active | NL Product - Halia | Halia | 0 | 0 | 0 | 0 |
| 10451 | fmonteferrante1 | BRESCIA (BS) POLIAMBULANZA | Active | NL Product - Halia | Halia | 0 | 0 | 0 | 0 |
| 10452 | eemsvil6-20121 | BRESCIA (BS) POLIAMBULANZA | Active | NL Product - Halia | Halia | 0 | 0 | 0 | 0 |

*Figura 132: Halia Version*

Tale service/item esegue sulla macchina eemsvil6-2012, infatti, si può notare come è avvenuta la scrittura della versione attraverso i parametri:

- HighVersion assume valore 2;
- MainVersion assume valore 4;
- Patch assume valore 0;

- Hotfix assume valore 1.

La versione risultante, quindi, sarà, 2.4.0.1 che, osservando le Figure 128 e 129, risulta essere uguale a quella monitorata. Anche in questo caso la scrittura è avvenuta con successo.

Per il service/item DEVICES CONFIGURATION HALIA:

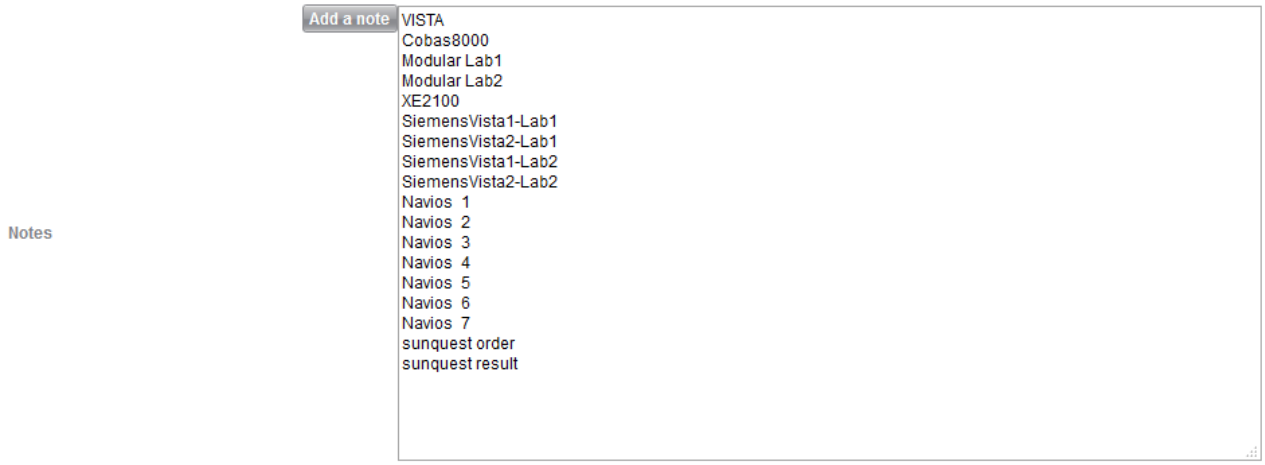



*Figura 133: Devices Configuration Halia*

Anche in questo caso, si può notare come nel campo notes sono stati memorizzati tutti i dispositivi medicali. In questa circostanza, però, per mostrarli tutti bisogna entrare all'interno del corrispondente CI perché nella vista generale non è possibile vederli. Anche in questo caso, quindi, la scrittura è avvenuta con successo.

Per i service/item AMBIENTE JAVA DNWEB, APACHE TOMCAT DNWEB e OE LIS VERSION si ha:

| # | CI Name | Company | Status | CI Type | CI Sub Type | High Version | Main Version | Patch | Hotfix | CI Cust Text 25 | CI Cust Int 14 | CI Cust Int 15 | CI Cust Int 16 | CI Cust Int 17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 548 | DNWeb | BRESCIA (BS) | Active | NL Product - | DNWeb | 3 | 6 | 1 | 0 | | 0 | 0 | 0 | 0 |
| 10453 | app-nldnweb | BRESCIA (BS) | Active | NL Product - | DNWeb | 3 | 6 | 0 | 0 | 1.6.0_18 | 5 | 5 | 36 | 0 |
| 10454 | appl-nldnweb | BRESCIA (BS) | Active | NL Product - | DNWeb | 0 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 |

*Figura 134: Ambiente Java, Apache Tomcat OE LIS Version di Dnweb*

Questi service/item eseguono tutti e tre sulla stessa macchina, ovvero, app-nldnweb. Dalla Figura 134, si possono osservare i diversi parametri:

- HighVersion pari a 3;
- MainVersion pari a 6;
- Patch pari a 0;
- Hotfix pari a 0.

Questi primi parametri rappresentano la versione OE LIS che sarà 3.6.0.0 come si può notare dalle Figure 128 e 129.

Il parametro CI Cust Text 25 rappresenta la versione dell'ambiente Java ed è pari a 1.6.0_18 ed, infine, i parametri:

- CI Cust Int 14 pari a 5;
- CI Cust Int 15 pari a 5;
- CI Cust Int 16 pari a 36

rappresentano la versione di Apache Tomcat che è 5.5.36.

Ultimo service/item di scrittura è PEOPLE VERSION:

| # | CI Name | Company | Status | CI Type | CI Sub Type | High Version | Main Version | Patch | Hotfix |
|---|---|---|---|---|---|---|---|---|---|
| 550 | People | BRESCIA (BS) POLIAMBULANZA | Active | NL Product - People | People | 2 | 2 | 0 | 0 |
| 10460 | Host_People | BRESCIA (BS) POLIAMBULANZA | Active | NL Product - People | People | 2 | 6 | 1 | 0 |

*Figura 135: People Version*

Tale service/item esegue sulla macchina Host_People, di conseguenza, si può osservare, dalla Figura 135, la memorizzazione dei parametri:

- HighVersion pari a 2;
- MainVersion pari a 6;
- Patch pari a 1;
- Hotfix pari a 0.

Ricomponendo i valori, si ottiene la versione di People che risulta essere 2.6.0.1 esattamente come mostrato nelle Figure 128 e 129.

Anche in questo caso, la scrittura è avvenuta con successo.

Ultimo aspetto importante, per quanto riguarda i risultati, è che sono stati calcolati anche i tempi di esecuzione di ciascun service/item. Tutti i valori si trovano all'interno dei file di log *nagios_time.log* e *zabbix_time.log*, però essendone molti ne vengono mostrati alcuni per farsi un'idea di come sono strutturati i file. Per quanto riguarda il file *nagios_time.log*:

```
DATE: 2017-02-07 14:12:28
IP: 10.69.248.40    Time to obtain AMBIENTE JAVA DNWEB: 44 ms

DATE: 2017-02-07 14:12:28
IP: 10.69.248.40    Time to obtain OE LIS DNWEB: 50 ms

DATE: 2017-02-07 14:12:28
IP: 10.69.248.40    Time to obtain APACHE TOMCAT DNWEB: 45 ms

DATE: 2017-02-07 14:14:28
IP: 192.168.223.155 Time to obtain usedMemory: 357 ms

DATE: 2017-02-07 14:14:29
IP: 192.168.223.155 Time to obtain activeThreads: 378 ms

DATE: 2017-02-07 14:14:29
IP: 192.168.223.155 Time to obtain cpu: 352 ms

DATE: 2017-02-07 14:14:29
IP: 192.168.223.155 Time to obtain activeConnections: 389 ms

DATE: 2017-02-07 14:14:28
IP: 192.168.223.155 Time to obtain usedConnections: 3179 ms

DATE: 2017-02-07 14:15:30
IP: 192.168.223.155 Time to obtain Xms-Xmx-XX:MaxPermSize: 351 ms

DATE: 2017-02-07 14:17:14
QUERY: select * from v_sk_res_let
Time to obtain value: 43 ms
```

*Figura 136: Tempi di ottenimento dei dati di alcuni service*

In questa figura, si può notare come è strutturato il file di log. Si ha la data che indica il momento in cui viene fatto il check in Skynet, l'IP della macchina su cui esegue il service ed, infine, il tempo, in millisecondi, impiegato per ottenere l'informazione di interesse.

In altri casi, invece, si può trovare la query effettuata con il tempo impiegato, o ancora, si può trovare il nome del database su cui viene effettuata l'operazione; tutto dipende dal service che viene eseguito.

Lo stesso discorso viene fatto per quanto riguarda il file di log di Zabbix, cioè *zabbix_time.log*:

```
DATE: 2017-02-03 16:57:48
IP: 192.168.240.146 Time to obtain numproc: 201 ms

DATE: 2017-02-03 16:57:49
IP: 192.168.240.146 Time to obtain procmodel: 159 ms

DATE: 2017-02-03 16:57:50
IP: 192.168.240.146 Time to obtain ram: 156 ms

DATE: 2017-02-03 16:57:51
IP: 192.168.240.146 Time to obtain os: 167 ms

DATE: 2017-02-03 16:57:52
IP: 192.168.240.146 Time to obtain hostname: 136 ms

DATE: 2017-02-03 16:58:22
IP: noemalife   Time to obtain wos: 247 ms

DATE: 2017-02-03 16:58:23
IP: noemalife   Time to obtain wnumproc: 237 ms

DATE: 2017-02-03 16:58:24
IP: noemalife   Time to obtain wcpumodel: 228 ms

DATE: 2017-02-03 16:58:25
IP: noemalife   Time to obtain wram: 245 ms

DATE: 2017-02-03 16:58:26
IP: noemalife   Time to obtain whostname: 234 ms
```

*Figura 137: Tempi di ottenimento dei dati di alcuni item*

La struttura dei file di log è pressoché identica a quella di *nagios_time.log* proprio per mantenere allineati i risultati dei due sistemi di monitoraggio.

# Conclusioni

La sfida del lavoro svolto durante la tesi è stata quella di trovare una soluzione per cercare di automatizzare il salvataggio di dati monitorati attraverso il sistema di monitoraggio e non solo, in modo da evitare che questo lavoro venisse svolto manualmente.

Il problema principale è rappresentato dal fatto che la trascrizione dei dati monitorati viene fatta a mano dal personale che si occupa del monitoraggio rallentando, di fatto, l'assistenza verso i clienti e portando, a volte, ad avere dei dati che non risultano aggiornati perché, ad esempio, la memorizzazione delle informazioni non è stata ancora effettuata.

L'obiettivo principale del progetto è quella di trovare e realizzare una soluzione che automatizzasse questo lavoro in tempi relativamente ridotti, in modo da semplificare e velocizzare il salvataggio di dati.

La soluzione a questo problema è stata trovata nella realizzazione di un server Web che ha il compito di allineare specifici dati, monitorati con il SysAid CMDB e nella realizzazione di service che permettono di aggiungere ulteriori informazioni con lo scopo di semplificare le necessità aziendali continuando a soddisfare le richieste dei clienti.

Tra i diversi service realizzati, ci sono quelli che vengono utilizzati per arricchire le informazioni iniziali che sono state memorizzate dal server Web all'interno del SysAid CMDB e altri che servono per il monitoraggio e controllo di parametri di interesse per l'azienda.

Gli obiettivi che sono stati posti all'inizio del progetto sono stati raggiunti con grandissimo successo. Il server è in grado di controllare quali sono gli host che sono correntemente attivi e monitorati attraverso il sistema di monitoraggio Skynet, su quali host monitorati esegue un particolare service, stabilire quali sono i middleware, quali sono gli host su cui sono presenti determinati prodotti aziendali e quali sono i database a cui la macchina, su cui esegue il server, riesce a collegarsi.

Tutte queste informazioni, successivamente, vengono memorizzate all'interno del SysAid CMDB consentendo al personale del customer service di accedere a tali dati ed assistere clienti che hanno diverse problematiche sulle proprie macchine.

Con l’aggiunta dei service realizzati, inoltre, c’è la possibilità di andare a popolare altri campi di interesse. In questa circostanza, quando un servizio esegue su una macchina, il risultato del service viene memorizzato all’interno del SysAid CMDB nel corrispondente campo del CI che rappresenta la macchina stessa. Inoltre, prima di effettuare la scrittura, il service provvede a controllare se il dato già memorizzato è diverso da quello restituito o meno. In caso di diversità, la scrittura viene effettuata altrimenti no, in quanto risulterebbe inutile salvare un qualcosa che nel tempo non è cambiato.

Infine, anche l’integrazione nel mondo di Zabbix ha avuto successo. Il server Web è in grado di controllare quali sono gli host correntemente attivi e monitorati dal sistema, su quali macchine esegue un determinato servizio e quali sono gli host su cui sono presenti determinati prodotti aziendali. Per i service realizzati, invece, inseriti all’interno di Zabbix e monitorando le stesse macchine, forniscono gli stessi risultati che si hanno con Skynet.

Per quando riguarda gli obiettivi futuri, in primo luogo si può dire che lo studio di Zabbix con conseguente integrazione del progetto al suo interno, ha suscitato molto interesse da parte del personale aziendale; infatti, hanno deciso di installare il sistema di monitoraggio per monitorare, inizialmente, tutti i dispositivi presenti in azienda e magari in futuro effettuare completamente il passaggio da Skynet a Zabbix stesso.

In secondo luogo, si può affermare che sia il server Web che i service realizzati hanno una base molto solida, nel senso che sono stati perfettamente integrati in Zabbix nonostante la realizzazione fosse predisposta per Skynet. Tutto questo è stato possibile effettuando semplici modifiche del codice sorgente senza dover riscrivere centinaia di righe di codice, pertanto si può affermare che come sia stato possibile integrare il tutto in Zabbix, allora esiste anche la possibilità di integrarlo in altri sistemi di monitoraggio. Inoltre, poiché in questo ambito c’è la possibilità di monitorare tutto ciò che è di interesse personale e aziendale, il server Web può essere esteso modificando il codice di base in modo da consentire il controllo e la scrittura di qualsiasi cosa, mentre per i service si può affermare che se ne possono realizzare a migliaia in base ai propri interessi e modificandoli questi possono andare anche a popolare diversi campi all’interno del SysAid CMDB.

# Bibliografia

[1] Ethan Galstad: "Nagios Core Version 3.x Documentation", Nagios Core Development Team and Community Contributors, 2009

[2] "GROUNDWORK MONITOR ARCHITECT (MONARCH) Administrator Guide 2.0", https://www.question-defense.com/documentation/monarch-documentation

[3] "Zabbix Documentation 3.2 - Host", https://www.zabbix.com/documentation/3.2/manual/config/hosts/host

[4] "Zabbix Documentation 3.2 - Triggers", https://www.zabbix.com/documentation/3.2/manual/config/triggers

[5] "Zabbix Documentation 3.2 - Requirements", https://www.zabbix.com/documentation/3.2/manual/installation/requirements

[6] A SOAP client e server for Node.js, https://github.com/vpulim/node-soap/blob/master/Readme.md