Capitolo 2

INFRASTRUCTURE AS CODE E ANSIBLE

Questo capitolo affronta in linea generale del concetto alla base di Infrastructure as Code, partendo da una sua definizione, passando per l’identificazione dei problemi principali di una gestione manuale di un’infrastruttura IT.

Si analizza una soluzione parziale per poi arrivare, successivamente, a quella fornita dall’approccio utilizzato con l’infrastructure as code, introducendo brevemente i tools esistenti che permettono tale approccio. A completamento, si illustrano le principali funzionalità dello strumento al momento più usato in ambito IaC: Ansible.

## Infrastructure as Code (IaC): definizione

Con Infrastructure as Code (IaC) intendiamo un approccio alla gestione e al provisioning dell'infrastruttura IT che avviene tramite codice anziché con processi manuali. I file di configurazione creati con l'IaC contengono le specifiche dell'infrastruttura e semplificano la modifica e la distribuzione delle configurazioni. Questa soluzione garantisce la ripetibilità del provisioning dello stesso ambiente. Codificando e documentando le specifiche della configurazione, l'IaC facilita la gestione della configurazione e permette di evitare eventuali modifiche ad-hoc non documentate. Il controllo delle versioni è un aspetto importante dell'IaC e i file di configurazione dovrebbero essere sottoposti al controllo della sorgente come qualsiasi altro file di codice sorgente software.

Implementare l'infrastruttura in forma di codice permette la suddivisione in componenti modulari che, attraverso l'automazione, possono essere combinati in modi diversi.

Automatizzare il provisioning dell'infrastruttura con l'IaC significa esonerare gli sviluppatori dalle attività di provisioning e gestione di server, sistemi operativi, storage e altri componenti dell'infrastruttura ogni volta che sviluppano o distribuiscono un'applicazione.

Codificare l'infrastruttura significa creare un modello di provisioning ripetibile, un'attività che può essere eseguita manualmente ma anche velocizzata con uno strumento di automazione.[[1]](#footnote-1)

## I problemi della gestione delle infrastrutture IT

Storicamente, il management di una infrastruttura IT è stato sempre un processo manuale. I tecnici si occupavano fisicamente dell’installazione e della configurazione dei server. Una volta terminato il processo di configurazione delle macchine, come ad esempio il setup del sistema operativo e di tutte le applicazioni necessarie, si effettuava il deploy delle app. Naturalmente, questo processo manuale portava spesso a risultati inaspettati e ad una serie di problemi.

Il primo grande problema è dato dal costo. Era necessario assumere molti professionisti esperti per eseguire determinati task, dall’ingegnere del network ai tecnici di manutenzione hardware. Tutte queste persone, ovviamente, dovevano essere pagate, ma anche gestite. Ne consegue che ciò portava ad una maggiore complessità strutturale e comunicativa all’interno dell’organizzazione. Ciò risultava, spesso, in una spesa economica consistente, senza considerare i costi di mantenimento dell’infrastruttura.

Altri due problemi riguardano la scalabilità e la disponibilità. Alla fine, tutto si riduce a problemi di performance: poiché la configurazione manuale dei sistemi è molto lenta, le applicazioni spesso si trovano a dover affrontare picchi di accesso, mentre gli amministratori di sistema cercano disperatamente di configurare i server per la gestione del carico. Ciò, ovviamente, va ad impattare la disponibilità. Se l’organizzazione non disponesse di server di backup o addirittura data center, l’applicazione potrebbe non essere disponibile per lunghi periodi di tempo.

Un altro problema riguarda il monitoraggio e la visibilità delle prestazioni. Una volta configurata e completata l’infrastruttura, è necessario monitorarla affinché ci si possa assicurare che funzioni in maniera ottimale. Quando si verifica un problema, come si individua esattamente da dove proviene? È la rete, il server o l’applicazione? Strumenti come *Netreo* possono fornire piena visibilità sulle prestazioni dell’intera infrastruttura IT. Con il rilevamento la configurazione automatica di *Netreo* è possibile assicurarsi che non ci siano *blind spots* nell’ambiente e la mappatura della piattaforma, la correlazione degli eventi e l’analisi automatizzata delle cause principali consentono di individuare esattamente il punto dove si è verificato un problema.

Ultimo ma non meno importante è il problema dell’inconsistenza. Se ci sono diverse persone che implementano manualmente le configurazioni, le discrepanze saranno inevitabili.

## Il Cloud Computing: una possibile soluzione?

Il cloud computing è nato anche per alleviare alcuni dei problemi elencati in precedenza. Esso libera dalla costrizione di costruire e mantenere i propri data center, abbattendo i costi ad essi associati.

Tuttavia, il cloud computing è tutt’altro che una panacea. Sebbene permetta di configurare rapidamente le infrastrutture, risolvendo gravi problemi come quelli correlati a disponibilità e scalabilità, non fornisce una risoluzione per quelli legati all’inconsistenza. Come detto in precedenza, quando ci sono più figure che eseguono le configurazioni, la probabilità di riscontrare inconsistenze a livello di configurazione sono elevate.

## Infrastructure as Code: il pezzo mancante del puzzle

Prendendo in considerazione la definizione di IaC data in precedenza, definiamo il suo punto chiave: prima della IaC, il personale IT avrebbe dovuto modificare manualmente le configurazioni per gestire la propria infrastruttura. Probabilmente, il modo più utilizzato per gestire ciò, era quello di sviluppare script usa e getta per automatizzare alcune attività. Con IaC, la configurazione dell’infrastruttura assume la forma in un semplice file di codice. Siccome è solo testo, è semplice da modificare, copiare e distribuire.

Fino ad ora, sono stati trattati i problemi causati da un approccio manuale alla gestione dell’infrastruttura IT. È stato detto come il cloud computing sia una soluzione a parte di questi problemi; concludiamo sostenendo che IaC è l’ultimo pezzo del puzzle. Saranno di seguito analizzati alcuni dei vantaggi che le organizzazioni trarranno dall’adozione di una soluzione basata su IaC.

### Velocità

Il primo benefit significativo fornito da IaC è la velocità. Infrastructure as code consente di configurare rapidamente l’intera infrastruttura eseguendo uno script. È possibile farlo per ogni tipo di ambiente, dal development alla produzione, passando per lo staging, il QA e altro ancora. IaC può rendere efficiente l’intero ciclo di vita dello sviluppo del software.

### Consistenza

I processi manuali causano errori. Gli esseri umani non sono infallibili; la gestione manuale dell’infrastruttura IT risulterà in discrepanze, non importa quante attenzioni si pongono alla forma e agli standard di configurazione.

IaC risolve in parte questo problema, facendo in modo che i files di configurazione stessi siano l’unica “fonte di verità”. In questo modo, è garantito che le stesse configurazioni vengano distribuite più e più volte, senza discrepanze.

### Controllo di versione

Poiché è possibile assegnare una versione ai files di configurazione generati con IaC, si ha una piena tracciabilità delle modifiche effettuate su ciascuna configurazione. Non sarà necessario “indovinare” le modifiche precedentemente effettuate e quando sono state effettuate.

### Maggiore efficienza del ciclo di sviluppo software

Utilizzando IaC è possibile distribuire l’architettura infrastrutturale in più fasi. Ciò rende lo sviluppo del software più efficiente, portando la produttività del team a nuovi livelli.

I developers potranno usare la IaC per creare e avviare ambienti sandbox, i quali permetteranno loro di sviluppare in completo isolamento e sicurezza. Lo stesso vale per i professionisti del QA, i quali possono avere copie perfette degli ambienti di produzione in cui eseguire i test.

Infine, al momento della distribuzione, sarà possibile il push sia dell’infrastruttura sia del codice di produzione, in un singolo step.

### Costi

Uno dei principali vantaggi dell’IaC è senza dubbio la diminuzione dei costi di gestione dell’infrastruttura IT. Impiegando il cloud computing combinato con IaC, i costi si riducono drasticamente. Ciò perché non si deve tener conto delle spese riguardanti l’hardware, così come l’assunzione di personale preposto alla sua cura o l’affitto dello spazio per custodire un data center.

Ogni qualvolta che un’organizzazione ha professionisti ben pagati che svolgono attività che è possibile automatizzare, esse stanno sprecando denaro. Tutta la loro attenzione dovrebbe essere rivolta alle attività che vanno ad aggiungere valore all’intera organizzazione. Ed è qui che le strategie di automazione, tra cui la IaC, tornano utili. Impiegando tali strategie, si liberano le figure professionali dall’esecuzione di tali attività manuali, lente e soggette a errori, in modo che possano concentrarsi su ciò che conta di più.

## Come funziona Infrastructure as Code?

Gli strumenti IaC possono variare per quanto riguarda la modalità di funzionamento, ma, generalmente, è possibile suddividerli in due gruppi principali:

* *Imperative Programming Tools,* il quale definisce una sequenza di comandi o istruzioni in modo che l’infrastruttura possa raggiungere il risultato finale.
* *Declarative Programming Tools,* secondo il quale si “dichiara” il risultato desiderato, ovvero invece di delineare esplicitamente una sequenza di passaggi di cui l’infrastruttura ha bisogno per raggiungere il risultato finale, l’approccio dichiarativo mostra come appare tale risultato.

### Imperative Programming for IaC

Nella programmazione imperativa, si specifica un elenco di passaggi che lo strumento IaC deve seguire per eseguire il provisioning di una nuova risorsa. Si specifica allo strumento come creare ogni ambiente, utilizzando una sequenza imperativa di comandi. *Chef*[[2]](#footnote-2)è un popolare tool IaC imperativo, ma anche *Ansible*[[3]](#footnote-3)e *Salt*[[4]](#footnote-4) hanno un supporto per la programmazione imperativa.

La programmazione imperativa richiede, d’altra parte, maggiori conoscenze di scripting, perché è necessario scrivere comandi per ogni passaggio di provisioning. Ciò dà il controllo sul modo in cui si eseguono le attività dell’infrastruttura, il che è l’ideale quando si devono apportare piccole modifiche, ottimizzare per uno specifico scopo o tenere conto di stranezze del software.

Gli script IaC imperativi sono spesso anche meno idempotenti: i passaggi predefiniti possono portare a risultati diversi a seconda dell’ambiente. Inoltre, gli script IaC imperativi sono così espliciti che un errore in un passaggio può causare il fallimento dell’intera operazione.[[5]](#footnote-5)

### Declarative Programming for IaC

Nella programmazione dichiarativa, invece, si specificano il nome e le proprietà delle risorse dell’infrastruttura della quale si vuole eseguire il provisioning. Successivamente, lo strumento IaC calcola come ottenere da solo quel risultato finale. Ad esso si dichiara ciò che si vuole, ma non il modo in cui giungere al risultato finale. Alcuni esempi popolari di strumenti IaC che utilizzano il paradigma di programmazione dichiarativa, includono *Terraform, Puppet, Ansible, Salt e CloudFormation.*

Tale approccio è molto popolare nella IaC. Si definisce lo stato finale desiderato della configurazione finale e la soluzione IaC capisce autonomamente in che modo arrivarci. La programmazione dichiarativa è altamente idempotante o ripetibile, il che significa che si possono eseguire i comandi IaC più e più volte e ottenere comunque lo stesso risultato.

Il paradigma dichiarativo si adatta bene anche alla modifica delle configurazioni, perché i passaggi di provisioning dello strumento IaC non sono definiti in modo esplicito.

Il più grande svantaggio dell’approccio dichiarativo è dato dal fatto che si rinuncia al controllo sui singoli passaggi del processo di provisioning. Inoltre, non è la scelta migliore per piccole correzioni e aggiornamenti che possono essere gestiti da un semplice script CLI. In questi casi, la programmazione dichiarativa può complicare eccessivamente le cose e rallentare il processo. [[6]](#footnote-6)

## Ansible

Ansible è un software open source utilizzato per l’automazione di macchine. Può gestire l’installazione e la configurazione di qualsiasi componente di sistema, così come la definizione di procedure di deploy automatizzate. Ansible, a differenza di altri tools quali Chef o Puppet, basa la sua filosofia chiave su una parola: semplicità. Creato nel 2012 da Michael DeHaan, autore di Cobbler, è balzato subito tra i sistemi di configuration management più utilizzati, grazie ad alcuni punti focali su cui è stato realizzato:

* non necessita altro che di un accesso SSH tra la macchina controllore e i nodi da controllare e tutto si gestisce con semplici file di testo (*YAML*);
* è scritto in Python, che lo rende multipiattaforma e non dipendente dalla distribuzione, leggero e performante;
* non richiede conoscenza di linguaggi di programmazione, la sintassi *YAML* con cui si scrivono le istruzioni è di semplice lettura e comprensione;

Ha preso così tanto piede che è stata fondata la Ansible Inc. per commercializzare il supporto e sponsorizzare la soluzione e, successivamente, l’azienda è stata acquistata da Red Hat.[[7]](#footnote-7)



### Terminologia



#### Inventario

Si tratta della lista di macchine sulle quali Ansible può operare. Opzionalmente tali macchine possono essere raggruppate in modo da avere più gruppi di host. L’inventario può essere scritto sia in formato *ini* che in *YAML*;

#### Moduli

I moduli possono essere visti come i comandi che vengono eseguiti sulle macchine. Esistono moduli per l’installazione e la rimozione di pacchetti, deployment di file (o generazione del loro contenuto), gestione dei servizi e tanto altro. La lista completa è davvero corposa ed è disponibile sulla documentazione ufficiale di Ansible. Ogni modulo ha il suo set di opzioni e proprietà. Nella *Figura 1* a pagina successivaviene invocato il modulo *debug*, il quale permette di stampare a video messaggi. Questi messaggi possono essere usati per mostrare informazioni di debug, come, ad esempio, contenuti di una variabile o l’output message ritornato da un comando. Tale modulo, richiede che sia specificata una proprietà chiamata *msg*, che contiene il messaggio da mostrare.

#### Tasks

I task sono le operazioni che, di fatto, eseguiamo sulle macchine. Possono essere visti come la serie di comandi, in sequenza, che verranno lanciati sulla macchina. Un task, in sostanza, il richiamo di un modulo con particolari opzioni.

I task Ansible sono generalmente definiti all’interno di playbook. Nella figura in basso, è riportato un semplice task di esempio, che stampa un messaggio tramite il modulo *debug.*

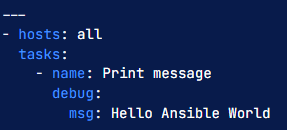


Figura 1 Esempio di Ansible task

Il task nella *Figura 1* è caratterizzato da un nome, il quale rappresenta l’operazione che sarà effettuata quando il task sarà eseguito.

#### Handlers

Le operazioni associate agli handler si differenziano dai task poiché non vengono sempre eseguite, ma vengono richiamate come reazione ad un evento di un task. Un banale esempio è quello di avviare un servizio dopo aver installato il software necessario, o riavviarlo dopo la modifica del suo file di configurazione.

#### Playbooks

I playbook sono una lista di task, handler e delle relative variabili legati a particolari gruppi di macchine in inventario. Vengono utilizzati per avere il “manuale di istruzioni” delle operazioni che Ansible dovrà eseguire.

#### Ruoli

I ruoli sono dei componenti dei playbook che raggruppano operazioni legate tra di loro con uno scopo specifico. Queste vengono di solito unificate per avere più riusabilità delle stesse. Ad esempio, l’installazione, la configurazione e la gestione del servizio NTP possono essere raggruppate in un ruolo, avendo la possibilità di riutilizzare questo ruolo in diversi Playbook.[[8]](#footnote-8)

1. Red Hat – *Introduzione all’automazione – Cosa si intende con Infrastructure as Code (IaC)?* (https://www.redhat.com/it/topics/automation/what-is-infrastructure-as-code-iac) [↑](#footnote-ref-1)
2. *Progress Chef – Automation Software for continuous Delivery of Secure Applications and Infrastructure* - https://www.chef.io/ [↑](#footnote-ref-2)
3. *Ansible – https://www.ansible.com/* [↑](#footnote-ref-3)
4. *Salt - https://repo.saltproject.io/* [↑](#footnote-ref-4)
5. Copado – *Declarative vs imperative programming for infrastructure as code, 2022 - https://www.copado.com/devops-hub/blog/declarative-vs-imperative-programming-for-infrastructure-as-code-iac* [↑](#footnote-ref-5)
6. *Ibidem.* [↑](#footnote-ref-6)
7. Michele Milaneschi*, Primi passi con Ansible*, 2018 [↑](#footnote-ref-7)
8. *Ibidem.* [↑](#footnote-ref-8)