Haaga-Helia
ammattikorkeakoulu Oy

Minecraft-palvelimen ylläpito- ja ajo-ohjelma

Elmo Rohula

Linux palvelimet

13.10.2019

Tiivistelmä

Haaga-Helia
ammattikorkeakoulu Oy

Tekijä(t)
Rohula Elmo

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Raportin/Opinnäytetyön nimi
Minecraft-palvelimen ylläpito- ja ajo-ohjelma

Sivu- ja liitesivumäärä
6 + 1

Lopputyön aiheeksi olin alun perin ajatellut pelkän Minecraft palvelimen asentamista ja sen pyörittämistä. Kuitenkin huomasin, että itse palvelimen asennus, käynnistys, sekä ajaminen tuotantoympäristössäkin oli suhteellisen helppoa. Tästä innostuneena päätin tehdä palvelimelle ohjelman, jolla pystyisi ylläpitämään ja ohjaamaan palvelimen toimintaa muutamalla komennolla.

Pelkän Minecraft-palvelimen ajo Ubuntu-palvelin ympäristössä on suhteellisen yksinkertaista. Tarvitset vain yhden palvelintiedoston, joka ensi kertaa ajettaessa luo/hakee internetistä tarpeelliset tiedostot ja hakemistot palvelimen ajoa varten.

Huomasin kuitenkin, että palvelimen ajo suoraan ilman jonkin näköistä päätteen multipleksausta oli huomattava riski, sillä jos yhteys palvelimen ja isäntäkoneen välillä katkesi, koko palvelin kaatui ja pahimmassa tapauksessa korruptoi tiedostoja.

Ideana olisi siis kehittää scripti/ohjelma, joka automaattisesti käynnistäisi palvelimen virtuaalisella päätteellä, jolloin yhteyden mahdollisesti katketessa palvelimelle, Minecraftin prosessi jäisi pyörimään virtuaalisella päätteellä taustalle. Päätteen multipleksauksesta on myös se hyöty, että ajetut ohjelmat voidaan jättää taustalle ja palvelinta voidaan hallinnoida pysäyttämättä ajossa olevia prosesseja.

Minecraftin-palvelin prosessin pyöriessä voidaan palvelinta ohjata muutamalla komennolla, pelin sisäistä säätä voidaan muuttaa, pelaajia voidaan potkia ulos palvelimelta jne. Päätin sisällyttää kirjoittamaani ohjelmaan muutamat näistä ominaisuuksista. Tämä tapahtui helposti, sillä scriptin piti vain lähettää komento tekstimuodossa virtuaalisella päätteelle ja painaa enteriä.

Loppujen lopuksi itse ohjelman ohjelmointi ei ollut niin haastavaa. Haasteena oli vain yksittäisten komentojen testaus ja opettelu, sekä ohjelmaprosessin järkeistäminen.

Asiasanat
Minecraft palvelin Linux bash script ohjelmointi

Sisällys

1 Minecraft-palvelimen asennus ja käynnistys

Minecraft-palvelimen avulla voidaan Minecraft-pelin maailmaa pitää ajossa silloinkin kun itse pelin maailmaan ei ole kirjautuneena yhtään pelaajaa. Ilman palvelinta pelin maailma on muiden pelaajien käytössä vain silloin, kun maailmaa ylläpitävä pelaaja on pelaamassa ja jakamassa maailmaansa verkkoon.

Palvelimen ajamiseen Ubuntu-palvelimella vaaditaan Java Runtime Environment, sekä itse Minecraft-palvelin. Minecraftin uusimman palvelimen löytää aina Minecraftin sivuilta (https://www.minecraft.net/fi-fi/download/server/), tai haluttaessa voidaan vanhempia tai muokattuja versioita etsiä netistä. On myös suotavaa, että Ubuntu-palvelimella, jolla Minecraftia pyöritään, olisi luotu erillinen käyttäjä pelkästään palvelimen ajoa varten. Suurin osa muista vaadittavista ohjelmista tulee valmiiksi asennettua Ubuntu-palvelimella.

Aloitin palvelimen asennuksen luomalla käyttäjän ”minecraft” ja lisäämällä käyttäjän samannimiseen ryhmään. Käyttäjän tulee olla järjestelmäkäyttäjä. Loin myös käyttäjälle kotihakemiston, joka on sama, johon Minecraft-palvelimen tiedostot luodaan

(' adduser --system --shell /bin/bash --home /opt/minecraft --group minecraft').

Tarkistin myös oliko palvelimelle jo asennettu Javan JRE:tä. Ei ollut, joten asensin senkin

('sudo apt-get install default-jdk').

Komento asentaa viimeisimmän/suositellun Javan Development Kitin. JDK:ssa tulee myös vaadittu JRE, mutta asensin JDK:n sitä varten, jos tulevaisuudessa tulen kehittämään Java sovelluksia Ubuntu-ympäristössä. Tämän jälkeen hain itse Minecraft-palvelimen ajotiedoston wgetillä. Testausympäristössä en saanut jaettua leikepöytää toimimaan, joten linkin kopioinnin sijaan vein tiedoston virtuaalikoneelle sftp:llä.

Tämän jälkeen koeajoin palvelimen komennolla *'java -Xmx1G -Xms1G -jar server.jar'*. Komento määrittelee, että ajetun prosessin minimi- ja maksimimäärä muistia tulee olemaan 1024Mb ja että ajettava tiedosto on ”.jar” muotoinen. Palvelin jakaa pelimaailmaa omasta IP-osoitteestaan, oletuksena portista 25565, joten avasin sen (*' ufw allow 25565'*). Kutsumattomista vieraista ei tarvitse huolehtia, sillä Minecraft-palvelin käyttää omaa whitelistiä, joka päästää oletuksena palvelimelle vain sallitut käyttäjät käyttäjien tunnuksen mukaan.

Kirjauduin peliin omalla koneellani ja pääsin sisälle palvelimen maailmaan. Enää oli tehtävänä kirjoittaa ohjelma, jolla pystyisin hallinnoimaan palvelinta.

2 Ohjelman kirjoittaminen

Päätin pienen haasteen luomiseksi luoda ohjelman täysin testiympäristön sisällä vimillä. Ajatuksena oli myös, että voisin kutsua ohjelmaa mistä vain ilman relatiivista tai absoluuttista polkua siihen. Minun oli siis vietävä ohjelma kansioon, josta palvelimen shell osaisi sitä automaattisesti hakea, tai luotava kansio ja määriteltävä PATH-muuttujaan tämän kansion sijainti. Tein tämän vasta ohjelman ollessa ajokunnossa. Käytin aluksi apuna ohjelmalogiikan selvittämisessä jo valmista ”start-up scriptiä”, jonka löysin netistä (https://minecraft.gamepedia.com/Tutorials/Server_startup_script). Törmäsin kuitenkin ongelmiin käyttäessäni ja testatessani screeniä. Päätin vaihtaa päätteen virtualisoivan ohjelman tmuxiin.

Suurin osa ajasta, jonka käytin ohjelman luontiin meni eri komentojen kokeiluun ja Ubuntun ohjelmien opetteluun. Man-sivut olivat suuri apu tässä vaiheessa. Sen sijaan, että olisin kopioinut suoraan scriptin pohjan ja muokannut sitä, kokeilin kaikki prosessit manuaalisesti, tutkin mihin ne johtivat, mitä ne tekivät ja tulostivat, sekä katsoin miten nämä prosessit toimivat scriptin logiikassa.

Aloitin scriptin siitä, että loin funktion, jolla palvelin käynnistyy tmuxissa. Valmiista sciptistä eroten päätin lisätä yhden varmistusvaiheen, jolla scripti todella varmentaa, että palvelin on käynnissä ja maailma on jaettuna ulkoverkkoon.

Valmis scripti tarkistaa palvelimen tilan komennolla

(’pgrep -u $käyttäjänimi -f $pavelu’)

ja vertaa sitä /dev/nulliin. Komennolla katsotaan aikaisemmin määritellyn käyttäjänimen mukaan, onko kyseisellä käyttäjällä ajossa prosessia, jonka nimi on myös aikaisemmin määritelty. Jos tämä komento antaa muuta kuin tyhjän, niin ohjelma olettaa, että palvelin on käynnissä. Törmäsin kuitenkin tilanteeseen, jossa ohjelma kertoi palvelimen olevan käynnissä, mutta kuitenkin yrittäessäni avata tmuxia sain virheilmoituksen siitä, että yhtään tmux prosessia ei ollut käynnissä. Tarkistin Minecraft-palvelimen lokitiedostosta, että palvelimen käynnistys oli keskeytynyt, sillä olin ajanut sen väärällä käyttäjällä, mutta palvelimen prosessi oli ollut käynnissä sen aikaa, että scripti oli kerennyt katsoa prosessin olevan käynnissä.

Lisäsinkin sciptiin while loopin joka pyörii niin kauan, kunnes se lukee grepillä Minecraft-palvelimen lokista, että palvelin todella on jaettuna ulkoiseen verkkoon. Tämä tieto näkyy lokissa tekstinä ”Done”, joten sitä oli suhteellisen helppo etsiä.

```
if pgrep -u root -f server.jar > /dev/null ; then
    echo -e "\rserver.jar is now running!"
else
    echo -e "\rERROR! Could not start server.jar"
fi
gfx_sleep
while true; do

    if grep -q ": Done" ${LOG}; then
        echo -e "\rServer ready!"
        echo -e "\n"
        echo "Use the command '$0 ctl_opentmux' to open"
        echo "administration view on tmux or use the command"
        echo "'$0 help' to view available commands"
        gfx_sleep
        break
    else
        echo -e "\rWaiting for 'Done'"
        gfx_spin &
        PID=$!
        disown
        sleep 5
        kill $PID
    fi
done
```

Lisätty while-luuppi.

Olen ajatellut vielä lisääväni jonkin ajastimen, jolla while-luuppi pyörii vain tietyn aikaa tai muuttujan, jolla määritellään odotuksien määrä ennen kuin ohjelma antaa virheilmoituksen. Tällä hetkellä ohjelma pyörii maailmanloppuun asti, jos se ei koskaan lue lokista donea.

Lisäsin myös palvelimen sulkemiseen funktion. Funktiosta jätin tässä vaiheessa pois lokitiedoston lukemisen, sillä kaikilla – myös virheellisillä – testiajoilla palvelin on aina sulkeutunut ongelmitta. Sulkemisfunktio on siitä erilainen verrattuna käynnistysfunktioon, että sulkemisfunktio lähettää käynnistysvaiheessa luodulle tmux-sessiolle näppäimen painalluksia. Tämä on käytännössä sama kuin se, että menisin manuaalisesti tmux-sessioon ja kirjoittaisin palvelimelle sulkemiskomennon itse. Screeniin verrattuna tmuxissa syötteen lähettäminen on paljon intuitiivisempaa ja tapahtuu komennolla

('tmux send-keys [parametrejä ja muuttujia]').

```
srv_stop(){
    if pgrep -u root -f server.jar > /dev/null ; then
        echo "Stopping server.jar"
        gfx_spin &
        PID=$!
        disown
        tmux send-keys -t minecraft_server.0 "say Server shutting down, saving map" ENTER
        tmux send-keys -t minecraft_server.0 "save-all" ENTER
        sleep 10
        tmux send-keys -t minecraft_server.0 "stop" ENTER
        sleep 7
        kill $PID
        echo -e '\r   '
    else
        echo "server.jar was not running"

    fi
    if pgrep -u root -f server.jar > /dev/null ; then
        echo "server.jar could not be stopped!"
    else
        echo "server.jar is stopped"
        gfx_sleep
    fi
}
```

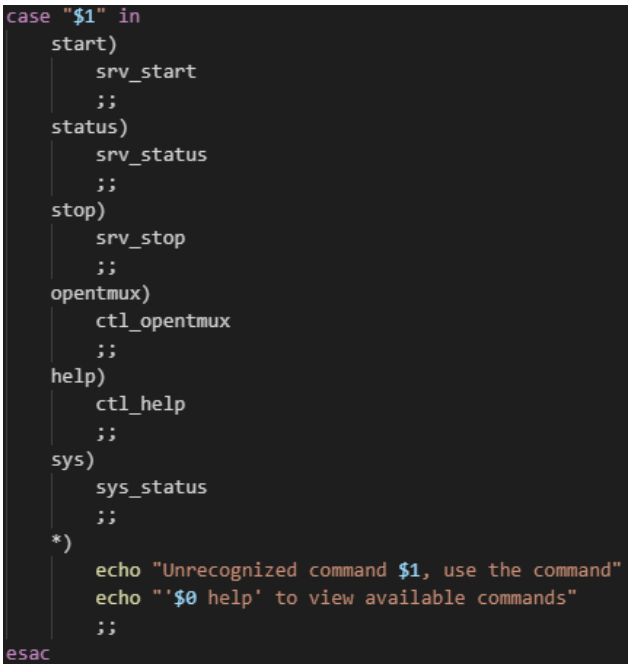
Ohjelman srv_stop-funktio, jolla palvelin suljetaan

Tässä vaiheessa lisäilin parit pienemmät funktion, lähinnä käyttöliittymän pientä animointia ja apukomentoja muulle ohjelmalle.

Lopuksi tein ”case”-lukijan, joka lukee käyttäjän ohjelmalle antamia parametrejä, joiden mukaan ohjelma ajaa eri funktioita. Esimerkiksi komento

(*'./testscr.sh start'*)

ajaa ohjelmasta palvelimen käynnistysfunktion.



```
case "$1" in
    start)
        srv_start
        ;;
    status)
        srv_status
        ;;
    stop)
        srv_stop
        ;;
    opentmux)
        ctl_opentmux
        ;;
    help)
        ctl_help
        ;;
    sys)
        sys_status
        ;;
    *)
        echo "Unrecognized command $1, use the command"
        echo "'$0 help' to view available commands"
        ;;
esac
```

Case-lukija. Lista myös tällä hetkellä valmiina olevista funktioista

3 Ohjelman lopullinen sijoitus

Tässä vaiheessa koin ohjelman olevan valmis, ainakin alkuperäisen speksauksen mukaan. Nyt ohjelmaa piti saada ajettua mistä vain koneella, tähän mennessä olin ajanut sen aina relatiivisella polulla *"./<ohjelman-nimi>"*. Sen sijaan, että olisin määritellyt ohjelman sijainnin PATH muuttujaan, päätin viedä sen hakemistoon /usr/local/bin. Hakemisto on tarkoitettu käyttäjän luomille ohjelmille, joita voidaan ajaa ”normaaleilla” käyttäjän oikeuksilla. Ohjelmalle oli tätä ennen annettu ajo-oikeudet

('sudo chmod 773 testscr.sh').

Tämän jälkeen pystyin helposti käynnistämään ja sulkemaan pelipalvelimen ilman, että olisin joutunut avaamaan tmuxia ja ajamaan palvelimenhallintakomentoja manuaalisesti.

Olin ajatellut itse ohjelman kirjoittamisen tuottavan virhetilanteita ja päänvaivaa, mutta toisin kävi. Kuten mainittu jo aikaisemmin, haastavin vaihe oli erinäisten komentojen kokeilu ja ohjelmien opettelu Ubuntu-palvelimella. Kun opin käyttämään niitä, niin erinäisten tulosteiden prosessointi ja ohjelmalogiikan rakennus oli suhteellisen helppoa.

4 Lähteet

1. Scipti-pohja, sekä sen esivaatimukset (https://minecraft.gamepedia.com/Tutorials/Server_startup_script)

2. Minecraft-palvelimen asennus ja ajo yleisesti (https://minecraft.gamepedia.com/Tutorials/Setting_up_a_server)

5 Liitteet

1. testscr.sh – ohjelma siinä muodossaan kuin se tuli ”valmiiksi” testiympäristössä