



fondo
sociale europeo



Report Finale “Laboratorio Integrato” Gruppo 6, Cloud Fiesta

De Iazzari, Dellerà, Oglietti,
Murta, Cafasso, Carrieri

2 febbraio 2022

in collaborazione con:



per una crescita intelligente,
sostenibile ed inclusiva

www.regione.piemonte.it/europa2020

INIZIATIVA CO-FINANZIATA CON FSE

Indice

1	Introduzione	2
1.1	Il Progetto	2
1.2	Il Team	2
2	Strumenti tecnico-organizzativi	4
2.1	GANTT e cronoprogramma	4
2.2	Strumenti di comunicazione	5
2.3	Organizzazione codice	5
2.4	Strumenti di scrittura	6
3	Componenti e architettura	7
3.1	In generale	7
3.1.1	Microsoft Azure	7
3.1.2	nopCommerce	7
3.2	Architettura di rete	8
3.3	Organizzazione container	8
4	Processo di implementazione	10
4.1	Brainstorming e ricerca	10
4.2	Prototipazione e scripting	11
4.3	Implementazione locale	11
4.4	Implementazione remota	12
4.5	Test di sicurezza	14
4.5.1	Analisi del perimetro	14
5	Penetration testing	16
5.1	Le premesse	16
5.2	Information gathering	16

Capitolo 1

Introduzione

1.1 Il Progetto

Il qui presente report ha lo scopo di illustrare lo svolgimento del progetto a opera del gruppo “Cloud Fiesta”, il progetto e’ stato commissionato dai docenti Blanchietti Andrea e Zimuel Enrico nell’ambito del corso “*Laboratorio Integrato*”.

Lo scopo del progetto e’ quello di realizzare una piattaforma di *e-commerce* per conto di un azienda che si occupa di commercio al dettaglio, il sistema deve essere *scalabile* in maniera da poter limitare i costi a quanto strettamente necessario e potersi mantenere aderente con le esigenze di crescita aziendale. Inoltre, e’ essenziale che la piattaforma possa avere degli *standard di sicurezza elevati*, come ben sappiamo, durante i recenti anni si e’ verificata un impennata dei crimini legati alla *Cybersecurity*, con un particolare aumento durante la corrente pandemia da COVID-19, come illustrato [dall’Interpol](#). E’ quindi fondamentale che un’applicazione che gestisce flussi di denaro sia quindi estremamente solida dal punto di vista della sicurezza informatica. Una seconda sezione del progetto, prevede che ogni gruppo si occupi di eseguire dei *penetration test* sul gruppo dall’*ID* successivo. Questo per simulare l’ingaggio di un azienda esterna allo scopo di testare la sicurezza di un prodotto prima di rilasciarlo effettivamente sul mercato, uno step di decisiva importanza che permettera’ ai componenti di ogni gruppo di sperimentare le proprie conoscenze di sicurezza informatica all’interno di una situazione altamente realistica.

Vista la complessita’ del progetto, e’ stato scelto di realizzarlo tramite Team multidisciplinari, con componenti appartenenti ad due corsi afferenti agli indirizzi di *Cloud Specialist* e *ICT Security Specialist*. All’interno di questi due corsi sono presenti le competenze tecniche atte a svolgere il progetto commissionato, coprendo sia l’area di sicurezza e di architettura della rete interna, che quella di utilizzo delle piattaforme cloud che permettono di assicurare la scalabilita’ necessaria all’azienda.

1.2 Il Team

Gli studenti di entrambi i corsi sono stati divisi in sei differenti gruppi, composti da un totale di otto persone, il nostro gruppo, denominato “*Cloud Fiesta*” e’

composto dai seguenti studenti:

- Cafasso Giovanni
- Carrieri Riccardo
- De Lazzari Riccardo
- Dellerà Lorenzo
- Murta Alessio
- Oglietti Riccardo
- Zuccarella Andrea

Suddivisi rispettivamente all'interno dei due corsi come da tabella:

Cloud Specialist	ICT Security Specialist
Cafasso Giovanni	De Lazzari Riccardo
Carrieri Riccardo	Dellerà Lorenzo
Murta Alessio	Oglietti Riccardo
Zuccarella Andrea	

Come consigliato dai docenti, sono stati assegnati alcuni *ruoli* in grado di aiutarci con l'organizzazione delle mansioni e in genere della gestione del progetto, in particolare abbiamo individuato il ruolo di *Team Leader* e di *Co-Team Leader*, essi sono stati rispettivamente assegnati a *Oglietti Riccardo* e *Murta Alessio*. Abbiamo optato per assegnare queste due cariche ripartendole tra i due differenti corsi che compongono il gruppo in maniera da mantenere un buon livello di equità e rappresentanza per entrambe le anime del team.

Capitolo 2

Strumenti tecnico-organizzativi

2.1 GANTT e cronoprogramma

Innanzitutto parlando di strumenti tecnico-organizzativi non e' possibile iniziare senza descrivere il *GANTT*. Strumento principe per l'organizzazione delle tempistiche, si tratta di una tabella a doppia entrata che permette di assegnare alcuni *task* ritenuti fondamentali a un membro e un momento nel quale realizzarlo.

Ecco una lista riassuntiva dei processi e degli *step* fondamentali che abbiamo individuato al fine della realizzazione ottimale del progetto, divisi in base al corso di afferenza dei destinatari:

- 1. Parsing file CSV
 2. Definizione struttura di rete
 3. Deploy infrastruttura
 4. Test di sicurezza
 5. Modifica struttura in base alle falle trovate
 6. Deploy struttura finale
 7. Implemmentazione certificato SSL
 8. Stesura report
 9. Stesura presentazione Powerpoint
- 1. Brainstorming
 2. Test locali nopCommerce
 3. Revisione manuale file CSV
 4. Selezione architettura Cloud
 5. Containerizzazione su distro linux
 6. Installazione locale nopCommerce/ DB su due macchine
 7. Upload su Cloud Provider

8. Calcolo dei prezzi dell'Hosting di tutto il progetto (macchine virtuali, storage, call)
9. Stesura report economico
10. Gestione permessi utenti azure

2.2 Strumenti di comunicazione

Durante il primo incontro uno dei principali punti che e' stato chiarito e' quello della *comunicazione*. E' infatti essenziale che in un gruppo di lavoro sia possibile gestire la comunicazione in maniera piu' efficiente e inclusiva possibile, senza quindi escludere membri o affidarsi a piattaforme troppo lente o non organizzate.

La scelta e' quindi ricaduta sulla piattaforma di messaggistica istantanea *Telegram*, grazie alla puntualita' delle opzioni di gestione di una *chat* di gruppo e' possibile *fissare* messaggi, creare sondaggi e inviare file di grandi dimensioni. Grazie a recenti aggiornamenti e' inoltre possibile effettuare videochiamate e condividere eventualmente il proprio desktop, una feature essenziale nel campo del lavoro collaborativo.

2.3 Organizzazione codice

Data la forte componente di scrittura software presente all'interno del progetto, si e' propenso per l'utilizzo di una piattaforma di sviluppo collaborativo, in maniera da organizzare la stesura del codice nella maniera piu' semplice ed esaustiva possibile. In particolare ci siamo affidati al software *GIT* a opera di *Linus Torvalds*, creando un organizzazione sulla popolare piattaforma di proprieta' *Microsoft*, *GitHub*.

Sulla piattaforma ci siamo quindi premurati di creare immediatamente tre *repository* atti a contenere il lavoro prodotto dal gruppo, in particolare essi sono:

1. `Random_Script`
2. `Report`
3. `Report_Economy`

Il repository numero 1, *Random_Script* e' atto al contenimento di una serie di programmi di piccola entita', come il *parser* che si e' occupato di scaricare le immagini dei prodotti da aggiungere successivamente al database dello store, o il *docker-compose.yml* che servira' per effettuare l'operazione di *deploy* sull'infrastruttura in ambiente di produzione.

Per quanto riguarda *Report*, ossia il numero 2, si tratta dello spazio atto alla creazione del report finale, esso e' stato redatto tramite l'utilizzo del linguaggio *L^AT_EX*, argomento che sara' affrontato in dettaglio in seguito.

Infine, il repository 3, nominato come *Report_Economy*, e' atto ad accogliere i documenti e gli appunti che permetteranno la stesura di un preventivo attendibile dell'implementazione, come richiesto dai requisiti del progetto.

2.4 Strumenti di scrittura

Come accennato durante la precedente sezione, lo strumento principe che e' stato impiegato per la redazione della relazione di progetto e' stato il linguaggio $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$. Si tratta di un linguaggio in grado di produrre un testo correttamente formattato a partire da semicodice, in questo modo viene automatizzata la gestione di alcune importanti caratteristiche del documento finale, come per esempio le immagini, spesso punto di debolezza dei comuni software di videoscrittura.

Capitolo 3

Componenti e architettura

3.1 In generale

3.1.1 Microsoft Azure

Per descrivere l'architettura da noi ideata riteniamo importante descrivere il *provider* al quale abbiamo deciso di appoggiarci. La scelta e' ricaduta sullo strumento "*Microsoft Azure*": si tratta di un servizio di *Cloud Computing* offerto da *Microsoft* che, come il resto dei *provider*, offre servizi di *Infrastructure as a Service*, *Platform as a Service* e *Software as a Service*. In particolare e' stato scelto in quanto aderente alle nostre necessita' di gestione di macchine remote da parte di un gruppo organizzato, inoltre, *Microsoft* offre un bonus gratuito di \$100 da spendere sulla piattaforma per ogni persona che vi si registri come studente. Cio', in concomitanza con i prezzi in linea con il mercato, ci ha permesso di sperimentare senza rischiare di dilapidare denaro.

3.1.2 nopCommerce

Durante la nostra ricerca di una soluzione che ci permettesse di creare uno *store online* ci siamo imbattuti in *nopCommerce*, una tecnologia a opera di *nopSolutions*. Si tratta di una soluzione *libre* e *open source* che permette la creazione di *store online* mantenendo una discreta semplicita' di utilizzo, nonche una facile integrazione in ogni sistema grazie alla possibilita' di essere installato all'interno di un *container Docker*.

Nato nel 2008, sviluppo e supporto non si sono mai interrotti, l'ampia *community* che lo mantiene e lo sviluppa permette inoltre una facile risoluzione di eventuali problemi grazie all'ampia *documentazione* prodotta nel corso degli anni. Questo prodotto abbraccia le moderne tecnologie in ambito di sviluppo web e di sicurezza grazie al massiccio utilizzo di *ASP.NET Core 5* e all'utilizzo come database predefinito di *MySQL* fino alle ultime versioni stabili.

3.2 Architettura di rete

L'architettura di rete scelta e' basata sull'utilizzo di una singola macchina virtuale in *cloud* sulla sopracitata piattaforma *MS Azure*.

La macchina virtuale monta un sistema operativo *GNU/Linux Ubuntu Server 20.04 LTS*, e ha le seguenti caratteristiche:

Informazione	Metrica
Tipologia	Standard_D2s_v3
CPU	2
RAM	8 GB
Disco	30 GB SSD Premium con ridondanza locale
Sicurezza	Standard

Al suo interno sono poi presenti alcuni elementi aggiuntivi, che compongono la vera e propria infrastruttura.

Innanzitutto e' presente *Docker*, si tratta del *runtime* piu' diffuso per *container* diffuso sul mercato. I *container* sono "entita'" che al loro interno contengono un ambiente minimale con tutte le componenti necessarie a un applicativo per svolgere il suo funzionamento, comprese tutte le dipendenze di ognuna delle sue parti. Questa entita' si interfaccia poi con il *Docker Engine*, un software che si occupa di tradurre le richieste del container in chiamate al sistema operativo sottostante, in questo caso una distribuzione di *Ubuntu GNU/Linux*.

Da sottolineare poi la fondamentale presenza di *SSH*. Si tratta dell'implementazione dell'omonimo protocollo di connessione remota per sistemi operativi *UNIX like*. Esso permette di effettuare una connessione remota con una macchina tramite una coppia di credenziali oppure attraverso una chiave *RSA*, criptando il traffico in maniera da mantenere la riservatezza della comunicazione.

Infine, gli ultimi due applicativi che e' opportuno segnalare come parti fondamentali della topografia di rete sono *Uncomplicated FireWall* e *Fail2Ban*. Il primo, come intuibile dal nome, e' un *Firewall* atto a limitare le connessioni non autorizzate verso la macchina per il quale e' configurato. In particolare, questo firewall e' in realta' un *frontend* semplificato per *IP Tables*, probabilmente il piu' utilizzato firewall in ambiente *GNU/Linux*. Anche *Fail2Ban* si occupa di una funzione simile, in quanto la sua funzione principale all'interno dell'architettura proposta e' quella di regolamentare e limitare l'accesso alle connessioni *SSH* per i soggetti non autorizzati.

In figura 3.3 una rappresentazione grafica di quanto illustrato.

3.3 Organizzazione container

Come accennato durante il precedente paragrafo, la nostra architettura e' organizzata tramite l'utilizzo di *container*. Questa metodologia di *deploy* e' stata scelta perche offre numerosi vantaggi rispetto alla piu' "classica" architettura basata sull'utilizzo di macchine virtuali dedicate. In particolare, una macchina e' in grado di gestire diversi *container*, ottimizzando al meglio le risorse *hardware* a disposizione. Inoltre, il parziale isolamento di un *container* rispetto alla macchina *host* rende l'infrastruttura piu' sicura, in quanto compromettere un singolo applicativo non mette a rischio il resto dell'infrastruttura o degli altri

servizi che condividono le medesime risorse. Infine e' importante sottolineare che, grazie all'utilizzo di *Docker Compose*, uno strumento per la definizione e l'esecuzione di applicazioni *Docker* multi-contenitore, possiamo utilizzare un singolo comando per creare e avviare tutti i container definiti nel file `yaml`. Non solo, ci permette di definire e collegare in una rete logica i *container*, che interagiranno tra loro come fossero macchine fisiche. Per implementare la nostra infrastruttura, abbiamo deciso di utilizzare i seguenti *container*:

- nopCommerce
- MySQL

Il primo denominato come *nopCommerce* contiene l'effettiva struttura dello *store online*, compreso di tutte le componenti atte a pubblicare le pagine *web* sulla porta 8080.

Il secondo *container* invece, contiene un database *MySQL* che si occuperà di contenere tutte le informazioni riguardanti i prodotti, i clienti, gli acquisti e molto altro.

Ecco alcuni dei campi presenti all'interno del *database*

1. ProductId
2. ProductType
3. Name
4. FullDescription
5. Vendor

I *container* sono configurati in maniera da utilizzare due porte per la comunicazione: la porta 80 e la porta 3306. Rispettivamente usate per permettere al *container* contenente *nopCommerce* di essere esposto verso internet, e sempre al medesimo di effettuare le comunicazioni con il database *MySQL*.

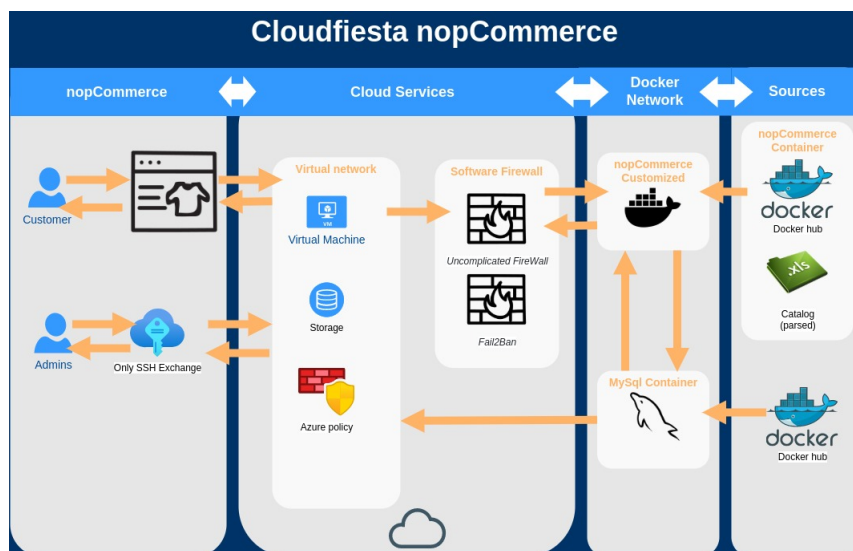


Figura 3.1: Infrastruttura di rete

Capitolo 4

Processo di implementazione

Per gestire al meglio le tempistiche e la messa a terra del progetto abbiamo strutturato un processo diviso in fasi, esse ci hanno permesso di poter controllare lo stato di avanzamento dei lavori e modificare i programmi e il carico di lavoro in maniera coerente. In particolare possiamo individuare tre principali fasi, elencate di seguito;

1. Brainstorming e ricerca
2. Prototipazione e scripting
3. Implementazione locale
4. Implementazione remota
5. Test di sicurezza

Ognuna di queste fasi del lavoro ha occupato un diverso ruolo e ha necessitato sforzi di natura diversa, ecco dunque presentata una breve sintesi di quanto accaduto in ognuna di esse.

4.1 Brainstorming e ricerca

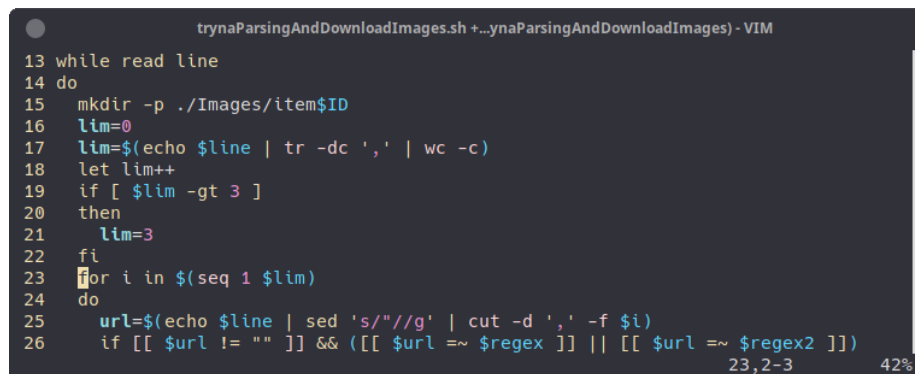
Durante la prima parte del progetto, il gruppo si e' dedicato all'individuazione degli strumenti precedentemente citati atti all'implementazione di quanto richiesto. Questo iniziale sforzo e' stato portato avanti contemporaneamente da tutti i membri del gruppo, mentre i membri afferenti al corso di *Cloud Specialist* si sono occupati di effettuare le necessarie ricerche per quanto riguarda il provider di servizi, i membri di *ICT Security Specialist* si sono occupati di iniziare a definire alcuni strumenti atti alla gestione della rete. Durante questa fase iniziale, sono stati inoltre individuati lo strumento atto alla creazione dello *store online*, la relativa gestione del *database* e tutti gli strumenti di comunicazione, organizzazione e videoscrittura.

4.2 Prototipazione e scripting

La fase successiva si e' rivelata essere molto piu' pratica di quella appena svolta, in quanto il gruppo si e' trovato a dover iniziare a risolvere alcuni problemi pratici, come il *parsing* del file *csv* contenente i dati atti a popolare il *database* e l'utilizzo di *nopCommerce*.

La necessita' di agire sul file originale contenente i dati iniziali per popolare il *database* e' nata dalla scelta operata dal team di sicurezza di mantenere una copia delle immagini localmente alla macchina in maniera da ridurre il perimetro di vulnerabilita' dell'infrastruttura. Senza questo passaggio, il *database* conterrebbe collegamenti a risorse esterne, i quali potrebbero essere sfruttati da attori terzi per ottenere accesso o controllo a parti dell'infrastruttura.

E' quindi stato prodotto uno *script* tramite il linguaggio *bash* che, partendo da una trascrizione in formato testuale del file originale, possa ottenere le immagini relative ai prodotti conservandone il corretto ordine e il riferimento al prodotto. Questo compito e' stato reso piu' difficile da alcuni problemi di formattazione contenuti all'interno del file originale. Come prima azione, lo *script* si occupa di controllare che tutti gli *URL* siano validi e non nulli, dopodiche procede con l'effettivo scaricamento e indicizzazione delle immagini, in figura 4.2 un'estratto dal sopracitato script.



```
trynaParsingAndDownloadImages.sh +...ynaParsingAndDownloadImages) - VIM
13 while read line
14 do
15     mkdir -p ./Images/item$ID
16     lim=0
17     lim=$(echo $line | tr -dc ',' | wc -c)
18     let lim++
19     if [ $lim -gt 3 ]
20     then
21         lim=3
22     fi
23     for i in $(seq 1 $lim)
24     do
25         url=$(echo $line | sed 's/"//g' | cut -d ',' -f $i)
26         if [[ $url != "" ]] && ([[ $url =~ $regex ]] || [[ $url =~ $regex2 ]])
```

Figura 4.1:

Script atto al *download* delle immagini.

Durante questa fase sono state esplorate le varie opzioni per l'implementazione di *nopCommerce*, considerando quale sistema operativo e strategia di *deploy* scegliere tra le diverse disponibili.

4.3 Implementazione locale

La fase logicamente successiva ha quindi previsto l'implementazione per intero dello *store* localmente, in maniera da poter evidenziare eventuali criticita' e difficolta' di messa in produzione. In particolare l'utilizzo del *database* ha inizialmente creato alcune difficolta', al fine di condurre test e sperimentazioni e' stato scelto di usare *MySQL* tramite il gruppo di *software* orientato alla programmazione *WEB XAMPP*. Si tratta di uno *stack* di *software* comunemente

usato all'interno della programmazione *WEB*, l'acronimo simboleggia i *software* conenuti al suo interno, ossia:

X	Cross-Plattform
A	Apache WEB-server
M	MySQL
P	PhP
P	Pearl

In particolare, le difficoltà sono state riscontrate nell'importazione del *database*: *nopCommerce* sfrutta una serie di parametri non presenti all'interno del file originariamente fornitoci, e' quindi stata necessaria un'operazione di modifica per adattare il file iniziale alle esigenze della piattaforma da noi scelta. Prima di giungere a questa soluzione, sono state riscontrate diverse anomalie, per esempio l'impossibilita' di visualizzare i prodotti ottenuti, o di aggiungerli al "carrello" per terminare l'acquisto. Il problema e' stato finalmente risolto tramite l'esportazione della tabella *products* tramite l'interfaccia alla piattaforma per comprenderne al meglio la struttura, e poter di conseguenza strutturare il file *CSV* in maniera consona. Una volta ultimata un implementazione completamente funzionante della piattaforma di *e-commerce* localmente, e' iniziata la sperimentazione remota, descritta in maniera piu' completa di seguito.

4.4 Implementazione remota

Durante la fase di implementazione remota, lo scopo del gruppo e' stato quello di ottenere un'infrastruttura funzionante e completa, compresa di dati e grossolane misure di sicurezza, da poter poi raffinare fino all'ottenimento del risultato finale.

Il processo e' partito dalla creazione di una macchina virtuale con le caratteristiche descritte nella sezione 3.2, il gruppo si e' poi cimentato nella gestione degli accessi alla sopracitata macchina, dapprima tentando un approccio basato sulle organizzazioni e i gruppi offerti nell'ambito della piattaforma *Microsoft Azure*, terminando poi con il piu' intuitivo sistema offerto nativamente dal sistema operativo scelto basato sul protocollo *SSH* descritto nella sezione 3.2. Il primo approccio ha presentato diverse difficoltà legate inizialmente all'assegnazione di tutti i componenti del gruppo a una singola entita' organizzativa, seguite poi da difficoltà di assegnazione dei necessari permessi per poter ottenere visibilita' sulla macchina che avrebbe ospitato l'infrastruttura e infine dell'assegnazione di chiavi di accesso univoche a tutti i membri. E' stato quindi scelto di lasciare la creazione della macchina e di un primo account con privilegi amministrativi a un membro, il quale ha poi condiviso le credenziali necessarie alla connessione remota tramite comunicazione criptata. Da questo primo account amministrativo e' stato poi possibile impostare profili personali per ognuno dei membri del gruppo, dotando ognuno dei minimi privilegi necessari allo svolgimento della sua funzione secondo il principio "*Least Privilege*". In figura 4.4 sono illustrati i gruppi di appartenenza di ciascun utente.

```

riky@prod2:/home$ for i in $(ls); do groups $i; done
cafaxx : cafaxx
griselbran : griselbran sudo
herfiodena : herfiodena adm dialout sudo audio dip video plugdev netdev lxd
lugeee : lugeee
nopCommerce : nopCommerce docker
riky : riky sudo

```

Figura 4.2:

Gruppi di appartenenza per ogni utente

Ognuno dei profili utente creati e' dotato di una coppia di chiavi *RSA* generata tramite il comando `ssh-keygen -b 4048`, che, in congiunzione con le coppie di chiavi delle macchine dei rispettivi membri del gruppo ha permesso di impostare un accesso ai profili utenti tramite chiave univoca. Grazie a questi scambi di chiavi, e' stato successivamente possibile disabilitare l'accesso via *password* a tutti gli utenti, rinforzando notevolmente la sicurezza della macchina e dell'infrastruttura. E' poi stato creato un utente denominato *nopCommerce* abilitato ad utilizzare solamente i comandi indispensabili alla gestione dei container contenenti la piattaforma e il relativo *database*. Questo account e' inoltre privo di chiavi e inibito dallo stabilire connessioni *SSH* in entrata o in uscita tramite l'utilizzo di *policy*, in maniera da renderlo accessibile solamente da un utente all'interno della macchina tramite il comando `su`, protetto da una *password* notevolmente robusta. Una volta terminata la messa in rete della piattaforma di *e-commerce* l'utente *nopCommerce* e' stato ulteriormente limitato tramite l'ipostazione della *shell* `/sbin/nologin` come interfaccia predefinita dell'utente, in questa maniera, anche se in possesso delle credenziali di accesso di un utente, e' comunque impossibile effettuare il *login* all'*account* *nopCommerce* in quanto esso non ha un effettiva *shell* interattiva con la quale interfacciarsi. Inoltre, sia tramite l'interfaccia di *Microsoft Azure* sia tramite il *firewall* implementato, sono state definite alcune *policy* atte a limitare l'accesso all'infrastruttura da parte di soggetti non autorizzati. In particolare le *policy* sono state suddivise principalmente in due categorie: quelle legate alle connessioni *SSH*, gestite da *Fail2Ban* e *UFW*; e quelle gestite dalla piattaforma *Azure*, atte principalmente all'amministrazione delle connessioni verso la macchina virtuale e le sue porte. Grazie alla fase di produzione in locale, e' stato possibile definire una serie di passaggi per rendere piu' agile il *deploy* dell'infrastruttura, partendo dall'immagine ufficiale di *nopCommerce* disponibile su *Docker Hub*, e' stato possibile costruirne una personalizzata, contenente direttamente le immagini afferenti ai prodotti presenti all'interno del catalogo. La nuova immagine e' custodita in repository privato sulla piattaforma *Docker Hub*, funzionale solo al nostro progetto. Bastera' dunque lanciare il comando `docker-compose up`, per ottenere in pochi minuti l'intera infrastruttura, attiva e pronta per la produzione. Questo espediente, insieme a due file di configurazione in formato *excel*, da la possibilita' di mettere in produzione la piattaforma *e-commerce* in pochi minuti, in maniera totalmente indipendente dal *provider* di servizi *cloud* utilizzato. L'ultima fase della "sperimentazione" sulla macchina di prova si e' concentrata sull'implementazione del metodo di pagamento. Risultato che e' stato facilmente ottenuto grazie all'integrazione della piattaforma con diversi metodi di pagamento, in particolare, e' stato scelto di utilizzare *PayPal*. Si tratta di un metodo di pagamento sicuro che offre canoni bassi e una relativa

facilita' di utilizzo. Per effettuare l'implementazione, e' stato necessario configurare il metodo tramite l'interfaccia grafica dello *store* e collegarlo con un *account* precedentemente creato.

In ultimo, grazie ai test e alle prove effettuate sulla macchina di prova, e' stato possibile creare una nuova macchina denominata **prod** dotata di tutte le misure di sicurezza sopracitate nella quale attivare la piattaforma in totale tranquillita', evitando inoltre di "sporcare" la macchina con esperimenti di sorta, il risultato si e' rivelato molto semplice e pulito, ideale per essere mantenuto al meglio. Come illustrato nell'immagine 3.3, l'accesso da parte degli amministratori di sistema e' stato mantenuto tramite autenticazione chiave *SSH* per rendere piu' semplice possibile la manutenzione, l'aggiornamento e l'eventuale aggiunta di servizi.

4.5 Test di sicurezza

Come ultima fase prima della pubblicazione, e' stato scelto di operare alcuni "sommari" test di sicurezza, al fine di controllare la solidita' dell'infrastruttura sviluppata. Per rendere la procedura utile al fine di migliorare la sicurezza dell'infrastruttura, e' stato scelto di adoperare una modalita' di azione denominata come *Black Box*. In questa modalita', l'attaccante, anche avendo accesso all'infrastruttura agisce come se non avesse nessun informazione, partendo quindi dall'analisi della superficie esposta.

4.5.1 Analisi del perimetro

Come prima azione, e' essenziale la scoperta di quante piu' informazioni possibili in merito al bersaglio, il primo passo in questa direzione e' sicuramente quello dell'utilizzo dello strumento *nmap* al fine di ottenere informazioni sui servizi attivi sul bersaglio ed eventualmente esporne vulnerabilita'. *Nmap*, abbreviazione di *Network Mapper* e' un programma di *network discovery* e *security auditing*, il suo scopo e' analizzare un dato indirizzo per scoprire informazioni sulle porte aperte, sui servizi attivi e su possibili vulnerabilita' relative ad essi.

La scansione tramite il sopracitato programma rivela che la macchina analizzata ha le seguenti porte attive:

- 22/tcp: *SSH*
- 53/tcp: *domain*
- 80/tcp: *HTTP*

Nonostante l'approccio *Black Box*, non sapendo quindi che tipo di autenticazione viene permessa da parte del server per quanto riguarda il protocollo *SSH*, tentare di forzare un accesso tramite un attacco *Brute Force* risulterebbe essere solamente una perdita di tempo. Anche assunto un nome semplice per un *account* utente quale "nopCommerce", desumibile dalle informazioni ottenute riguardanti il servizio sulla porta 80, un attacco a forza bruta basato su una *wordlist* di *password* diffuse avrebbe comunque delle tempistiche molto lunghe, e si rivelerebbe comunque probabilmente infruttuoso. Viene quindi scelto di percorrere la strada dell'analisi della piattaforma pubblicata sulla porta 80. Tramite

il programma di scansione *gobuster* e' possibile ottenere informazioni aggiuntive su cio' che e' pubblicato sulle porte *http* e *https*, il *software* si occupa di tentare di contattare una serie di pagine collegate al dominio di secondo livello in successione, prendendo i nomi da una lista di pagine comuni. Filtrando poi le pagine che restituiscono una risposta *http* positiva, ossia i seguenti codici:

- 200: OK
- 301: Moved Permanently
- 203: Found

Capitolo 5

Penetration testing

5.1 Le premesse

5.2 Information gathering