Elies.jebri@gmail.com

Simulation d'une application d'architecture à trois niveaux en réseau avec Vagrant

Vagrant est un excellent outil pour aider à simuler des systèmes dans des réseaux isolés, nous permettant de simuler facilement les architectures trouvées en production. L'idée derrière les multiples niveaux est de séparer la logique et l'exécution des différents éléments de l'application, et de ne pas tout centraliser en un seul endroit. Un modèle courant consiste à obtenir une première couche qui reçoit les demandes courantes des utilisateurs, une deuxième couche qui fait le travail d'application et une troisième couche qui stocke et récupère les données, généralement dans une base de données.

Dans cette simulation, nous aurons les trois niveaux traditionnels, chacun exécutant des machines virtuelles CentOS 7 sur son propre réseau isolé:

- **Front** : proxy inverse NGINX
- **Application** : une application Node.js s'exécutant sur deux nœuds
- **Base de données** : Redis

Nom de la machine virtuelle	front_lan IP	app_lan IP	db_lan IP
front-1	10.10.0.11 / 24	10.20.0.101 / 24	N / A
app-1	N / A	20.0.11 / 24	10.30.0.101 / 24
app-2	N / A	20.0.12 / 24	10.30.0.102 / 24
db-1	N / A	N / A	10.30.0.11 / 24

Vous accéderez au proxy inverse (NGINX), qui seul peut contacter le serveur d'applications (Node.js), qui est le seul à pouvoir se connecter à la base de données.

Se préparer

Pour parcourir cette recette, vous aurez besoin des éléments suivants:

- Une installation Vagrant fonctionnelle
- Une installation VirtualBox fonctionnelle
- Une connexion Internet

Comment faire…

Suivre ces étapes pour simuler une application d'architecture à trois niveaux en réseau avec Vagrant.

Niveau 3 - la base de données

La base de données vit dans un réseau privé db_lan avec l'IP 10.30.0.11/24.

Cette application utilisera une installation Redis simple. L'installation et la configuration de Redis dépassent le cadre de ce livre, nous allons donc le garder aussi simple que possible (installez-le, configurez-le pour écouter sur le port LAN au lieu de 127.0.0.1 et démarrez-le):

```
  config.vm.define "db-1" do |config|
    config.vm.box = "centos/7"
    config.vm.hostname = "db-1"
    config.vm.network "private_network", ip: "10.30.0.11", virtualbox__intnet:
"db_lan"
    config.vm.provision :shell, :inline => "sudo yum install -q -y epel-release"
    config.vm.provision :shell, :inline => "sudo yum install -q -y redis"
    config.vm.provision :shell, :inline => "sudo sed -i 's/bind 127.0.0.1/bind
127.0.0.1 10.30.0.11/' /etc/redis.conf"
    config.vm.provision :shell, :inline => "sudo systemctl enable redis"
    config.vm.provision :shell, :inline => "sudo systemctl start redis"
  end
```

Niveau 2: les serveurs d'applications

Ce niveau est l'endroit où notre application réside, soutenue par un serveur d'application (Web). L'application peut se connecter au niveau base de données et sera disponible pour l'utilisateur final via des serveurs proxy de niveau 1. C'est généralement là que toute la logique est effectuée (par l'application).

L'application Node.js

Cette volonté être simulé avec le code Node.js le plus simple que j'ai pu produire pour démontrer l'utilisation, en affichant le nom d'hôte du serveur (le nom du fichier est `app.js`).

Tout d'abord, il crée une connexion au serveur Redis sur le `db_lan` réseau:

```
#!/usr/bin/env node
var os = require("os");
var redis = require('redis');
var client = redis.createClient(6379, '10.30.0.11');
client.on('connect', function() {
    console.log('connected to redis on '+os.hostname()+' 10.30.0.11:6379');
});
```

Ensuite, si cela se passe bien, il crée un serveur HTTP à l'écoute `:8080`, affichant le nom d'hôte du serveur:

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Running on '+os.hostname()+'\n');
}).listen(8080);
console.log('HTTP server listening on :8080');
```

Démarrez l'application, avec le fichier de service `systemd`:

```
[Unit]
Description=Node App
After=network.target

[Service]
ExecStart=/srv/nodeapp/app.js
Restart=always
User=vagrant
Group=vagrant
Environment=PATH=/usr/bin
Environment=NODE_ENV=production
WorkingDirectory=/srv/nodeapp
[Install]
WantedBy=multi-user.target
```

Répétons le déploiement d'un certain nombre de serveurs d'applications (dans ce cas: deux) pour servir l'application. Déploiement d'applications Node.js et déploiement d'unités systemd. En production, cela se ferait probablement via un outil de gestion de configuration tel que Chef ou Ansible et peut-être couplé avec un outil de déploiement approprié:

```
# Tier 2: a scalable number of application servers
vm_app_num = 2
  (1..vm_app_num).each do |n|
    app_lan_ip = "10.20.0.#{n+10}"
    db_lan_ip = "10.30.0.#{n+100}"
    config.vm.define "app-#{n}" do |config|
      config.vm.box = "centos/7"
      config.vm.hostname = "app-#{n}"
      config.vm.network "private_network", ip: app_lan_ip, virtualbox__intnet:
"app_lan"
      config.vm.network "private_network", ip: db_lan_ip, virtualbox__intnet:
"db_lan"
      config.vm.provision :shell, :inline => "sudo yum install -q -y epel-
release"
      config.vm.provision :shell, :inline => "sudo yum install -q -y nodejs npm"
      config.vm.provision :shell, :inline => "sudo mkdir /srv/nodeapp"
      config.vm.provision :shell, :inline => "sudo cp /vagrant/app.js
/src/nodeapp"
      config.vm.provision :shell, :inline => "sudo chown -R vagrant.vagrant
/srv/"
      config.vm.provision :shell, :inline => "sudo chmod +x /srv/nodeapp/app.js"
      config.vm.provision :shell, :inline => "cd /srv/nodeapp; npm install redis"
      config.vm.provision :shell, :inline => "sudo cp /vagrant/nodeapp.service
/etc/systemd/system"
      config.vm.provision :shell, :inline => "sudo systemctl daemon-reload"
      config.vm.provision :shell, :inline => "sudo systemctl start nodeapp"
    end
  end
```

Niveau 1: le proxy inverse NGINX

Le niveau 1 est représenté ici par une configuration de proxy inverse NGINX sur CentOS 7.

```
events {
  worker_connections 1024;
}
http {
  upstream app {
    server 10.20.0.11:8080 max_fails=1 fail_timeout=1s;
    server 10.20.0.12:8080 max_fails=1 fail_timeout=1s;
  }
  server {
    listen 80;
    server_name  _;
    location / {
      proxy_set_header   X-Real-IP $remote_addr;
      proxy_set_header   Host      $http_host;
      proxy_pass         http://app;
    }
  }
}
```

Maintenant, allons créer la machine virtuelle proxy inverse qui servira `http://localhost:8080` via le pool de serveurs d'applications. Cette machine virtuelle écoute 10.10.0.11/24 sur son propre LAN (`front_lan`) et sur `10.20.0.101`/ `24` sur le LAN des serveurs d'applications (`app_lan`):

```
  # Tier 1: an NGINX reverse proxy VM, available on http://localhost:8080
  config.vm.define "front-1" do |config|
    config.vm.box = "centos/7"
    config.vm.hostname = "front-1"
    config.vm.network "private_network", ip: "10.10.0.11", virtualbox__intnet:
"front_lan"
    config.vm.network "private_network", ip: "10.20.0.101", virtualbox__intnet:
"app_lan"
    config.vm.network "forwarded_port", guest: 80, host: 8080
    config.vm.provision :shell, :inline => "sudo yum install -q -y epel-release"
    config.vm.provision :shell, :inline => "sudo yum install -q -y nginx"
    config.vm.provision :shell, :inline => "sudo cp /vagrant/nginx.conf
/etc/nginx/nginx.conf"
    config.vm.provision :shell, :inline => "sudo systemctl enable nginx"
    config.vm.provision :shell, :inline => "sudo systemctl start nginx"
  end
```

Démarrez ceci (`vagrant up`) et accédez à `http://localhost:8080`, où l'application affiche le nom d'hôte du serveur d'applications afin que vous puissiez confirmer que l'équilibrage de charge sur les réseaux fonctionne (tandis que les serveurs d'applications peuvent communiquer avec le backend Redis).