Technická dokumentácia

Meno: Ján Nemčík

Predmet: APS

Ak. rok: 2019/2020

Obsah

- 1. Zadanie projektu
- 2. Návrh
- 3. Implementácia
- 4. Testovanie
- 5. Používateľská príručka

1. Zadanie projektu

Zadaním tohto projektu bolo otestovanie konkrétneho scenáru na rôznych typoch virtualizácií a porovnať výsledky medzi jednotlivými typmi. Pre tento účel som si ako scenár zadefinoval build linux kernel-u. V tomto projekte som použil definíciu pre zariadenie Odroid X-2. Pôvodným zámerom bolo buildenie kernelu pre debian, čo sa ale ukázalo pri softvérovej virtualizácii ako nevhodný scenár, nakoľko trval rádovo dni. Taktiež som mal v úmysle buildovať kernel na nasledujúcich typoch virtualizácie:

- hardvérová
- softvérová
- paravirtualizácie
- kontajnerizácia

Z týchto typov som ale vylúčil paravirtualizáciu, nakoľko by bolo potrebné zakomponovať softvér, ktorý paravirtualizáciu podporuje do grub loaderu môjho počítača, čo som považoval za príliš nebezpečné.

2. Návrh

Scripty som sa snažil navrhnút tak, aby boli čo najviac flexibilné a zároveň čo najjednoduchšie. Z tohto dôvodu som sa rozhodol použiť *Shell*, nakoľko je použiteľný cross viaceré linux distribúcie. Cieľom bolo taktiež nutnosť inštálacie ďalších nadbytočných balíčkov a programov, preto som defaultne použil KVM na provisioning virtuálnych strojov, ktorý je natívne súčasťou kernelu. Na účely kontajnerizácie použijem Docker.

Získavanie výsledkov

Nakoľko scripty budú bežať v rôznych prostrediach, čo znamená, že budú spúšťané priamo na konkrétnych virtuálnych strojoch, poprípade ako docker container-y, tak som vytvoril jednoduchú aplikáciu na verejne prístupnom serveri, kde sú posielané výsledky zo všetkých vykonaných buildov kernelu. Viac info v sekcii používateľskej príručky.

3. Implementácia

Implementácia prebiehala pre dva prípady:

- virtuálne stroje
- docker container-y

Všetky scripty, spoločne so spoločnými, sú k dispozícii na tomto Github linku.

3.1 Virtuálne stroje

Ako som už spomenul v časti návrhu, na provisioning používam KVM. Ako operačný systém som si zvolil Debian 10 v 32-bit architektúre, keďže celková veľkosť je menšia ako pri Ubuntu a na účely buildenia kernelu bohato postačuje. Scripty sú ale samozrejme vykonateľné aj v Ubuntu prostredí. Script najskôr stiahne kernel vo verzii 4.19.80 pomocou príkaze *wget*, ktorý následne rozbalí pomocou *unxz* a *tar*. Potom skopíruje príslušný config súbor do priečinku rozbalených kernel modulov. Následne spustím script vykonaním príkazu *time* v rošírenej verzii. Následne po vykonaní buildu sa získané výsledky odošlú na server pomocou príkazu *curl*, kde sú následne spracované a uložene v databáze.

3.1.1 Softvérová virtualizácia

V prípade, že si želáte spustiť build kernelu v prostredí softvérovej virtualizácie, je potrebné vypnúť podporu virtualizácie priamo v BIOS-e a nainštalovať virtuálny stroj, nakoľko jedna inštancie nie je schopná byť hardvérovým a zároveň softvérovým virtuálnym strojom.

3.2 Docker containers

V prípade docker kontajnerizácie, som vytvoril image, ktorý už obsahuje predpripravené kernel module spoločne s potrebnou konfiguráciou. Pri spustení kontajneru je build kernelu automaticky spustený. Predpis Dockerfile-u je uvedený nižšie. Základom je taktiež image debianu obohatený o nevyhnutné balíky pre build kernelu. Vybuildovaný docker image sa nachádza na Docker Hub-e.

```
FROM debian

SHELL [ "/bin/bash", "-c" ]

RUN apt-get update && apt-get install -y \
build-essential \
libncurses-dev \
bison \
flex \
libssl-dev \
libelf-dev \
bc \
wget \
time \
curl

RUN wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.19.80.tar.xz
```

```
RUN unxz linux-4.19.80.tar.xz

RUN tar xf linux-4.19.80.tar

COPY ./.config linux-4.19.80/

COPY build-kernel_docker.sh .

ENTRYPOINT [ "./build-kernel_docker.sh"]

CMD ["/bin/exit"]
```

4. Testovanie

Build kernelu som testoval prevažne ja, kde som testoval v prostredí hardvérovej aj softvérovej virtualizácie, taktiež na mojom osobnom počítači a samozrejme aj v prostredí dockeru. skúšal som rôzne varianty počtu jadier pred build a aj alokovanej pamäte pre virtuálny stroj, resp. pre docker container. Moje riešenie otestoval Matúš Vlček v prostredí dockeru a na hardvérovom virtuálnom stroji, ktorý naprovisionoval pomocou VirtualBox-u a ako OS použil Debian.

5. Používateľská príručka

Tento projekt pozostáva z nasledujúcih scriptov:

- build-kernel.sh
- build-docker.sh
- run-docker.sh
- get-result.sh
- virt-install.sh

Každý z vyššie vymenovaných sciptov, okrem *build-docker*, obsahuje svoj vlastný *help*, a preto budú v nasledujúcich podsekciách iba zhrnuté ich úlohy, *help* je možné vykonať nasledovne: <*script_name*> *-h*|*--help*. **V prípade, že používateľ spúšťa build na virtuálnom stroji, musí zabezpečiť aby sa súbory *build-kernel.sh* a *.config* nachádzali v tom istom priečinku!!**

build-kernel.sh

Tento script nainštaluje potrebne dependencies pre build kernelu a pre spracovanie výsledkov. Súčasťou scriptu je taktiež stiahnutie a rozbalenie samotného kernelu. Po týchto krokoch sa spustí samotný build pomocou príkazu make. Po úspešnom vykonaní buildu sa odstránia stiahnuté súbory a výsledky sa odošlú na server. Používateľ má taktiež možnosť po vykonaní buildu odstrániť všetky doinštalované dependencies (viac v *help*-e scriptu). Script taktiež umožňuje definovať počet jadier, na ktorých build prebiehať (viac v *help*-e scriptu). **Script je potrebné spúšťať ako sudo user, poprípade s userom, ktorý ma práva inštalovať dependencies ako root!** Predpokladom pre úspešné odoslanie výsledkov je pripojenie k internetu.

Inštalované dependencies:

build-essential
libncurses-dev

bison
flex
libssl-dev
libelf-dev
bc
time
curl
jq
wget

build-docker.sh

Tento script builduje docker image z Dockerfile v aktuálnom adresári a taguje ho ako aps:latest. Využiteľný v prípade, že nie je dostupný image z Docker hub-u, je možné po zbuildovaní spustiť container priamo lokálne pomocou príkazu docker run alebo spustením scriptu run-docker.sh.

run-docker.sh

Tento script zabezpečuje spestenie docker containera podľa predpripravenej definície, ktorá už obsahuje všetky potrebné dependencie aj samotný priečinok s kernel modulmi. Pri spustení containera sa automacitky spustí build kernelu. Po ukončení sa výsledky odošlú na server a container sa zastaví a vymaže. Script obsahuje možnosť -r --results s hodnotami *auto* alebo *load*, kde *auto* znamená, že výsledky sa automaticky zobrazia pre posledný vykonaný build v docker containeri. Možnosť *load* hovorí o tom, že používateľ chce výsledky zobraziť manuálne, viac informácií o možnostiach scriptu na sa nachádzajú v *help*-e.

get-result.sh

Tento script slúži na získanie výsledkov buildov kernelu v rámci projektu. Ponúka rôzne možnosti filtrovania a formátovania výstupu. Všetky prípady použitia sú popísané v *help*-e scriptu. **Dependencie potrebné pre získanie výsledkov boli nainštalované v rámci scriptu build-kernel.sh a môžu byť donštalované vykonaním príkazu ./get-result.sh -i, ktorý musí byť spustený ako sudo príkaz.**

Example result with description:

```
{
  "id": 25, // PK
  "time": 797.13, // Build execution time in seconds
  "numofcores": 4, // Number of cores used by build
  "cpuname": "Intel(R) Core(TM) i5-5300U CPU @ 2.30GHz", // Name of CPU on
which was the build executed
  "memusage": 991284, // Memory usage of process in KB
  "cpuusage": 76, // Usage of CPU in %
  "arch": "x86_64", // Architecture of virtual machine
  "timestamp": "2019-11-13T22:32:04.891Z", // Timestamp of insertion
  "hostname": "04bf9c041262", // Name of the host, where the build was
executed
  "username": "docker" // Name of user, who executed the script
}
```

virt-install.sh

Tento script zabezpečí inštaláciu KVM virtuálneho stroja s Debian 10 OS. Všetky možnosti použitia sú popísané v *help*-e scriptu.