***CAPITOLO 4***

In questo capitolo viene fornita un’applicazione pratica delle metodologie e degli strumenti propri del DevOps, presentati in forma teorica nei Capitoli 2 e 3.

Sarà creata da zero una piccola deployment pipeline utilizzando le funzionalità di Jenkins, un’applicazione viaggerà attraverso i vari stage della pipeline. Ciò su cui si vuole porre l’accento è l’utilizzo pratico dei tool presentati nel Capitolo 3, soprattutto quelli inerenti al DevOps. Non assume importanza, in questo elaborato, l’applicazione in sé, né il tipo né la complessità; una catena di produzione software snella ed efficiente, infatti, risulta estremamente scalabile e adattabile a progetti più o meno complessi.

*Architettura*

Jenkins  
dev 🡪 test 🡪 prod

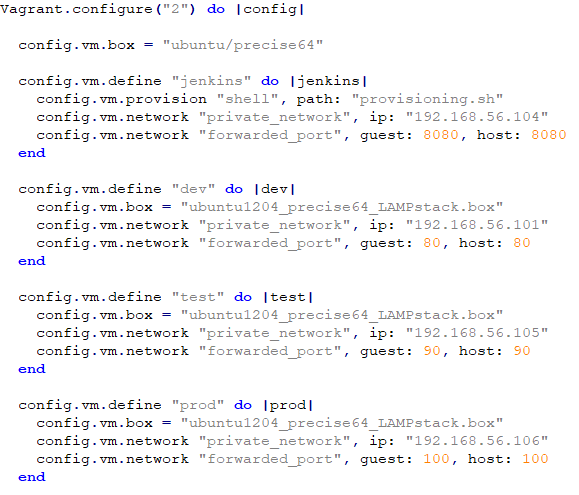
*Sviluppo*

*Creazione degli ambienti*

Per creare i vari nodi della catena ho utilizzato il concetto di macchina virtuale. In un contesto aziendale, nella stragrande maggioranza dei casi, ogni ambiente fa riferimento a diversi elementi hardware: server e pc per la parte di sviluppo, altri per la parte di test e così via. Nel mio caso, non disponendo di tutte queste risorse, ho usato una sola macchina host dalla quale ho creato quattro macchine virtuali:

* Macchina ‘jenkins’: è l’unica delle quattro macchine che non fa parte della pipeline. Su tale macchina è stato installato Jenkins, piattaforma sulla quale è stata creata e configurata la catena. Sulla macchina ‘jenkins’, inoltre, è presente il repository remoto del progetto. Creando un qualsiasi tipo di file, utile ai fini del lavoro, dalla macchina host, tramite una push tale file raggiungeva la macchina ‘jenkins’, da cui poi i vari nodi scaricheranno il materiale necessario.
* Macchina ‘dev’: è la macchina si occupa della compilazione del progetto e dell’esecuzione degli unit test.
* Macchina ‘test’: è la macchina che si occupa, tramite Selenium, degli acceptance test sulla parte web dell’applicazione.
* Macchina ‘prod’: è la macchina che si occupa, se il software non ha incontrato precedenti problemi, della messa in produzione.

Per la creazione delle macchine virtuali, appoggiandosi a VirtualBox, è stato utilizzato Vagrant. Il Vagrantfile trasforma in script il lavoro sistemistico di creazione e configurazione dei vari ambienti, rendendo estremamente semplice anche apportare qualche modifica.



(didascalia)

L’istruzione *config.vm.box* indica il sistema operativo della macchina ed eventuali programmi da installare immediatamente. Nel caso delle macchine dev, test e prod ho creato un .box che installa Ubunti12.04 e il famoso LAMPstack (Linux, Apache, MySQL e PHP). Per la macchina jenkins invece, ho utilizzato un’istruzione alternativa: *config.vm.provision* di tipo shell che specifica un file .sh\* (in questo caso provisioning.sh) da cui leggere le istruzioni per installare vari programmi nella macchina.



(didascalia)

Richiamando il file .sh vengono eseguite le istruzioni al suo interno in un bash linux.

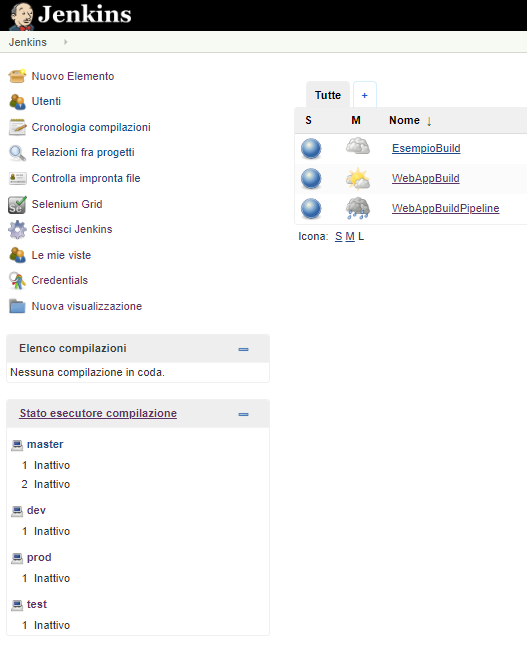
Con l’istruzione *config.vm.network* invece vengono configurate le impostazioni di rete, ad esempio l’indirizzo ip all’interno della rete privata.

Adesso è ancora più chiaro quanto sia semplificato il lavoro del sistemista, che non deve più compiere queste operazioni a mano su ciascuna macchina ma può utilizzare lo stesso script (che va messo sotto controllo di versione!). E’ altrettanto facile applicare aggiornamenti o modifiche a macchine già create, cambiando o aggiungendo qualche riga allo script.  
Con l’utilizzo di tool come Ansible, Chef o Puppet l’Infrastructure as a Code diventa uno strumento ancora più potente e permette di creare architetture elaborate che vanno oltre lo scopo di questo elaborato.

\* un file .sh in ambiente Linux equivale a un file .bat su Windows

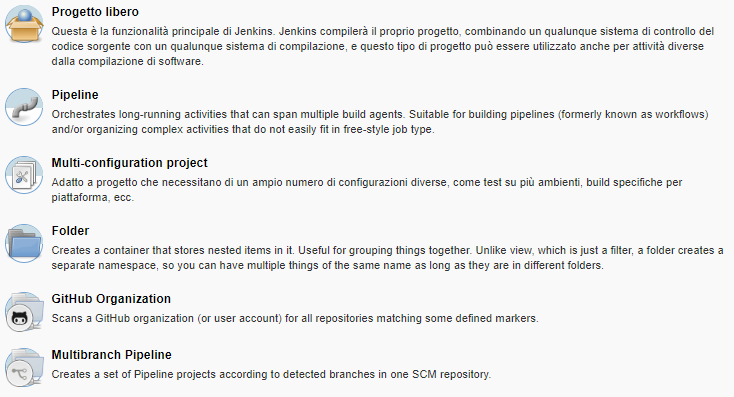
*Jenkins*

Come precedentemente specificato, la piattaforma che ho utilizzato per la creazione della pipeline è Jenkins, strumento per Continuous Integration e Continuous Delivery presentato nel Capitolo 3.   
È stato installato sulla macchina ‘jenkins’ in fase di creazione con Vagrant attraverso il file provisioning.sh mentre sulle altre macchine, ovvero i nodi della pipeline, è presente uno slave.jar di jenkins che permette di eseguire un job.



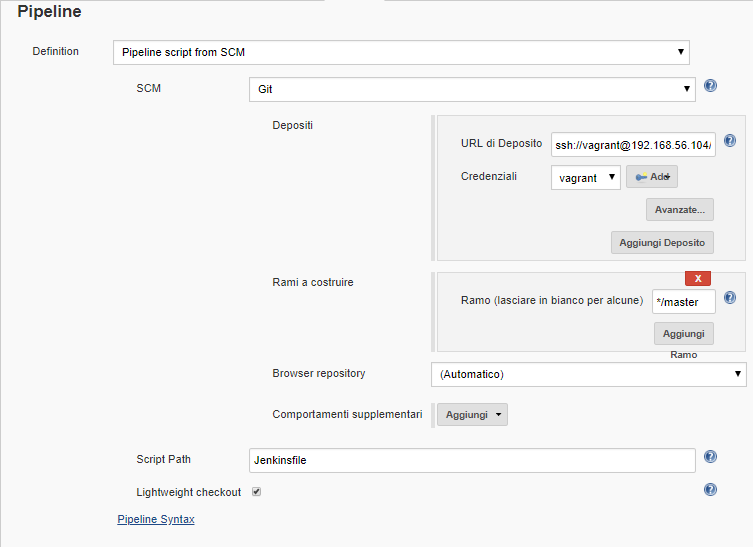
(didascalia)

Nella home di Jenkins si possono notare le macchine virtuali che stanno usando la piattaforma: master (jenkins), dev, test, prod.   
In alto sulla sinistra le voci del menu: ‘Nuovo Elemento’ per creare un nuovo Job, ‘Gestisci Jenkins’ per le configurazioni e per installare i plugin necessari.  
Sulla destra i progetti creati, in particolare quello preso di riferimento per questo Capitolo è il progetto ‘WebAppBuildPipeline’.



(didascalia)

Il Job ‘WebAppBuildPipeline’ è stato creato scegliendo la tipologia “Pipeline”, che permette di definire la nostra pipeline attraverso il Jenkinsfile, un file in cui, tramite script, vengono indicati i vari stage e step da eseguire.

Una volta creato il Job, bisogna per prima cosa configurarlo secondo le proprie necessità:  


Punti principali della configurazione:

* Con ‘Pipeline script from SCM’ specifichiamo che lo script che costituisce la pipeline non sarà scritto direttamente nella configurazione del Job, ma è contenuto in un repository di un Control Version System.
* Subito sotto viene indicato che la tipologia di SCM utilizzata è Git.
* Nella URL di Deposito ho inserito ‘ssh://vagrant@192.168.56.104/*path\_to\_repository*’. Ciò serve per specificare a Jenkins dove si trova il repository remoto che contiene i sorgenti del nostro progetto. Per questo progetto, come specificato precedentemente, il repository remoto si trova sulla macchina ‘jenkins’ perciò quando un nodo dovrà scaricare il materiale si collegherà via ssh a tale macchina (192.168.56.104 è infatti l’indirizzo ip privato assegnato alla macchina ‘jenkins’ nel Vagrantfile).