Lucru laborator № 7

Tema lucrarilor:

Crearea containerilor. Virtualizare ţn Docker.

Partea Teoretica:

Containerizarea este o tehnică de virtualizare în care nucleul sistemului de operare menține mai multe instanțe izolate de spațiu utilizator în loc de una singură. Aceste instanțe (denumite în mod obișnuit containere sau zone) sunt identice cu o singură instanță a sistemului de operare în ceea ce privește procesele care rulează în ele. Pentru sistemele bazate pe Unix, această tehnologie este similară unei implementări îmbunătățite a mecanismului chroot (operațiunea de schimbare a directorului rădăcină în sistemele de operare de tip Unix). Kernelul asigură izolarea completă a containerelor, astfel încât programele din containere diferite nu se pot afecta reciproc. Spre deosebire de virtualizarea hardware, care emulează mediul hardware și poate rula o gamă largă de sisteme de operare invitate, un container poate rula doar o instanță a unui sistem de operare cu același nucleu ca și sistemul de operare gazdă (toate containerele de pe un nod utilizează un nucleu comun). Containerizarea nu implică un consum suplimentar de resurse pentru emularea hardware-ului virtual și rularea unei instanțe de sistem de operare complet, ceea ce este tipic pentru virtualizarea hardware-ului.

Există implementări axate pe crearea de instanțe aproape complete de sisteme de operare (Solaris Containers, containere Virtuozzo, OpenVZ) și variante care se concentrează pe izolarea serviciilor individuale cu un mediu de operare minim (jail, Docker). Docker este un software pentru automatizarea implementării și gestionării aplicațiilor în medii containerizate, un containerizator de aplicații. Acesta permite "împachetarea" unei aplicații cu tot mediul și dependențele sale într-un container care poate fi implementat pe orice sistem Linux cu suport pentru grupuri de control în kernel și oferă, de asemenea, un set de comenzi pentru gestionarea acestor containere. Software-ul rulează într-un mediu Linux cu un kernel care suportă grupuri de control și izolarea spațiilor de nume; există versiuni pentru platformele x86-64 și ARM.

Pentru a economisi spațiu de stocare, proiectul utilizează sistemul de fișiere Aufs (o versiune alternativă a UnionFS, un sistem de fișiere auxiliar care efectuează montarea unificată în cascadă - permițând montarea simultană a mai multor sisteme de fișiere ca unul singur: pentru sistemele de fișiere Linux) cu suport pentru montarea unificată în cascadă: containerele utilizează imaginea sistemului de operare de bază, iar modificările sunt scrise într-o zonă

separată. Containerele sunt, de asemenea, acceptate în sistemul de fișiere Btrfs (un sistem de fișiere cu copiere la scriere pentru Linux bazat pe structuri de tip B-tree) cu funcția de copiere la scriere activată.

Software-ul include un daemon server de containere (executat cu comanda docker -d), instrumente client care vă permit să gestionați imagini și containere din interfața de linie de comandă și un API care vă permite să gestionați containerele în mod programatic în stil REST. Daemonul asigură izolarea completă a containerelor care rulează pe o gazdă la nivelul sistemului de fișiere (fiecare container are propriul sistem de fișiere rădăcină), la nivel de proces (procesele au acces numai la propriul sistem de fișiere al containerului, iar resursele sunt separate de libcontainer) și la nivel de rețea (fiecare container are acces numai la spațiul de nume de rețea legat de el și la interfețele de rețea virtuale corespunzătoare).

Un set de instrumente client permite pornirea proceselor în containere noi (docker run), oprirea și pornirea containerelor (docker stop și docker start), suspendarea și reluarea proceselor în containere (docker pause și docker unpause). O serie de comenzi permite monitorizarea proceselor în curs de execuție (docker ps prin analogie cu ps în sistemele Unix, docker top prin analogie cu top și altele). Se pot crea imagini noi dintr-un fișier script special (docker build, fișierul script se numește Dockerfile), este posibilă scrierea tuturor modificărilor efectuate în container în noua imagine (docker commit). Toate comenzile pot funcționa atât cu demonul docker al sistemului local, cât și cu orice server Docker disponibil în rețea. În plus, interfața de linie de comandă are încorporate capacități de interacțiune cu depozitul public Docker Hub, care găzduiește imagini de aplicații pre-construite, de exemplu, comanda docker search permite căutarea printre imaginile găzduite acolo, imaginile pot fi descărcate pe sistemul local (docker pull), este posibilă și trimiterea imaginilor construite local către Docker Hub (docker push).

Docker are, de asemenea, un manager de pachete, Docker Compose, care permite descrierea și rularea aplicațiilor multi-container; fișierele de configurare pentru acesta sunt descrise în YAML.

Sursa: Wikipedia

Dockerfile

Docker poate construi imagini în mod automat prin citirea instrucțiunilor dintr-un Dockerfile. Un Dockerfile este un document text care conține toate comenzile pe care un utilizator le poate

invoca în linia de comandă pentru a construi o imagine. Această pagină descrie comenzile care pot fi utilizate în Dockerfile.

Formatul Dockerfile:

Comentariu

Argumente INSTRUCTION

Instrucțiunile nu țin cont de majuscule și minuscule. Cu toate acestea, se obișnuiește ca acestea să fie scrise cu MAJUSCULE pentru a le distinge mai ușor de argumente. Docker execută instrucțiunile din Dockerfile în ordine. Un fișier Dockerfile trebuie să înceapă cu o instrucțiune FROM. Aceasta poate fi plasată după directivele parser, comentarii și ARG globale. Instrucțiunea FROM specifică imaginea părinte de la care se construiește. O instrucțiune FROM poate fi precedată doar de una sau mai multe instrucțiuni ARG, care declară argumentele utilizate în șirurile FROM din fișierul Docker. Docker tratează șirurile care încep cu # ca pe un comentariu, cu excepția cazului în care șirul este o directivă valabilă pentru parser. Un marker # oriunde altundeva în șirul de caractere este tratat ca un argument. Acest lucru permite expresii precum:

```
# Comment
RUN echo 'we are running some # of cool things'
```

Instrucțiuni Dockerfile:

- **FROM**

  ```
  FROM [--platform=<platform>] <image> [AS <name>]
  FROM [--platform=<platform>] <image>[:<tag>] [AS <name>]
  FROM [--platform=<platform>] <image>[@<digest>] [AS <name>]
  ```

  Instrucțiunea FROM inițializează o nouă etapă de construire și stabilește imaginea de bază pentru instrucțiunile următoare. Prin urmare, un fișier Docker corect trebuie să înceapă cu o instrucțiune FROM. Orice imagine adecvată poate fi utilizată ca imagine - este deosebit de convenabil să se înceapă cu o imagine din depozitele publice.

- **RUN**

  RUN <comandă> (forma shell, comanda este executată într-un shell, care este /bin/sh -c în Linux sau cmd /S /C în Windows în mod implicit)
  RUN ["executabil", "param1", "param2"] (forma de execuție)
  Comanda RUN va executa toate comenzile într-un nou strat peste imaginea curentă și va confirma rezultatele. Imaginea confirmată rezultată va fi utilizată pentru următorul pas din Dockerfile.

  RUN --mount vă permite să creați montări ale sistemului de fișiere pe care build-ul le poate accesa.

  RUN –mount=[type=<TYPE>][,option=<value>[,option=<value>]...]…]

  Tipuri de montaj:

  bind (implicită) - directoare de context de montaj bind (numai pentru citire). cache - Montează un director temporar pentru a stoca în memoria cache directoare pentru compilatoare și administratori de pachete.

  secret - Permite containerului de compilare să acceseze fișiere protejate, cum ar fi cheile private, fără a le coace într-o imagine.

  ssh - Permite containerului de compilare să acceseze cheile SSH prin intermediul agenților SSH, cu suport pentru fraze de trecere.
  RUN --network vă permite să controlați în ce mediu de rețea este executată comanda.

  RUN –network=<TYPE>.

  Tipuri de rețea:

  default - Se execută în rețeaua implicită.

  none - Se execută fără acces la rețea.

  host - Se execută în mediul de rețea gazdă.

- **CMD**

  CMD ["executabil", "param1", "param2"] (forma exec, aceasta este forma preferată)

CMD ["param1", "param2"] (ca parametri impliciți pentru ENTRYPOINT)
CMD command param1 param2 (forma shell)

Nu poate exista decât o singură instrucțiune CMD într-un fișier Dockerfile. Dacă sunt enumerate mai multe CMD, doar ultima CMD va avea efect. Scopul principal al unui CMD este de a furniza setări implicite pentru containerul executabil. Aceste valori implicite pot sau nu să includă executabilul, caz în care trebuie specificată și instrucțiunea ENTRYPOINT. În cazul în care CMD este utilizat pentru a furniza argumente implicite pentru instrucțiunea ENTRYPOINT, atât instrucțiunea CMD, cât și instrucțiunea ENTRYPOINT trebuie să fie specificate în format JSON array.

- **COPY**

COPY [--chown=<user>:<group>] [--chmod=<perms>] <src>... <dest>
COPY [--chown=<user>:<group>] [--chmod=<perms>] ["<src>",... "<dest>"]

Instrucțiunea COPY copiază noi fișiere sau directoare din <src> și le adaugă în sistemul de fișiere container în calea <dest>. Pot fi specificate mai multe resurse <src>, dar căile fișierelor și directoarelor vor fi interpretate ca fiind relative la sursa contextului de construire.

- **EXPOSE**

EXPOSE <port> [<port>/<protocol>...].

Instrucțiunea EXPOSE îi spune lui Docker că containerul ascultă pe porturile de rețea specificate în timpul execuției. Puteți specifica dacă portul ascultat este TCP sau UDP, iar dacă nu se specifică niciun protocol, TCP este implicit.

Pentru a invoca atât TCP, cât și UDP, includeți două linii:

EXPOSE 80/tcp

EXPOSE 80/udp

Sursa: https://docs.docker.com/engine/reference/builder/

Ghid de comenzi de terminal Linux: https://linuxjourney.com/lesson/the-shell

Sarcina principală:

Sarcina principală a acestei lucrări de laborator este de a crea o imagine și un container folosind tehnologia docker ca element de virtualizare, precum și de a rezolva sarcina din punct de vedere al complexității. Rezolvarea sarcinii cu momentul virtualizării docker, este necesar să se rezolve într-o mașină virtuală creată anterior, bazată pe sistemul Linux. În cazul în care

sistemul de operare principal este linux sau mac os, este permisă realizarea soluției în mod nativ.

Rezultatul final trebuie să conțină un fișier docker, precum și un program pentru rezolvarea variantei selectate.

Opțiuni pentru implementare:

- **Opțiune ușoară:** rulați sistemul pe care l-ați dezvoltat și demonstrați funcționarea componentelor sale.

- **Medie:** să rulați sistemul pe care l-ați dezvoltat și să demonstrați funcționarea componentelor sale în raport cu sarcina scriitorilor și a cititorilor.

- **Opțiune dificilă:** rulați sistemul pe care l-ați dezvoltat și demonstrați funcționarea acestuia în realizarea sarcinii unui barbier dormit.

Într-o frizerie există 1 frizer și n scaune pentru clienți care așteaptă.
Dacă ***v*** este varianta și ***s*** este numărul grupei atunci sarcina va fi:
Dacă v<=5, n= 2*v+5+2*s;
Dacă v>5, n= 2*v-6+s;

- când nu sunt clienți care așteaptă frizerul stă pe scaunul lui și doarme
- când apare un client, aceasta va trebuie să trezească frizerul
- dacă mai apare un client în timp ce frizerul tunde un client, aceasta va trebui să aștepte pe un scaun (dacă este liber unul) dacă nu va părăsi frizeria.
- ca număr de variantă, se ia cel mai mare număr al persoanei din subgrup ca număr de variantă

După finalizarea activității, scrieți un raport, care trebuie să includă - numele, prenumele, grupul, tema de lucru, varianta de implementare a sarcinii, o scurtă descriere a implementării sarcinii, un link către codul sursă pe GitHub. Împingeți codul sursă în ramura dvs. în depozitul corespunzător - https://github.com/FCIM-SO/Practice-Work-RO. Salvați raportul în format PDF și trimiteți-l la ELSE - https://else.fcim.utm.md.