README

Premessa

Per la realizzazione dell’architettura sono stati utilizzati 2 servizi di Hashicorp per comunicare con il cloud pubblico di AWS, quali Terraform e Packer: uno usato per il deploy dell’infrastruttura, l’altro per la costruzione dell’immagine con Wordpress e le sue dipendenze. L’utilizzo di Terraform è stato fatto senza prendere in considerazione l’utilizzo di moduli già presenti online, questa scelta a mie spese non mi ha permesso di deployare tutti i servizi che avrei voluto ma solo lo stretto necessario per il funzionamento, d’altro canto però ho avuto la possibilità di aumentare la padronanza nell’utilizzo del linguaggio per un use case sfidante e motivante.

L’architettura Utilizzata

Per Il raggiungimento dei punti richiesti dall’architettura ho utilizzato le best practice di AWS per architetture scalabili e affidabili, deployando i servizi utilizzati duplicati su due Avability Zone.

I servizi utilizzati sono:

* VPC (10.0.0.0/16)
* Subnet (numero = 4) -> 2 pubbliche e 2 private su 2 az differenti
* Internet gateway
* Route table(numero = 3) -> 2 private e 1 pubblica
* Istanza Nat Ec2 (numero = 2) -> uno per ogni subnet pubblica
* Bastion host (numero = 2) - > uno per ogni subnet pubblica
* Pubblic Load Balance
* Target Group - > collegato con ASG
* ASG - > su due subnet private
* Cloudwatch -> per triggherare la ASG policy
* Configurazione di lancio
* Security Group (numero = 4)
* RDS -> cluster aurora, non testato

Il progetto

Il progetto è suddiviso in 2 cartelle: nella cartella “Packer” sono presenti il file “**wordpress.pkr.hcl”** che effettua la costruzione dell’immagine, il file “**provisioning-wordpress.sh**” che esegue un codice sh per scaricare wordpress e dipendenze (nel file c’è una variabile BASH che salva la versione di wordpress da scaricare). L’altra cartella “Terraform” contiene i file .tf per deployare l’architettura e un file che contiene le variabili utilizzate all’interno del progetto

Per prima cosa per utilizzare lo script bisogna:

1. Creare un utente AIM
2. Dare dei privilegi di accesso a quest’ultimo
3. Creare delle SDK associate

Create le SDK è possibile sostituirle all’interno del file “variable.tf” di Terraform nelle variabili: access\_key e secret\_key. Lo stesso deve essere fatto all’interno del file **“wordpress.pkr.hcl”** nella sezione delle variabili.

Creazione dell’immagine Wordpress

Il primo step è la costruzione dell’immagine con packer. Lo script come prima cosa crea l’immagine e la salva come snapshot su EBS di AWS, da quello viene poi creata L’AMI utilizzabile per la creazione delle istanze EC2. I comandi da eseguire sono: “**packer validate .**” per il controllo sintattico del codice, “**packer fmt .**“ per dare una formattazione tabulare al codice e “**packer build .**” per la costruzione dell’immagine

Codice Terraform

il codice è diviso in otto file, ognuno di essi responsabile di una parte dell’architettura: rete, istanze, bilanciatore ecc, uno di questi file è utilizzato per le variabili che possono essere modificate in base all’esigenze. All’interno del file network.tf vengono definiti tutti i servizi di rete utilizzati e i loro collegamenti, inoltre viene definito un servizio di Nat offerto da una AMI della comunity di AWS. Attenzione una volta deploiato il Nat bisogna a spuntare l’opzione “arresta” sotto la voce “Controllo dell'origine/della destinazione” (**per un ambiente di produzione è fortemente consigliabile il Nat Gateway offerto da AWS, in quanto ha prestazioni molto più alte**).

Il file **bastion-host.tf** si occupa di deployare le istanze ec2 che si occupane della connessione ssh con le macchine hostate nelle sottoreti private. Il file: genera le chiavi ssh, le associa con le istanze create e le scarica in locale nella nostra macchina; in fine si prende la chiave ssh associata alle istanze private e la salva sul bastion host(remote exec).

Nel file di “**Auto\_Scaling\_group.tf”** vengono definite: il modello di lancio delle nuove istanze (In questo caso viene presa la AMI con wordpress costruita con packer), il gruppo di auto scaling (definito sulle 2 subnet private), la policy di scalabilità collegata con Cloudwatch e i vari collegamenti con gli oggetti.

nel **Load\_Balancer.tf** sono definiti: il bilanciatore applicativo definito su 2 subnet private, il gruppo di target e il listener http.

Tutti i network securty group utilizzati nell’architettura si trovano nel file **“security\_group.tf”** per motivi di sicurezza alcune porte sono state aperte in ingresso ed uscita solamente ad alcuni gruppi

In fine il “**Configuration.tf”** è il file usato per specificare il provider utilizzato per il deploy, versione di terraform e per la configurazione del backend per il tfstate.

Per eseguire il codice Terraform mettersi nel path della cartella ed eseguire i comandi

* terraform init: comando usato per inizializzare la cartella del progetto e scaricare i pacchetti per il provider utilizzato
* terraform validate: comando usato per controllare la validità sintattica del codice
* terraform plan: per visualizzare le infrastrutture che verranno deployate, modificare o distrutte
* terraform apply: per eseguire il deploy delle infrastrutture presentate dal terraform plan
* terraform destroy: da usare per distruggere tutti i servizi deployati sul cloud, presenti sul tfstate

Conclusione e migliorie

In conclusione, per il completo raggiungimento dei punti richiesti come: la sicurezza e la velocità dell’architettura, sono stati pensati i servizi Memcached per quanto riguarda la velocità per le richiedere al database, Cloudfront con backet S3 per esporre i media file su CDN per renderli più veloci da esporre, in basa alla locazione dell’utente. Per quanto riguarda la sicurezza, il certificato SSL da mettere davanti Bilanciatore, questo ovviamente implicherebbe l’apertura delle porte 443 per i security group e anche i target group vanno modificati sulla nuova porta.

