Proxy strežnik za menedžiranje Docker kontejnerjov

Tehniška dokumentacija

Projektni seminar II

Matic Adamič, Pika Povh
Mentor: Vid Jagodič

Univerza na Primorskem
Fakulteta za matematiko, naravoslovje in informacijske tehnologije

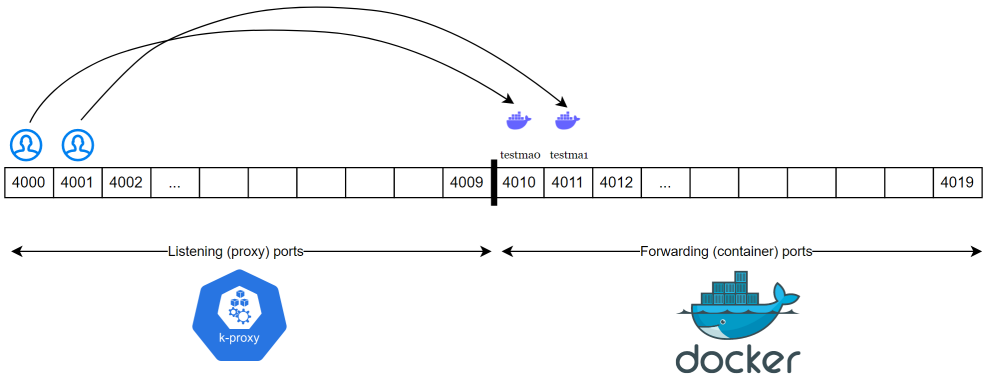
Junij 2022

Kazalo

1 Opis problema

Scenarij problema je sledeč: imamo strežnik (fizično strojno opremo oziroma en računalnik), na katerem imamo postavljen Docker. Strežnik gosti veliko število kontejnerjov, na katere uporabniki vzpostavljajo in prekinjajo povezave velikokrat na dan. Ko se enkrat kontejner postavi, na strežniku ostane v stanju izvajanja, dokler ga uporabnik ročno ne ugasne – kar se v praksi ne dogaja pogosto, saj uporabniki na njih pozabijo. Problem se pojavi, ko se skozi čas na strežniku nabere preveč kontejnerjov, ki porabijo veliko virov: CPU, RAM, disk, ... kar upočasni delovanje celotnega strežnika in odzivnost kontejnerjov in drugih operacij, ki jih strežnik izvaja dnevno.

Ideja rešitve je, da se na strežniku postavi aplikacijo oziroma strežnik, ki bo deloval kot namestniški strežnik (angl. proxy server), ki bo deloval kot posrednik podatkov med uporabnikom in kontejnerjom, na katerega se uporabnik povezuje. Poleg posredovanja podatkov, bo strežnik izvajal še ključno nalogo: ugašanje in prižiganje kontejnerjov. V primeru, da na nekem kontejnerju ni nobene aktivne povezave – torej ga nihče ne uporablja, in je v tem stanju že nek določen čas, potem ga bo strežnik ugasnil oziroma pavziral. To pomeni, da strežnik razbremeni. Ko se na kontejner poveže nov uporabnik, strežnik poskrbi, da se kontejner štarta oziroma zbudi iz pavziranega stanja. Osnovna ideja rešitve je vidna na poenostavljenem diagramu na sliki 1.

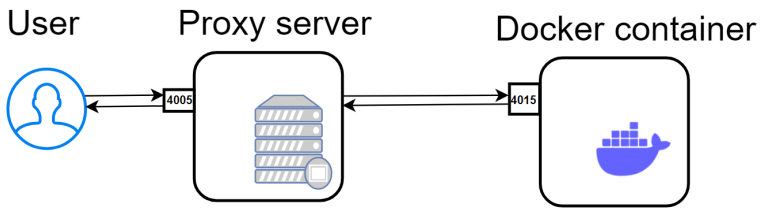


Slika 1: Diagram proxy strežnika in njegovo delovanje: poslušanje na portih [4000, 4009] in posredovanje kontejnerjom, ki poslušajo na portih [4010, 4019]

2 Analiza

Potrebno je vedeti, kako postaviti strežnik, ki zna posredovati podatke med dvema točkama. V našem scenariju sta točni uporabnik in kontejner. Podatki se bodo posredovali med dvema vratoma (angl. port). Uporabnik se poveže na port npr.: 4005, naš program zahtevo prestreže in jih posreduje na port npr.: 4015. To v programskem jeziku Java dosežen o z uporabo ServerSocket in Socket knjižnicami, ki so vključene v standardni distribuciji jezika [3]. Enostaven pregled scenarija, ko se uporabnik poveže na proxy strežnik je prikazan na sliki 2

Druga pomembna stvar je ključni del strežnika, ki mora znati ugašati in prižigati kontejnerje. Sam strežnik sam po sebi ne zna ustvariti in pognati kontejner, zna pa pavzirati in zbuditi (angl. resume) kontejnerje. Ko je to potrebno. Te funkcije ponuja in že sam Docker, do katerih dostopamo preko vmesnika z ukazno vrstico (angl. command-line interface CLI).



Slika 2: Osnovni pregled scenarija, ko uporabnik vzpostavi povezavo s kontejnerjom preko strežnika. Uporabnik se poveže na proxy strežnikov port 4005, proxy strežnik sprejme povezavo, in se poveže na kontejner na portu 4015. Strežnik nato posreduje podatke v obeh smereh: od uporabnika do kontejnerja in obratno.

3 Načrtovanje in zahteve sistema

Zahteve za strežnik so sledeče:

- Odzivnost: vsa komunikacija s kontejnerji bo potekala preko strežnika, zato mora biti hiter, uporabnik se naj ne bi zavedal, da dostopa do kontejnerjov preko strežnika.
- Zanesljivost: strežnik bo prižgan dlje časa, oziroma ni predvidenega časa izklopa, torej mora biti učinkovit in robusten glede nepričakovanih dogodkov
- Nastavljivost: koliko kontejnerjov lahko nadzoruje in nastavitev dostopa do strežnika, ki gostuje kontejnerje

Dodatne zahtevnosti, ki si jih ko uporabniki strežnika želimo so logiranje, izvajanje ukazov in nastavljivost časov, po katerem se kontejnerji ugasnejo.

3.1 Nastavljivost ob zagonu

Konfiguracija strežnika se bo definirala v obliki XML in bo sestavljena iz dveh datotek: *server-config.xml* in *ports-config.xml*. Datoteki bosta podani strežniku ob zagonu.

Primer konfiguracijske datoteke *server-config.xml* je vidna na sliki 3, struktura datoteke pa je sledeča:

- Podatek o avtentikaciji (uporabniško ime in privatni ključ) do strežnika, na katerem so gostovani kontejnerji (ponavadi je to en in isti strežnik)
- Možnost nastavitve pod-strežnika, na katerega se poveže aplikacija, ki skrbi za logiranje
- Možnost nastavitve pod-strežnika, na katerega se poveže uporabnik, za izpis logiranja v realnem času

Primer konfiguracijske datoteke *ports-config.xml* je vidna na sliki 4, struktura datoteke pa je sledeča:

- Podatek o imenu strežnika, na katerem so gostovani kontejnerji
- Razpon portov, na katerih strežnik posluša, in razpon portov, na katere strežnik podatke posreduje (razpon poslušanih portov so porti, na katere se povežejo uporabnik, razpon portov, na katere strežnik pošilja podatke, pa so porti kontejnerov)

```
<?xml version="1.0" encoding="UTF-8"?>
<broccoli>

    <host name="test" url="docker-host.si">
        <auth username="matic" keyfile="c:/ssh_keys/private_key"/>
    </host>

    <user_interface>
        <command_interface enable="true" port="777"/>
        <logging_interface enable="true" log4jPort="888" clientPort="999"/>
    </user_interface>

    <!--
     s: seconds
     m: minutes
     h: hours
     d: days
    -->
    <default_container_timeout/>1d<default_container_timeout/>
    <container_resume_timeout>2s</container_resume_timeout>

    <!-- Special containers -->
    <container containerPort="4200" timeout="5m"/>
    <container containerPort="4201" timeout="5m"/>
    <container containerPort="4202" timeout="5m"/>
</broccoli>
```

Slika 3: Primer nastavitvene datoteke *broccoli-config.xml*. Strežnik, na katerega se povezujemo smo nastavili na *"docker-host.si"*, in ga pomenovali "test". Uporabniško ime je *"matic"*, pot do privatnega ključa pa je *"C:/ssh_keys/private_key"*. Strežnik smo nastavili, tako, da bo poslušal ukaze na portu 777. Vsak, ki bo želel spremljati logiranje strežnika, pa se lahko poveže na port 999. Vsak kontejner se pavzira po enem dnevu neuporabe, kar je nastavljeno z vrednostjo *"1d"* (angl. 1 day). Strežnik bo po tem, ko zbudi kontejner, počakal 2 sekundi, predenj z njem poskuša vzpostaviti povezavo, specificirano s parametrom *"2s"* (angl 2 seconds). Specificiranih je nekaj posebnih kontejnerjev, na portih 4200-4202, ki se ugasnejo po 5 minutah neuporabe, specificirano s parametrom *"5m"* (angl. 5 minutes).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ports>
    <server name="test">
        <range start="4000" end="4200" remap="100"/>
    </server>
    <server name="prod">
        <range start="5000" end="5400" remap="200"/>
    </server>
</ports>
```

Slika 4: Primer nastavitvene datoteke *ports-config.xml*. Nastavljeno je poslušanje na portih [9000, 9009], in posredovanje na porte [9010, 9019], na katerih pričakujemo, da poslušajo docker kontejnerji.

3.2 Izvajanje ukazov

Podprti morajo biti ukazi:

- *start*: zagon novega posredovalnega proxy pod-strežnika
- *disconnect*: prekinitev vseh trenutnih povezav posredovalnega proxy pod-strežnika
- *restart*: ponovni zagon posredovalnega proxy pod-strežnika
- *info*: izpis vseh informacij o strežniku in njegovem trenutnem stanju
- *log*: časovno neomejena povezava na strežnik, ki nam posreduje vsa logiranja v realnem času (angl. monitoring)

Uporabnik pošilja ukaze strežniku z porabo ukazne vrstice. Na strežnik lahko pošiljamo ukaze s poljubnim programom, ali pa uporabimo Bash skripto broc, ki vhodne parametre posreduje na strežnik in uporabniku vrača odgovore. Skripto lahko na Windows operacijskem sistemu zaganjamo preko Cygwin.

3.2.1 Start

Ukaz deluje v dveh načinih: s podano eno številko porta ali s podanima dvema številkama portov. V primeru podane ene številke, strežnik odpre nov proxy pod-strežnik, ki posluša na specificiranem portu in posreduje na port, ki je implicitno določen iz nastavitvene datoteke.

V primeru dveh podanih portov, prvi predstavlja številko porta na kateri nov proxy pod-strežnik posluša, druga pa na katero posreduje. Če je specificiran port že zaseden, ali ne sledi določenim razponom portov, ki so podani v konfiguracijski datoteki, potem strežnik javi napako.

3.2.2 Disconnect

Ukaz sprejme eno port številko, za katero ugasne vse trenutno žive povezave.

3.2.3 Restart

Ukaz sprejme eno port številko. V tem za dan proxy pod-strežnik izvede najprej ukaz *disconnect* in ugasne pod-strežnik, potem pa ga nazaj prižge z ukazom start.

3.2.4 Info

Ukaz sprejme eno port številko ali noben parameter. V primeru ene številke izpiše informacije o enem proxy pod-strežniku. V primeru nobenega parametra izpiše vse informacije o vseh pod-strežnikih: proxy, logging strežniku in strežniku za izvajanje ukazov. Za vsak pod-strežnik izpiše, koliko časa je prižgan in koliko ima aktivnih povezav.

3.2.5 Log

Ukaz ne sprejme nobenega parametra. Ob zagonu ukaza vzpostavimo povezavo s strežnikom, od katerega začnemo prejemati logiranje dokler povezave samo ne prekinemo.

4 Uporabljena tehnologija

4.1 Java

Java je preprost, učinkovit jezik splošnega namena, prvotno zasnovan za vgrajene omrežne aplikacije, ki delujejo na več platformah. Je prenosen, objektno orientiran, interpretiran in prevajan jezik. En izmed glavnih ciljev in moto jezika je: *"Write once, run anywhere"*, kar pomeni, da lahko program napišemo enktar, in ga poganjamo na kateri koli strojni opremi - jezik je tako imenovano neodvisen od platforme na kateri se izvaja (angl. platform independent).

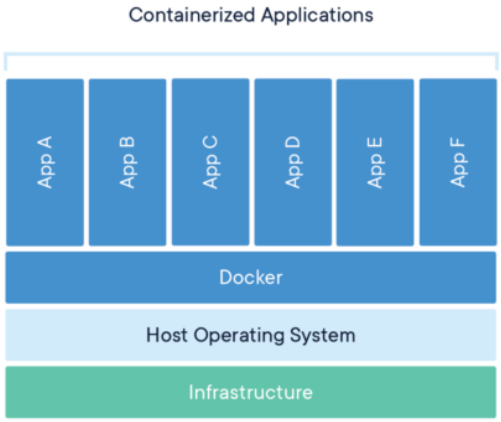
Na sintakso Jave v veliki meri vplivata C++ in C. Za razliko od C++ je bila Java zgrajena skoraj izključno kot objektno usmerjen jezik. Vsa koda je napisana v razredih (angl. classes), podatki pa so instance razredov, tako imenovani objekti (angl. objects) - z izjemo primitivnih podatkovnih tipov kot so cela števila (angl. integers), števila s plavajočo vejico oziroma realna števila (angl. floating point numbers), zaradi performančnih razlogov [2].

4.2 Docker

Docker je programska platforma, ki vam omogoča hitro izdelavo, testiranje in uvajanje aplikacij. Docker je bil prvič uradno predstavljen leta 2013 Docker in je danes ena izmed najbolj uporabljenih tehnologij v računalniškem svetu.

Omogoča enostavno pakiranje programske opreme v standardizirane enote, imenovane kontejnerji. Podobno kot je programski jezik Java neodvisna od stropjne opreme, enako velja za kontejnerje. Vsak kontejner, v katerega zapakiramo svojo programsko opremo, vsebuje vse, kar naš program potrebuje za svoje izvajanje. Vsak kontejner si lahko predstavljamo kot lahek in hitrer (angl. light-weight) virtualni operacijski sistem, ki gostuje naš program. Virtualni operacijski sistem, ki gostuje poljuben program, je Linux kernel.

Docker lahko namestimo na vse popularne operacijske sisteme, na primer Windows ali poljubno Linux distribucijo. Glavni del docker programa je *Docker Engine*, ki skrbi za poganjanje poljubnega števila kontejnerjov. Kontejnerji lahko med seboj komunicirajo preko virtualnega omrežnega sistema, ki ga simulira Docker. Vsakemu kontejnerju lahko nastavimo port, preko katerega lahko komunicira z zunanjim svetom [1, 4].

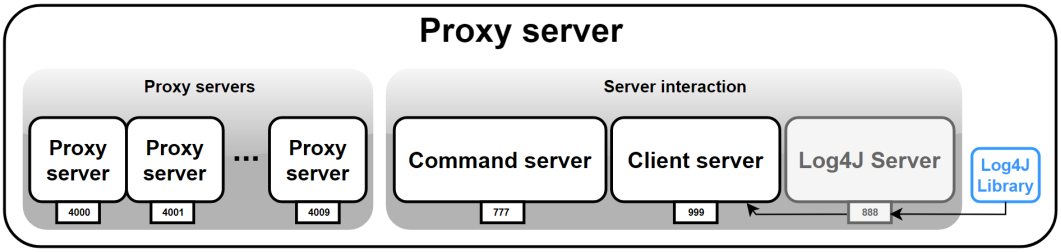


Slika 5: Preprosti diagram prikaza abstraktnega delovanja Dockerja. Vsaka aplikacija (A-F) je svoj virtualni operacijski sistem, ki gostuje svojo aplikacijo.

5 Implementacija

Celotna rešitev je implementirana v Javi, predvsem z uporabo *ServerSocker* in *Socket* knjižnic. Za logiranje je uporabljena knjižnica *Log4J*. Strežnik se ob zagonu konfigurira z konfiguracijskima datotekama *server-config.xml* in *ports-config.xml*.

Strežnik je sestavljen iz večih pod-strežnikov in modulov, ki skrbijo za izvajanje ukazov in spremljanje stanja strežnika preko logiranja. Osvnovna struktura implementacije strežnika je vidna na sliki 6.



Slika 6: Osnovna struktura implementacije strežnika. Na konkretnem primeru diagram prikazuje postavitev proxy strežnika z desetimi pod-strežniki na portih v razponu [4000, 4010). Omogočeno je izvajanje ukazov preko porta 777 in poslušanje logiranje preko porta 999. Zunanja knjižnica Log4J se poveže na pod-strežnik na portu 888, ki pošilja vse loge morebitnim uporabnikom, ki se lahko povežejo na port 999

5.1 Proxy pod-strežniki

Proxy pod-strežniki so niti, ki poslušajo na določenem portu, in skrbijo za sprejemanje povezav, na podlagi zahtev uporabnikov. Vsakemu pod-strežniku sta dodeljena dva porta: poslušalni port in posredovalni port.

Na poslušalnem portu se posluša uporabnike, ki pošiljajo zahteve za vzpostavitev povezave. Ob takem dogodku, se povezavo sprejme, če strežnik najde točno en kontejner, ki posluša na posredovalnem portu. To strežnik doseže z Docker ukazom:

```
$ docker ps -filter publish="port_kontejnerja" --format {{.Names}}
```

Argument *"ps"* pomeni, naj Docker izpiše stanje kontejnerjov. Argument *"–all"* pomeni, naj upošteva vse kontejnerje, ne glede na njegovo stanj. V argumentu filter, filtriramo kontejnerje glede na parameter, v tem primeru filtriramo glede na številko porta. V argumentu *"–format"* povemo, katere informacije želimo na izpisu. V tem primeru, želimo samo njegovo ime. Zgornji ukaz nam tako vrne ime kontejnerja, ki posluša na specificiranem portu. V primeru, da tak kontejner ne obstaja, se povezavo zavrne. V nasprotnem primeru se preveri stanje kontejnerja z Docker ukazom:

```
$ docker ps -all -filter=name="ime_kontejnerja" --format {{.Status}}
```

Argument *"–format"* s specifikacijo *".Status"* pove, naj za določen kontejner, izpiše njegov status.V primeru vrnjenega stanja *"Exited"*, to pomeni, da je kontejner pavziran oziroma spi – v tem primeru ga zbudimo z ukazom

```
$ docker start ime_kontejnerja
```

V primeru vrnjenega stanja *"Running"* je kontejner že prižgan, in lahko nemudoma vzpostavimo povezavo. V primeru vzpostavitve povezave, ustvarimo dve novi posredovalni oziroma proxy niti, ki skrbita za pretok podatkov med uporabnikom in kontejnerjom: ena nit skrbi za pretok od uporabnika do kontejnerja, druga pa v obratni smeri. Dve niti sta potrebni za možni istočasni prenos podatkov v obe smeri. Obe niti med samo povežemo na sledeč način: če se povezava v eni niti prekine, ali pride do nepričakovane napake, niti ena drugo ustavita, tako da ne more priti do nekonsistentnega ali nesmiselnega stanja v posredovanju podatkov samo v eni smeri. V primeru, da kontejner ali uporabnik prekineta povezavo v eni smeri, se obe niti nehata izvajati. Strežnik ob vsaki prekinitvi povezave parov niti preveri, ali je bila to zadnja aktivna povezava na kontejnerju. V takem primeru, ustvari novo nit, ki je namenjena ugašanju kontejnerja. Ob tem nit za ugašanje kontejnerja zaspi za določen čas, ki je prebran iz konfiguracijske datoteke. Ko se nit zbudi, izvede Docker ukaz, ki kontejner pavzira:

```
$ docker stop ime_kontejnerja
```

Po izvedbi ukaza se nit ugasne. V primeru, da se v času spanja niti za ugašanje kontejnerja na kontejner vzpostavi nova povezava, se čas spanja niti ponastavi. Ukaz *disconnect* ugasne vse take pare niti, ki so prižgane za določen proxy pod-strežnik.

5.2 Logiranje

Za logiranje stanja strežnika poskrbi knjižnica *Log4J*. Knjižnici nastavimo, da se ob zagonu poveže na pod-strežnik, ki je namenjen izključno njej. Ta strežnik sprejme povezavo, ki jo knjižnica vzpostavi, in posreduje vsa sporočila na pod-strežnik, na katerega se povezujejo klienti z ukazom log. Knjižnici dodatno nastavimo nastavitev, naj logira vse tudi v datoteke *.log*. Politika logiranja je določena z omejitvami: strežnik lashko ustvari maksimalno 5 datotek, pri kateri je vsaka lahko velika do 10MB, predenj se stari logi začnejo prepisovati.

Hkrati strežnik ustvari dodatno *.log* datoteko, kamor zapisuje samo napake (angl. log lever error) in opozorila (angl. log level warning), za lažji nadzor oziroma opazovanje delovanja strežnika. Politika te datoteke omejuje datoteko samo na ena in maksimalno velikost 10MB, predenj se logi začnejo prepisovati.

5.3 Log pod-strežnika

Strežnika za logiranje sta med seboj povezana oziroma soodvisna. Prvi log podstrežnik, imenovan *log4j pod-strežnik* je namenjen knjižnici *Log4J*, v kateri je realizirano logiranje. Namen tega pod-strežnika je, da vzpostavi povezavo z Log4J knjižnico, katera mu posreduje vsa logiranja. V tem strežniku je predvidena 1 aktivna povezava, ki naj se ne bi nikoli prekinila. Povezava se vzpostavi takoj ob zagonu strežnika.

Drugi log pod-strežnik, imenovan *log-klientni podstrežnik* je namenjen uporabnikom, ki želijo spremljati dogajanje strežnika v realnem času. Nanj se povezujejo zunanji uporabniki.
Strežnika sta soodvisna na sledeč način: vsi klienti log4j strežnika (naj bi bil en; knjižnica Log4J) posredujejo izhode na vse kliente, ki so povezani na log-klient pod-strežniku. Tako vsi uporabniki prejemajo logiranja, ki se dogajajo na strežniku v realnem času.

5.4 Ukazni pod-strežnik

Ukazni pod-strežnik je namenjen iterpretaciji in izvajanju ukazom, ki so bili opisani v sekciji 3.2. Ta strežnik ne vzpostavlja nobenih povezav, vendar kot vhod dobi niz (angl. string), ki ga zna razčleniti in izvesti. Ukaz na koncu izvajanja ustvari sporočilo, o njegovi uspešni ali neuspešni izvedbi, ki ga posreduje uporabniku.

5.5 Večnitnost in izvajanje Docker ukazov

Zaradi večnitnosti strežnika je seveda treba ustvariti sinhronizacijo med nitmi. Sinhronizacija je narejena na nivoju kontejnerjov, in preprečuje nesmiselno hkratno izvajanje Docker ukazov, npr: *docker stop* in *docker start*.

Zgodi se lahko namreč scenarij: nit za sprejemanje povezav izvede ukaz, ki vrne stanje kontejnerja, v tem primeru naj bo kontejner prižgan. Zaradi razporejevalnika niti (angl. thead scheaduler), na ravni operacijskega sistema, se nit preneha izvajati, in pride na vrsto nit, ki ugaša kontejner. Naj se ta nit zbudi, in ker ni nobene aktivne povezave, izvede ukaz *docker stop*, ki pavzira kontejner. Ta nit se nato ugasne, in za njo se začne izvajati prva nit,m ki sprejema povezavo. Nit ima še iz prvega dela izvajanja v spominu, da je ravno preverila stanje kontejnerja, in izmerjeno stanje je bilo, da je kontejner aktiven (čeprav je v

resnici kontejner ustavljen). Nit nadaljuje s sprejemanjem povezave, kar je seveda neuspešno, saj kontejner ni prižgan. Povezava nepričakovano pade. Rešitev: Na začetku sprejemanja povezave, se dostop do izvajanje docker ukazov za specifičen kontejner zaklene. Tako je lahko povezava sinhrono sprejeta ali zavrnjena na varen način. Na koncu sprejemanja ali zavračanja povezave je ključavnica sproščena.

6 Sodelovanje

- Matic Adamič: implementacija, načrtovanje, dokumentacija implementacije v tehniški dokumentaciji, dokumentacija v uporabniški dokumentaciji
- Pika Povh: načrtovanje, dokumentacija uporabljenih tehnologij, dokumentacija v uporabniški dokumentaciji, načrtovanje in izvedba vseh konfiguracijskih datotek

Literatura

[1] Amazon. What is doceker?

[2] J. Gosling and H. McGilton. Design goals of the java™ programming language.

[3] Oracle. Writing the server side of a socket.

[4] S. Vaughan-Nichols. What is docker and why is it so darn popular?