Devops_everywhere

💡 Tramite un provisioner a vostra scelta (shell scripting, Puppet, Ansible, Chef, ecc...) configurate una macchina Linux (nessuna preferenza sulla distribuzione).

Sarete liberi di decidere cosa far fare alla macchina. L'importante è che sia portabile. Colui che analizzerà il vostro progetto dovrà eseguire 'vagrant up' e utilizzarla.

Il progetto deve veramente essere di base. Date spazio alla fantasia.

👉🏻 **How to run?**

- Vagrant up
- localhost:8080

VAGRANT PROJECT - DYNAMIC WEBPAGE WITH GITHUB REPO STATS

`API_URL = https://api.github.com/repos/<repoToMonitor>`

Per prima cosa creiamo la repo del progetto e spostiamoci dentro per inizializzare Vagrant con un Vagrantfile e una ./vagrant directory

```
•~» mkdir devops_everywhere
•~» cd devops_everywhere
•~» vagrant init
```

Verrà così generato un Vagrant file che andremo a personalizzare per configurare la VM e i servizi da installare.

- Specifico una box (immagine preconfigurata che contiene un'installazione base di Ubuntu 18)
  - `config.vm.box = "ubuntu/bionic64"`
- Attivo il forward delle porte per rendere raggiungibile la pagina web dall'host
  - `config.vm.network “forwarded_port”, guest: 80, host: 8080`
- Script inline che fa diverse cose:
  1. installa nginx, curl e jq (per elaborare JSON)
  2. crea una directory per l'applicazione web
  3. copia lo script Bash update_repo.sh esterno nella VM e lo rende eseguibile
  4. Esegue lo script per aggiornare la pagina web la prima volta
  5. Configura un cron job con permessi di root che esegue automaticamente lo script ogni 5 minuti
  6. Crea un collegamento simbolico tra il file HTML generato dallo script che contiene i dati aggiornati e il file HTML che nginx usa di default (`/var/www/html/index.html`)

update_repo.sh

```
# Repo GitHub da monitorare (formato: utente/repo)
REPO="Martybb01/formazione_sou"

# URL dell'API GitHub per la repo specificata
API_URL="https://api.github.com/repos/$REPO"

# Funzione per generare la pagina HTML con i dati del repository
# Rendo le variabili locali, quindi visibili solo dentro la funz
generate_html() {
  local name="$1"
  local description="$2"
  local stars="$3"
  local forks="$4"
```

```
  local open_issues="$5"
  local last_commit="$6"

    # Il contenuto dell'heredoc è reindirizzato al file index.ht
  cat <<HTML > /var/www/html/github-monitor/index.html
<!DOCTYPE html>
<html>
<head>
    <title>GitHub Repo Monitor</title>
</head>
<body>
    <h1>GitHub Repo Monitor: $name</h1>
    <p><strong>Description:</strong> $description</p>
    <p><strong>Stars:</strong> $stars</p>
    <p><strong>Forks:</strong> $forks</p>
    <p><strong>Open Issues:</strong> $open_issues</p>
    <p><strong>Last Commit:</strong> $last_commit</p>
</body>
</html>
HTML
}

# Richiedo i dati della repo dall'API GitHub.
# response conterrà la risposta in formato JSON
response=$(curl -s "$API_URL")

# Estraggo i dati necessari usando jq
# l'opzione -r dice.a jq di restituire l'output senza ""
name=$(echo "$response" | jq -r '.name')
description=$(echo "$response" | jq -r '.description')
stars=$(echo "$response" | jq -r '.stargazers_count')
forks=$(echo "$response" | jq -r '.forks_count')
open_issues=$(echo "$response" | jq -r '.open_issues_count')
last_commit=$(echo "$response" | jq -r '.updated_at')
```

```
# Genero la pagina HTML con i dati estratti
generate_html "$name" "$description" "$stars" "$forks" "$open_is
```