

PROGRAM

SPLETNE IN INFORMACIJSKE TEHNOLOGIJE

**NAČRT ZA IZVEDBO NALOGE: MINECRAFT STREŽNIŠKA PLATFORMA NA AWS**



**Mentor**: pred. Dušan Fugina **Študent/-a**: Christian Adam

Jani Kojc

**Maribor, maj 2025**

**POVZETEK**

V tej seminarski nalogi je predstavljen celovit postopek načrtovanja, konfiguracije in zagona Minecraft strežnika v okolju AWS. Postavili smo ločena podomrežja, konfigurirali Internet Gateway, NAT Gateway, varnostne skupine in zagnali dve EC2 instanci (bastion host in strežnik). Implementacija vključuje avtomatizacijo varnostnih kopij z uporabo AWS Lambda in EventBridge ter spletni nadzorni vmesnik s pomočjo Flask aplikacije na Apache strežniku. Dokument podrobno opisuje posamezne korake, arhitekturne odločitve in vključuje vizualne dokaze o delovanju.

Naloga dokazuje, da je mogoče s premišljenim načrtovanjem, uporabo AWS storitev in osnovami sistemske avtomatizacije vzpostaviti kompleksno platformo, ki omogoča tako zabavno kot profesionalno uporabo. Rešitev je razširljiva, odporna na napake in pripravljena na nadaljnji razvoj (npr. nadzor igralcev z metrikami, samodejno obvestilo o polni strežniški kapaciteti, povezava z Discord strežnikom).

**KAZALO VSEBINE**

[1 UVOD 1](#_Toc198038605)

[1.1 CILJI NALOGE 1](#_Toc198038606)

[2 OPIS UPORABLJENE AWS INFRASTRUKTURE 2](#_Toc198038607)

[2.1 Podomrežja (subnets) 2](#_Toc198038608)

[2.2 Internet Gateway in Route Table 2](#_Toc198038609)

[2.3 Varnostne skupine (Security Groups) 2](#_Toc198038610)

[2.4 Aurora RDS (MySQL) 3](#_Toc198038611)

[3 IMPLEMENTACIJSKI KORAKI 4](#_Toc198038612)

[3.1 Zagon EC2 instanc 4](#_Toc198038613)

[3.2 SSH povezava preko bastion hosta 5](#_Toc198038614)

[3.3 Omogočanje internetnega dostopa za posodobitve na MC server 7](#_Toc198038615)

[3.4 Namestitev Docker in zagon MC strežnikov v kontejnerjih 10](#_Toc198038616)

[3.4.1 Konfiguracija strežniških lastnosti 12](#_Toc198038617)

[3.4.2 Oblikovanje direktorijev in volumnov 13](#_Toc198038618)

[4 POVEZAVA IN TEST STREŽNIKOV 14](#_Toc198038619)

[5 POTRDITEV DELOVANJA 15](#_Toc198038620)

[6 NAMESTITEV AWS CLI 15](#_Toc198038621)

[7 DODELITEV IAM VLOGE EC2 INSTANCI 17](#_Toc198038622)

[8 AVTOMATIZACIJA BACKUPOV PRKEO LAMBDA FUNKCIJE 19](#_Toc198038623)

[9 USTVARJANJE EVENTBRIDGE ČASOVNIKA ZA BACKUP 23](#_Toc198038624)

[10 NAMESTITEV FLASK APLIKACIJE NA APACHE STREŽNIK 25](#_Toc198038625)

[11 FUNKCIONALNOSTI ADMIN UI APLIKACIJE 27](#_Toc198038626)

[12 SAMODEJNI IZKLOP STREŽNIKA OB NEDEJAVNOSTI 27](#_Toc198038627)

[13 AVTOMATIZIRANO NAMEŠČANJE APLIKACIJE PREK GITHUB ACTIONS 28](#_Toc198038628)

[14 DEPLOYMENT APLIKACIJE Z GITHUB REPOZITORIJA 29](#_Toc198038629)

[15 DODAJANJE UPORABNIKA NA GITHUB REPOZITORIJ 32](#_Toc198038630)

[16 ADMIN CENTER – UPRAVLJANJE UPORABNIKOV 33](#_Toc198038631)

[17 PODATKOVNA BAZA UPORABNIKOV Z MYSQL NA AWS RDS 34](#_Toc198038632)

[18 PREVERJANJE POVEZAVE APLIKACIJE S PODATKOVNO BAZO 35](#_Toc198038633)

[19 TESTIRANJE RCON POVEZAVE IN POŠILJANJE UKAZOV 36](#_Toc198038634)

[20 WHITELIST UPORABNIKOV PREKO FLASK APLIKACIJE 38](#_Toc198038635)

[21 VIZUALNA PRENOVA ADMIN APLIKACIJE 40](#_Toc198038636)

[22 ZAKLJUČEK 42](#_Toc198038637)

# UVOD

V današnjem času oblačne storitve omogočajo hitro, prilagodljivo in stroškovno učinkovito gostovanje različnih aplikacij. Amazon Web Services (AWS) ponuja zanesljivo infrastrukturo za vzpostavitev kompleksnih rešitev.

Namen te seminarske naloge je prikazati, kako s pomočjo AWS storitev načrtovati in postaviti produkcijsko okolje za Minecraft strežnik. Projekt vključuje zasnovo varnega omrežja, konfiguracijo strežnikov, avtomatizacijo varnostnih kopij in uporabniški spletni vmesnik za nadzor.

S tem projektom sva želela prikazati, kako se lahko z uporabo sodobnih oblačnih storitev in odprtokodnih orodij izdela popolnoma delujoča, modularna in varnostno ustrezna igralna platforma, ki presega zgolj tehnično vzpostavitev in vključuje tudi avtomatizacijo, CI/CD prakso in upravljanje uporabnikov.

## CILJI NALOGE

* Načrtovanje omrežne arhitekture v AWS (VPC, subnets, IGW, route tables).
* Ustvarjanje varnostnih skupin za zaščito strežnika.
* Zagon EC2 instanc: bastion host in glavni Minecraft strežnik.
* Dokaz uspešne implementacije prek konzolnega vmesnika.

# OPIS UPORABLJENE AWS INFRASTRUKTURE

Ustvarili smo glavno VPC z naslovnim prostorom 10.0.0.0/16 z imenom minecraft-vpc. Ta je osnova za nadaljnje podomrežja in varnostne nastavitve.

## **Podomrežja (subnets)**

* **minecraft-public-subnet**: javno podomrežje v AZ eu-central-1a (CIDR 10.0.1.0/24).
* **minecraft-private-subnet**: zasebno podomrežje v AZ eu-central-1b (CIDR 10.0.2.0/24).

## **Internet Gateway in Route Table**

Pripeli smo **Internet Gateway** (minecraft-igw) na VPC in v javni Route Table dodali pot 0.0.0.0/0 proti IGW. Zasebno podomrežje ni imelo izvoza v internet.

## **Varnostne skupine (Security Groups)**

* **sg-admin**: omogoča SSH (TCP/22) iz vašega IP naslova in HTTP (TCP/80) od koderkoli.
* **sg-minecraft**: odprti porti TCP/25565–25567 za Minecraft promet in SSH (TCP/22) iz vašega IP naslova.

## **Aurora RDS (MySQL)**

Za potrebe shranjevanja podatkov o uporabnikih smo vzpostavili Amazon Aurora z MySQL pogonom. Podatke za povezavo smo pridobili iz AWS konzole naslednje:

1. **Ustvarjanje Aurora RDS**
   * AWS konzola → RDS → Databases → Create database → Engine options: Amazon Aurora (MySQL compatible).
   * Izbira števila replik (1 instance v zasebnem subnetu).
   * Nastavitev master username **admin** in gesla (nastavljeno med kreacijo).
   * Konfiguracija omrežja: VPC **minecraft-vpc**, private subnet, brez javnega dostopa.
2. **Pridobitev povezovalnih podatkov**
   * Po uspešni kreaciji smo v RDS konzoli odprli podrobnosti baze in zabeležili:
     + **Endpoint** (npr. minecraft-db.cluster-xxxxxxxxxxxx.eu-central-1.rds.amazonaws.com)
     + **Port** (privzeto 3306)
     + **Master uporabnik**: admin
   * Ti podatki so shranjeni kot okoljske spremenljivke v Admin UI aplikaciji in skripti za generiranje whitelist.json.
3. **Varnost**
   * Varnostna skupina **sg-database** smo nastavili tako, da dovoljuje TCP/3306 iz EC2 instanc v zasebnem subnetu.

Ti koraki zagotavljajo, da imata tako spletna aplikacija kot skripte varen in zanesljiv dostop do baze podatkov za upravljanje uporabnikov.

# IMPLEMENTACIJSKI KORAKI

## **Zagon EC2 instanc**

1. **minecraft-admin** (bastion host)
   * AMI: Ubuntu 22.04
   * Tip: t3.micro
   * Subnet: minecraft-public-subnet (z javnim IP)
   * Varnostna skupina: sg-admin
2. **minecraft-server** (glavni strežnik)
   * AMI: Ubuntu 22.04
   * Tip: t3.large
   * Subnet: minecraft-private-subnet (brez javnega IP)
   * Varnostna skupina: sg-minecraft

Slika : Pregled AWS EC2 konzole z obema instancama v stanju Running

## **SSH povezava preko bastion hosta**

Za varno povezavo do Minecraft strežnika, ki se nahaja v zasebnem podomrežju brez javnega IP, smo uporabili instanco minecraft-admin kot bastion host. Postopek je potekal v naslednjih korakih:

1. **Priprava SSH ključa**
   * Na lokalnem računalniku smo privzeti PuTTY .ppk ključ pretvorili v OpenSSH .pem format s PuTTYgen:
     + Odprli smo PuTTYgen, naložili datoteko .ppk in izbrali **Conversions → Export OpenSSH key**, ter jo shranili kot minecraft-key.pem.
2. **Prenos .pem na bastion host**
   * Povezali smo se na minecraft-admin z uporabo PuTTY in .ppk ključa.
   * Preko WinSCP smo prenesli minecraft-key.pem v domači imenik /home/ubuntu/ na admin instanci.
   * Na admin instanci smo popravili pravice ključa:

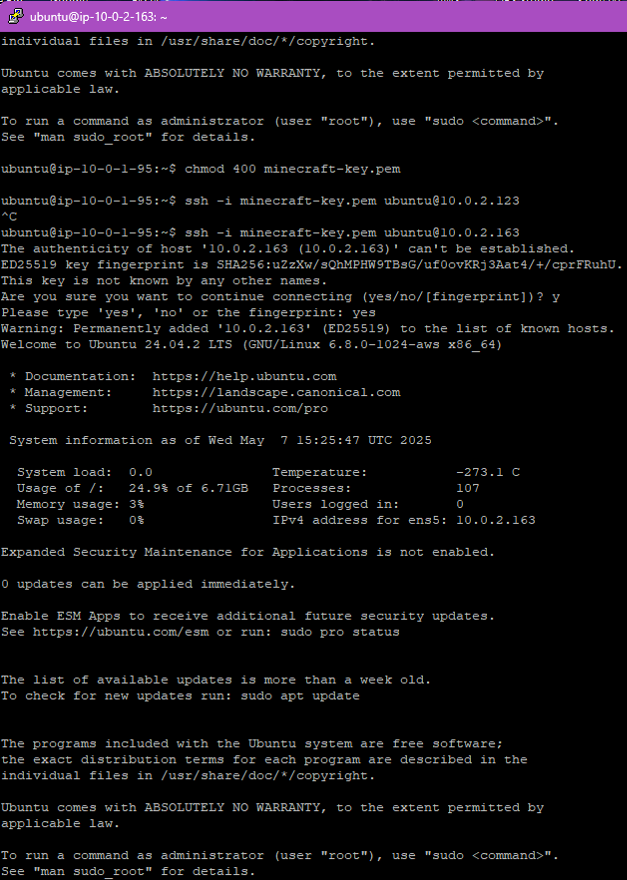
chmod 400 minecraft-key.pem

1. **Povezava na Minecraft strežnik**
   * V AWS konzoli smo preverili zasebni IP minecraft-server (npr. 10.0.2.163).
   * Na admin instanci smo izvedli ukaz:

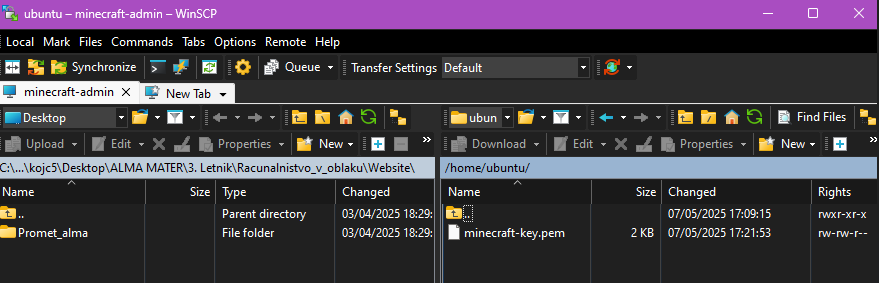
ssh -i minecraft-key.pem ubuntu@10.0.2.163

* + Uspešna povezava je prikazana na sliki spodaj:  
    ![SSH povezava preko bastion hosta](/mnt/data/povezava na minecraft-server preko minecraft-admin serverja.png)

1. **Potrdilo prenosa ključa**
   * V WinSCP se vidi datoteka minecraft-key.pem v /home/ubuntu/, kot dokaz prenosa:  
     ![Prenos PEM ključa](/mnt/data/nalaganje pem na minecraft-admin server.png)

****

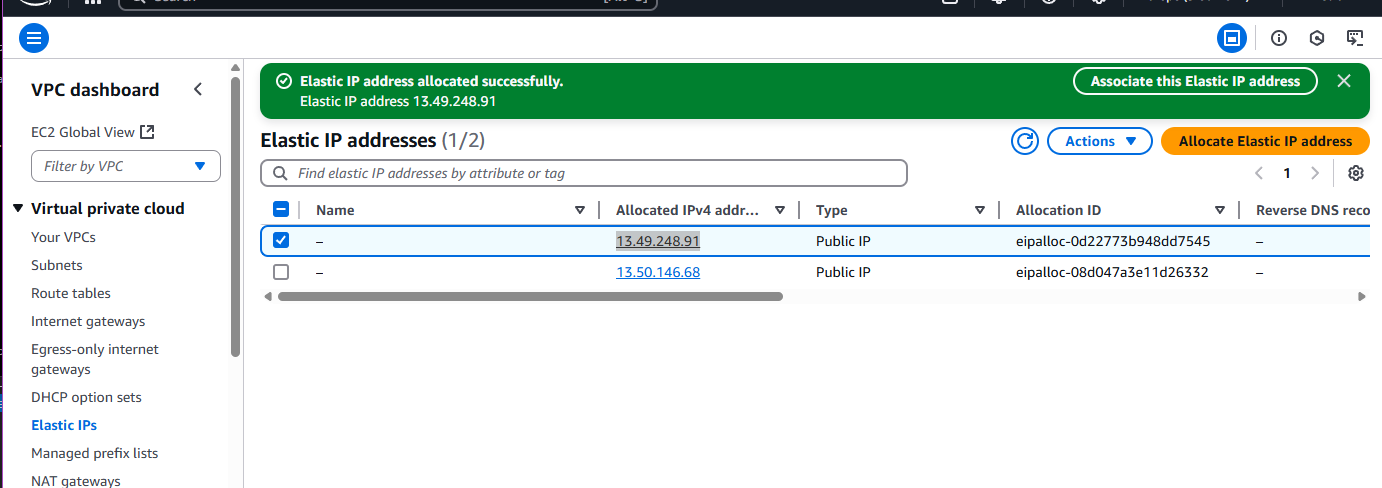
Slika : Uspešna SSH povezava na minecraft-server preko bastion hosta

****

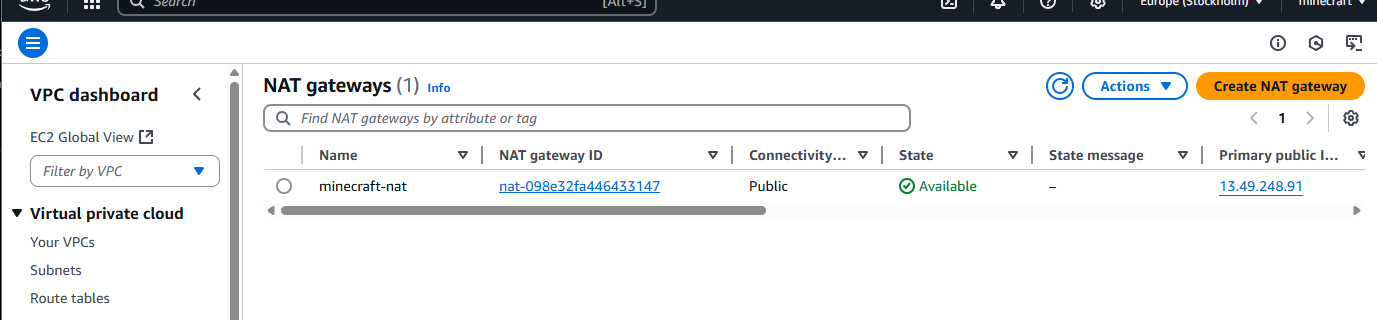
Slika : Prenos PEM ključa prek WinSCP na bastion host

## **Omogočanje internetnega dostopa za posodobitve na MC server**

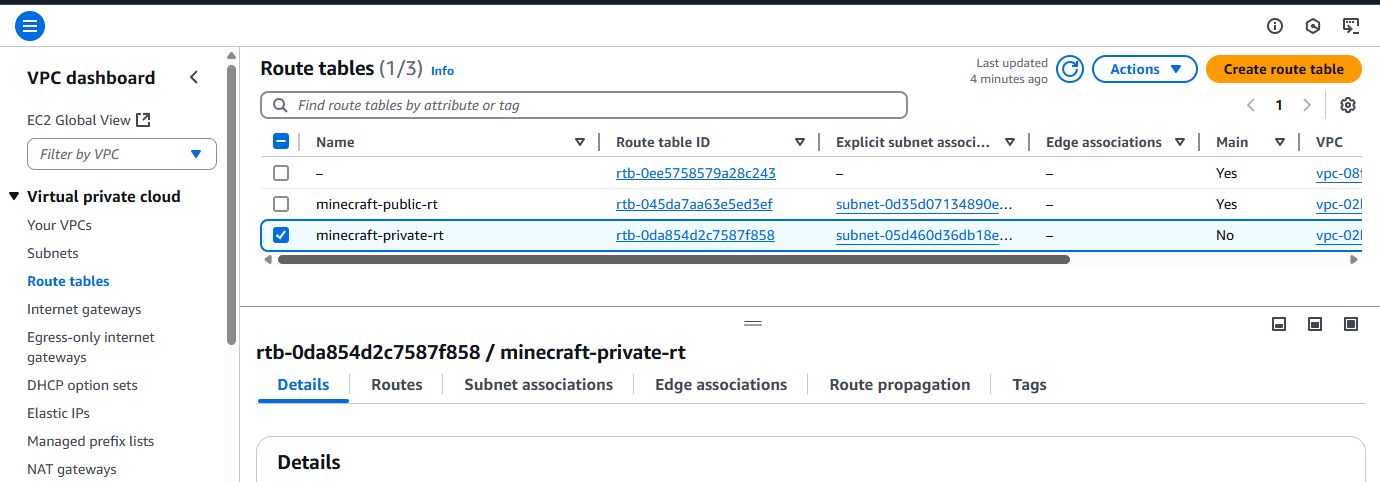
Ker minecraft-server deluje v zasebnem subnetu brez javnega IP, nima neposrednega dostopa do interneta, kar preprečuje namestitev paketov in posodobitev. Za rešitev smo izvedli naslednje korake:

1. **Alokacija Elastic IP naslova**
   * V AWS konzoli: VPC → Elastic IPs → Allocate Elastic IP address.
   * Zabeležili smo nov Elastic IP: **13.49.248.91**.
2. 

Slika : Alokacija Elastic IP naslova (13.49.248.91)

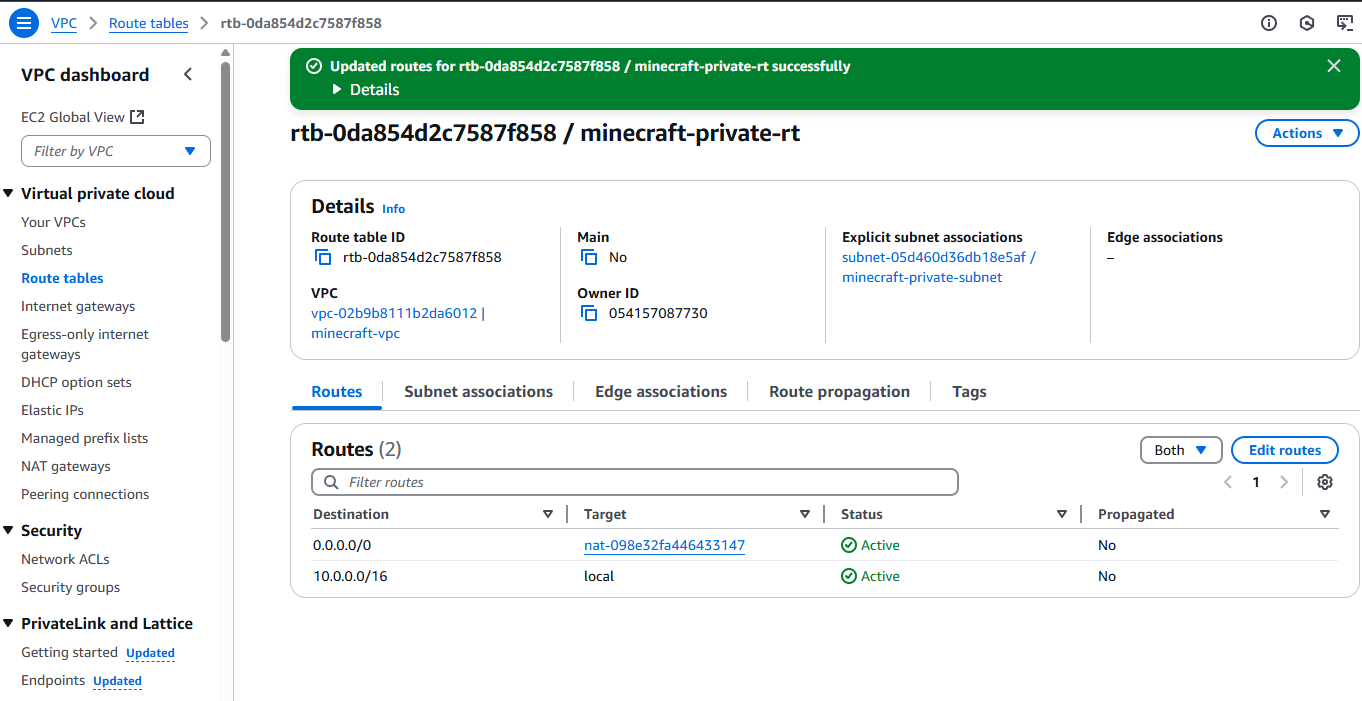
1. **Ustvarjanje NAT Gateway**
   * V AWS konzoli: VPC → NAT Gateways → Create NAT Gateway.
     + Ime: minecraft-nat
     + Subnet: minecraft-public-subnet
     + Elastic IP: **13.49.248.91**
   * NAT Gateway je bil uspešno vzpostavljen.

Slika : Ustvarjanje NAT Gateway z imenom minecraft-nat

1. **Ustvarjanje ločene route table za private subnet**
   * V AWS konzoli: VPC → Route Tables → Create route table.
     + Ime: minecraft-private-rt
     + VPC: minecraft-vpc
   * Subnet associations → Edit subnet associations → označili minecraft-private-subnet.
2. 

Slika : Ustvarjanje ločene route table za zasebno podomrežje

1. **Dodajanje poti prek NAT Gateway**
   * V zavihku Routes za minecraft-private-rt smo uredili poti:
     + Destination: 0.0.0.0/0
     + Target: nat-098e32fa446433147 (Minecraft NAT Gateway).



Slika : Dodajanje poti prek NAT Gateway v minecraft-private-rt

Po 1–2 minutah je minecraft-server pridobil dostop do interneta, kar smo preverili z ukazom:

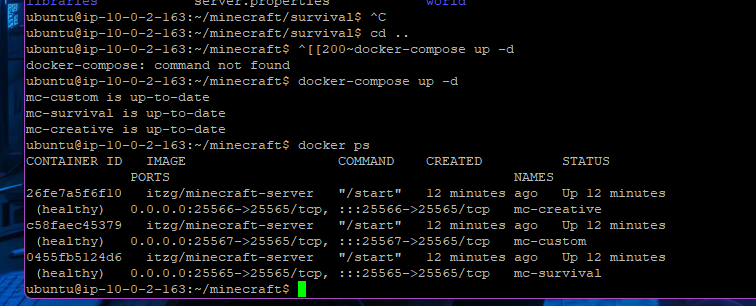
ssh -i minecraft-key.pem ubuntu@10.0.2.163

sudo apt update && sudo apt upgrade -y

Uspešno je bilo izveden apt update brez napak, kar dokazuje dostop do repozitorijev.

## **Namestitev Docker in zagon MC strežnikov v kontejnerjih**

Za avtomatizacijo zagona več različnih serverjev smo uporabili Docker in Docker Compose.



Slika : Izpis zagona Docker kontejnerjev (docker ps rezultat)

Koraki so bili naslednji:

**Namestitev Docker**

*sudo apt update*

*sudo apt install -y docker.io docker-compose-plugin*

*sudo systemctl enable --now docker*

**Priprava direktorijev za strežnike**

*mkdir -p ~/minecraft/survival*

*mkdir -p ~/minecraft/creative*

*mkdir -p ~/minecraft/custom*

*cd ~/minecraft*

**Ustvarjanje docker-compose.yml**

*version: "3"*

*services:*

*survival:*

*image: itzg/minecraft-server*

*container\_name: mc-survival*

*ports:*

*- "25565:25565"*

*environment:*

*EULA: "TRUE"*

*volumes:*

*- ./survival:/data*

*restart: unless-stopped*

*creative:*

*image: itzg/minecraft-server*

*container\_name: mc-creative*

*ports:*

*- "25566:25565"*

*environment:*

*EULA: "TRUE"*

*volumes:*

*- ./creative:/data*

*restart: unless-stopped*

*custom:*

*image: itzg/minecraft-server*

*container\_name: mc-custom*

*ports:*

*- "25567:25565"*

*environment:*

*EULA: "TRUE"*

*volumes:*

*- ./custom:/data*

*restart: unless-stopped*

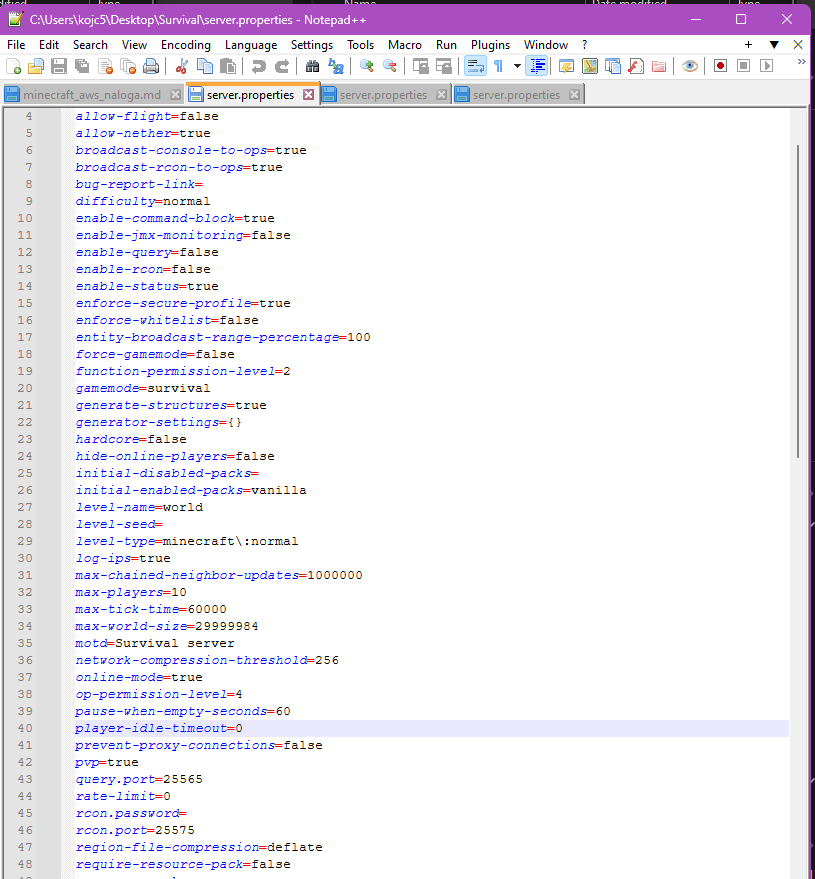
**Zagon kontejnerjev**

*docker compose up -d*

**Preverjanje zagona**

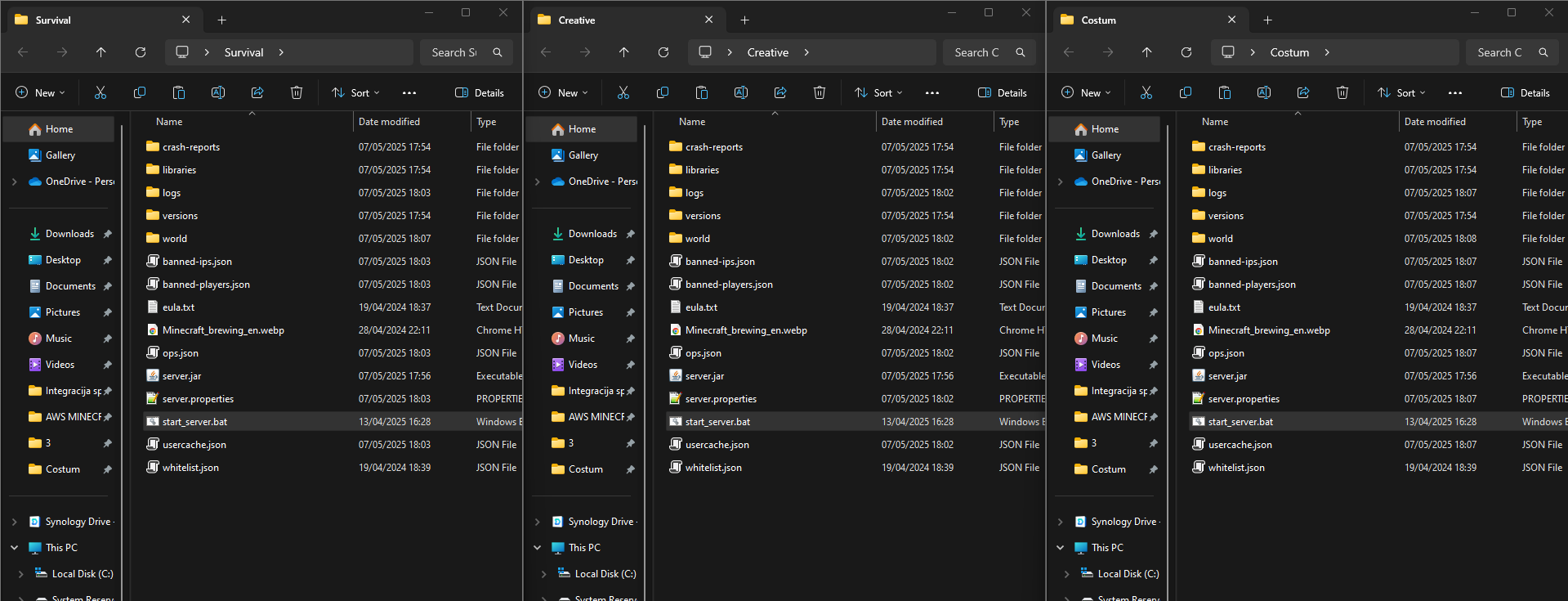
*docker ps*

### Konfiguracija strežniških lastnosti



Slika : Vsebina datoteke server.properties

### Oblikovanje direktorijev in volumnov



Slika : Explorer pogled na mape survival, creative in custom, s podmapi (world, logs, ops.json, whitelist.json …) in start\_server.bat

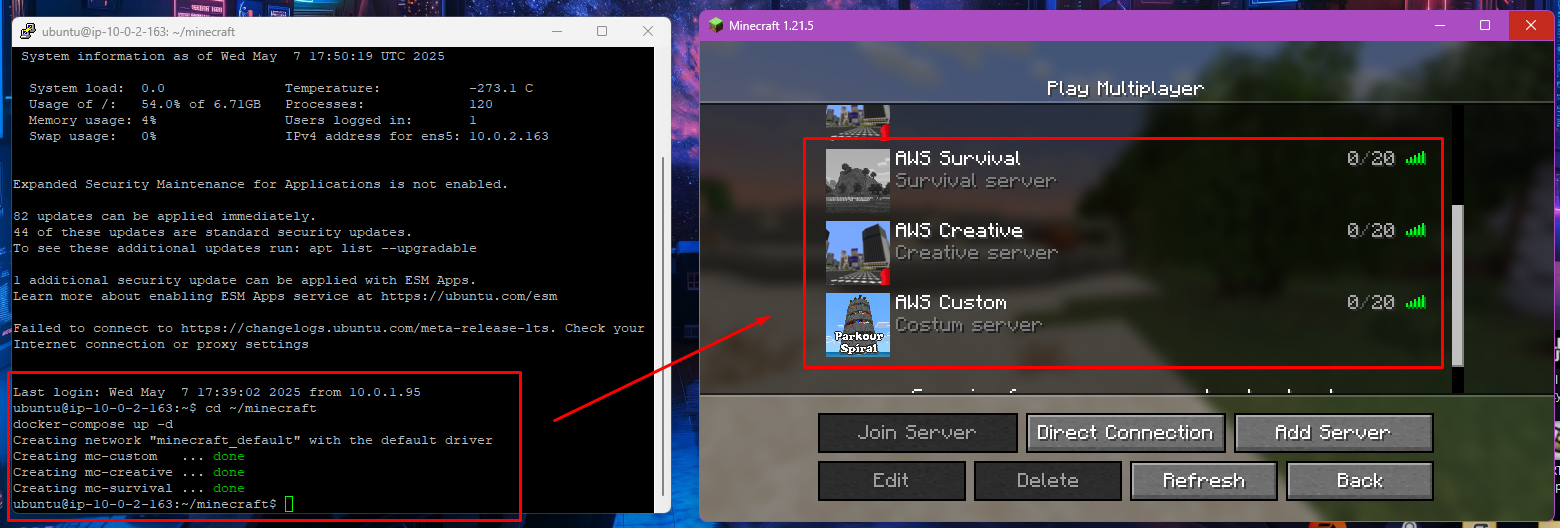
# POVEZAVA IN TEST STREŽNIKOV

Po zagonu Docker kontejnerjev smo strežnike preizkusili z vnosom javnega IP naslova instance in ustreznih portov v Minecraft klientu:

|  |  |
| --- | --- |
| **Server** | **IP naslov in port** |
| Survival | 13.50.146.68:25565 |
| Creative | 13.50.146.68:25566 |
| Custom | 13.50.146.68:25567 |

Uspešna povezava je vidna na sliki:



Slika : Povezava na Minecraft strežnike v odjemalcu

Slika : Konzolni izpis uspešnega zagona (docker-compose up -d) in seznam dostopnih strežnikov v Minecraft klientu.

# POTRDITEV DELOVANJA

Na Sliki 1 je prikazan AWS EC2 konzola, kjer vidimo instanci **minecraft-admin** (t3.micro) in **minecraft-server** (t3.large) v regiji EU North-1. Poleg tega je na Sliki 2 razvidno, da so Docker kontejnerji uspešno zagnani in strežniki dostopni preko javnega IP naslova. Dokaz implementacije Dokaz implementacije Na Sliki 1 je prikazan AWS EC2 konzola, kjer vidimo instanci **minecraft-admin** (t3.micro) in **minecraft-server** (t3.large) v regiji EU North-1. To dokazuje uspešen zagon in pravilno dodeljene nastavitve.

# NAMESTITEV AWS CLI

Po zaključeni konfiguraciji strežnika je bil naslednji korak namestitev orodja AWS CLI (Command Line Interface), ki omogoča neposredno upravljanje AWS storitev iz ukazne vrstice.

Na instanci minecraft-server z operacijskim sistemom Ubuntu 24.04 smo najprej poskusili namestitev s standardnim ukazom:

sudo apt update

sudo apt install awscli -y

Vendar je ta ukaz povzročil napako, saj je bila podpora za awscli v standardnem APT repozitoriju odstranjena:

E: Package 'awscli' has no installation candidate

Zato smo uporabili uradno priporočeno metodo za namestitev AWS CLI v2:

**1. Prenos arhive:**

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

**2. Namestitev programa unzip (če še ni nameščen):**

sudo apt install unzip -y

**3. Razpakiranje arhive:**

unzip awscliv2.zip

**4. Namestitev AWS CLI:**

sudo ./aws/install

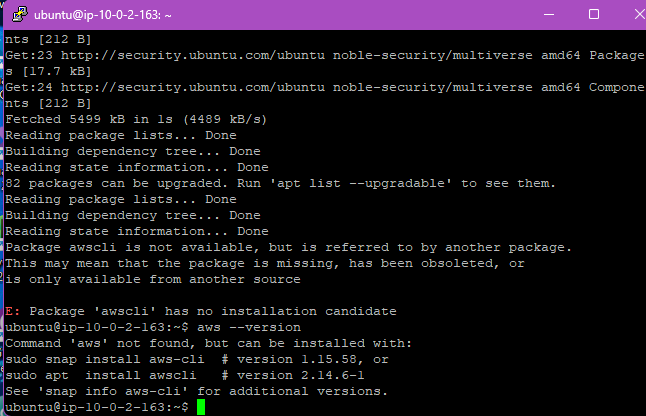
**5. Preverjanje namestitve:**

aws --version

Uspešna namestitev je vrnila izpis podobno naslednjemu:

aws-cli/2.27.11 Python/3.13.3 Linux/6.8.0-1024-aws exe/x86\_64.ubuntu.24

Na sliki spodaj je prikazan neuspešen poskus namestitve preko APT in sistemski odziv:



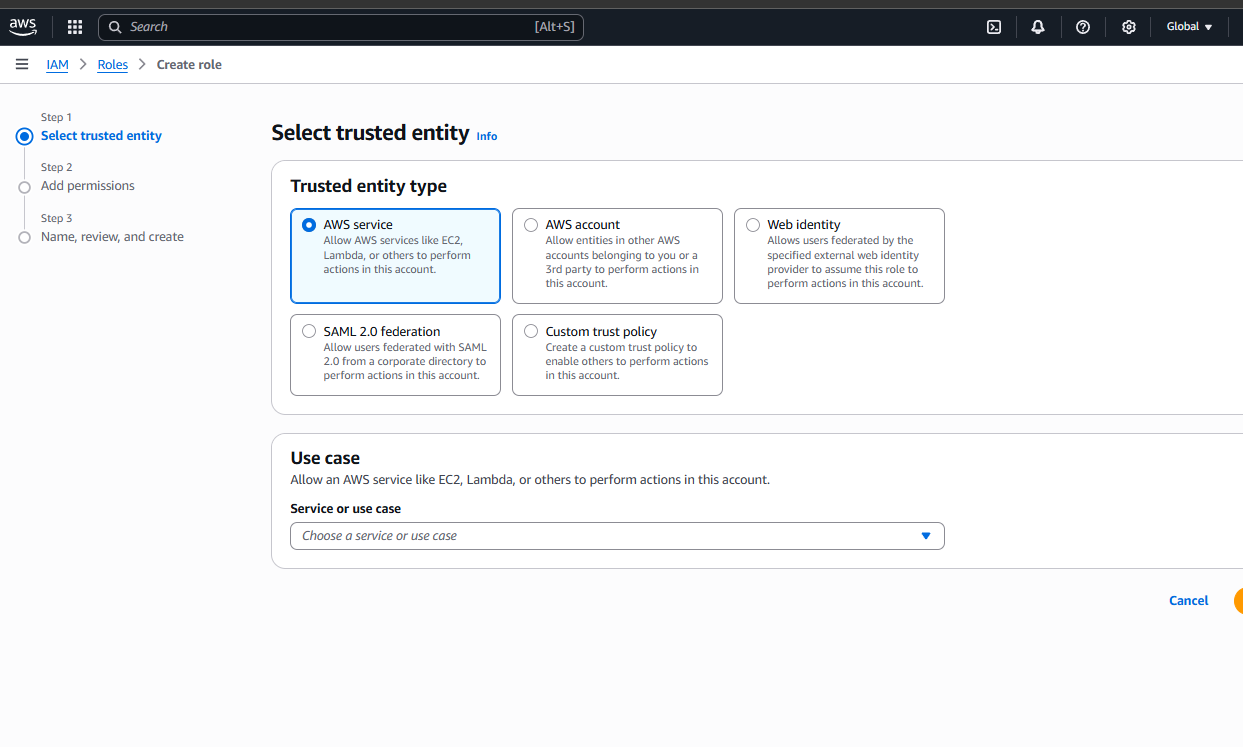
Ta korak je ključen za nadaljnjo avtomatizacijo postopkov varnostnega kopiranja preko AWS CLI.

# DODELITEV IAM VLOGE EC2 INSTANCI

Za izvajanje varnostnih kopij neposredno iz EC2 instance v S3 bucket je bilo potrebno instanci dodeliti IAM vlogo, ki dovoljuje dostop do storitve S3.

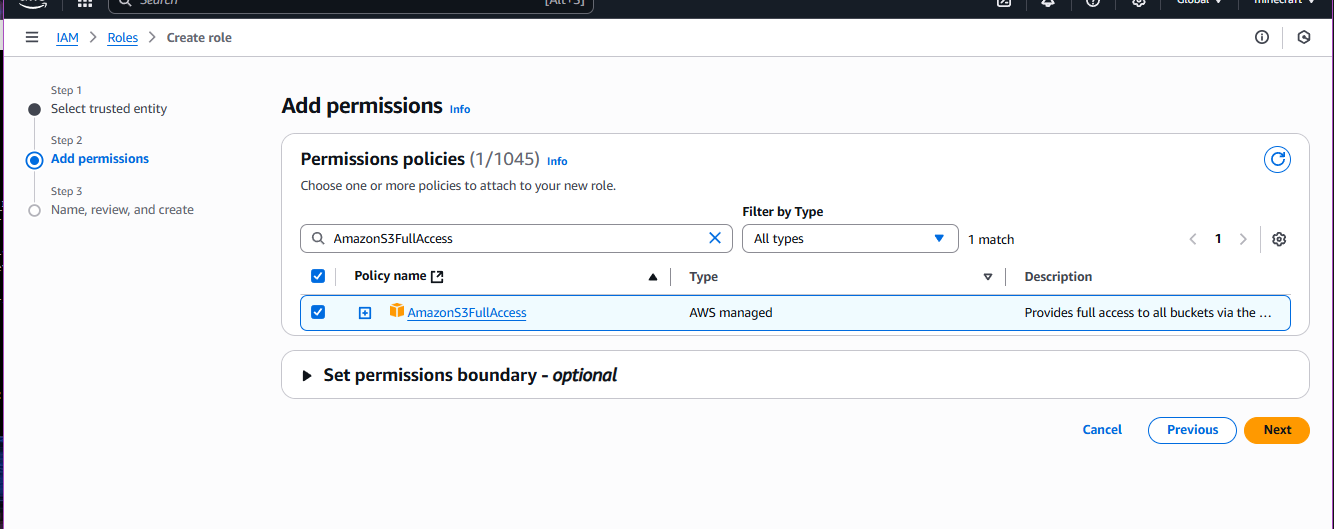
**KORAK 1: Ustvarjanje IAM vloge**

1. V AWS konzoli smo odprli: **IAM → Roles → Create role**.
2. Izbrali smo:
   * *Trusted entity type:* AWS service
   * *Use case:* EC2



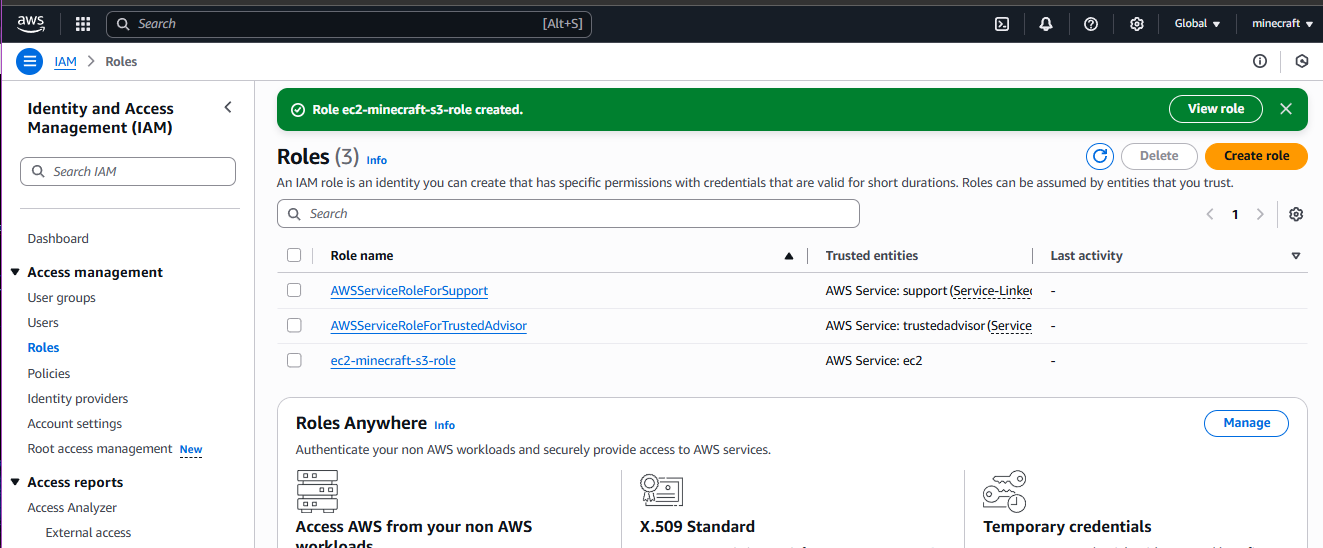
Slika : Izbira zaupanja vredne entitete (trusted entity) med ustvarjanjem nove IAM vloge za EC2 instanco.

1. V naslednjem koraku smo poiskali in označili **AmazonS3FullAccess** politiko.



Slika : Dodeljevanje pravic vlogi – izbrana je sistemska politika AmazonS3FullAccess, ki omogoča dostop do vseh S3 storitev.

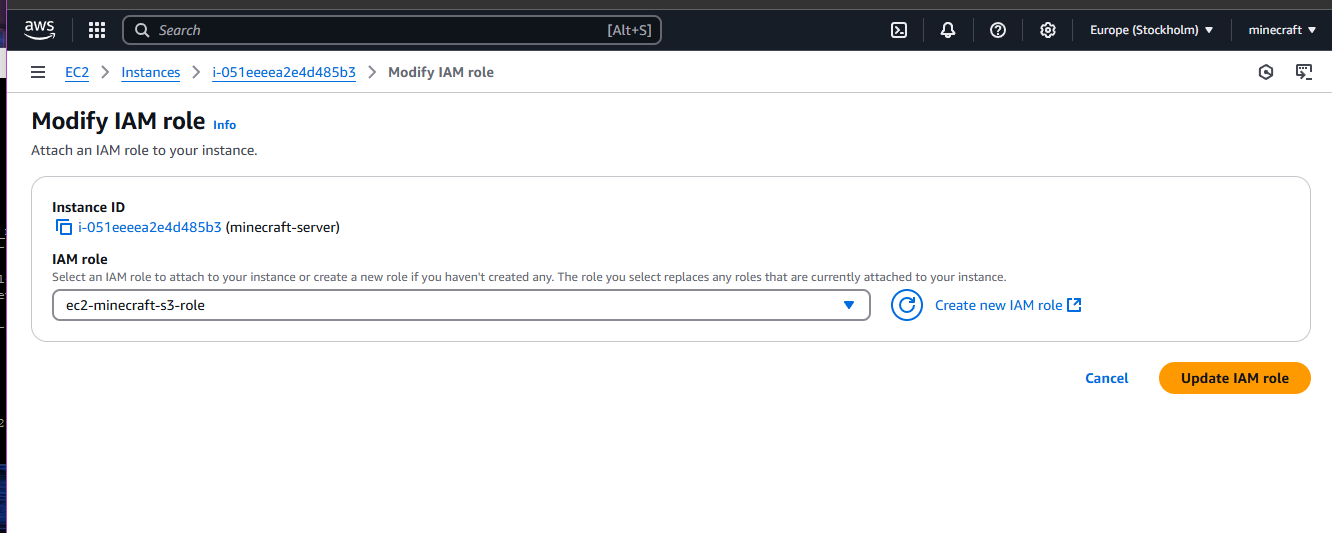
1. Poimenovali smo vlogo kot **ec2-minecraft-s3-role** in kliknili **Create role**.



Slika : Pregled ustvarjenih IAM vlog – uspešno ustvarjena vloga ec2-minecraft-s3-role je prikazana v seznamu.

**KORAK 2: Povezava IAM vloge z EC2 instanco**

1. Odšli smo v **EC2 → Instances**, označili instanco z imenom *minecraft-server*.
2. Izbrali **Actions → Security → Modify IAM Role**.
3. V meniju smo izbrali pravkar ustvarjeno vlogo **ec2-minecraft-s3-role**.
4. Potrdili z **Update IAM Role**.



Slika : Dodeljevanje IAM vloge EC2 instanci – izbrana je bila vloga ec2-minecraft-s3-role za instanco minecraft-server.

Z dodeljeno vlogo ima instanca zdaj pravico nalaganja datotek v S3 bucket brez potrebe po ročni konfiguraciji dostopnih ključev (kar je varnejše in bolj skladno s priporočili AWS).

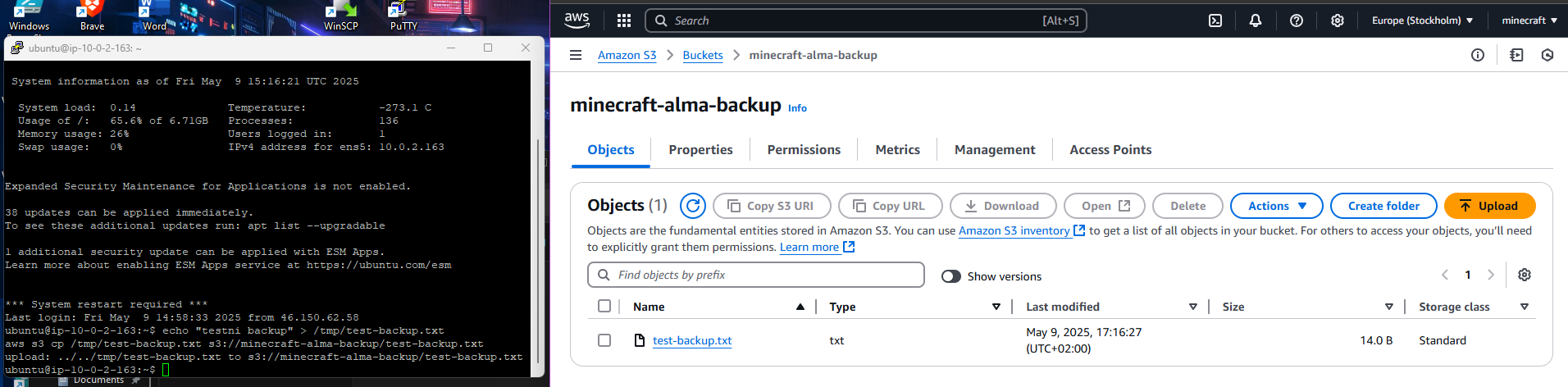
Ta korak omogoča uporabo ukaza aws s3 cp neposredno iz ukazne vrstice na EC2 instanci.

# AVTOMATIZACIJA BACKUPOV PRKEO LAMBDA FUNKCIJE

V tem koraku smo želeli doseči, da se skripta za varnostno kopiranje strežniških map (survival, creative, custom) zažene samodejno vsak dan ob 23:00. Namesto uporabe lokalnega crona na strežniku smo uporabili AWS-native pristop:

* EventBridge pravilo vsak dan ob 23:00
* sproži Lambda funkcijo
* ki preko SSM (AWS Systems Manager) zažene skripto backup-to-s3.sh na EC2 instanci

**1. Test delovanja AWS CLI in nalaganja na S3**



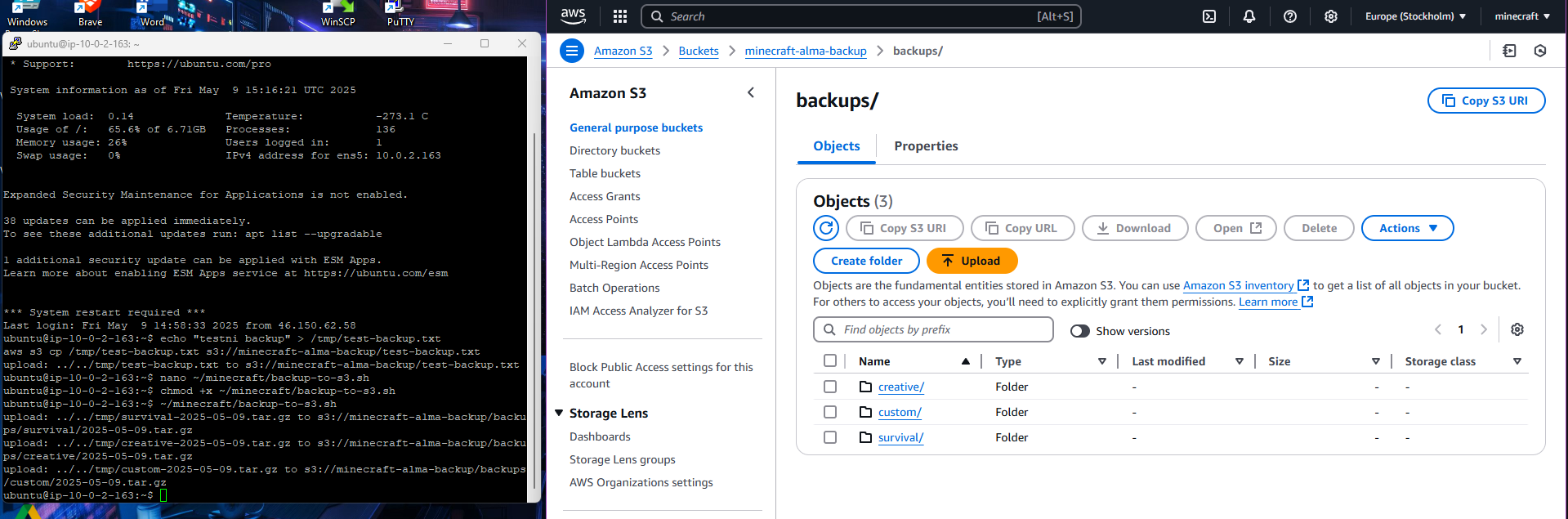
Slika : Preverjanje, ali EC2 instanca lahko naloži datoteko v S3 preko AWS CLI (test-backup.txt datoteka).

**2. Ustvarjena skripta za varnostno kopijo**



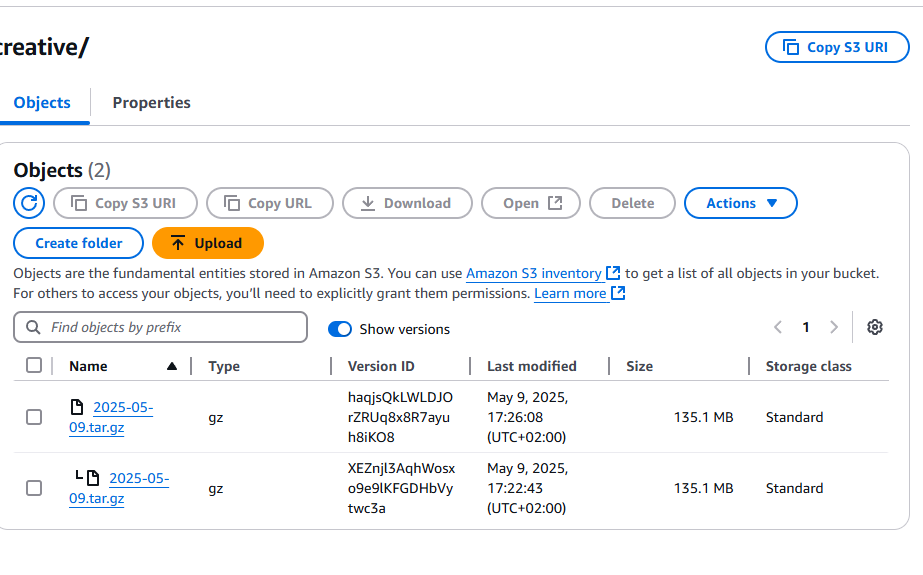
Slika : Vsebina skripte backup-to-s3.sh, ki arhivira posamezne strežniške direktorije in jih naloži v S3 bucket.

1. **Ročni test skripte**



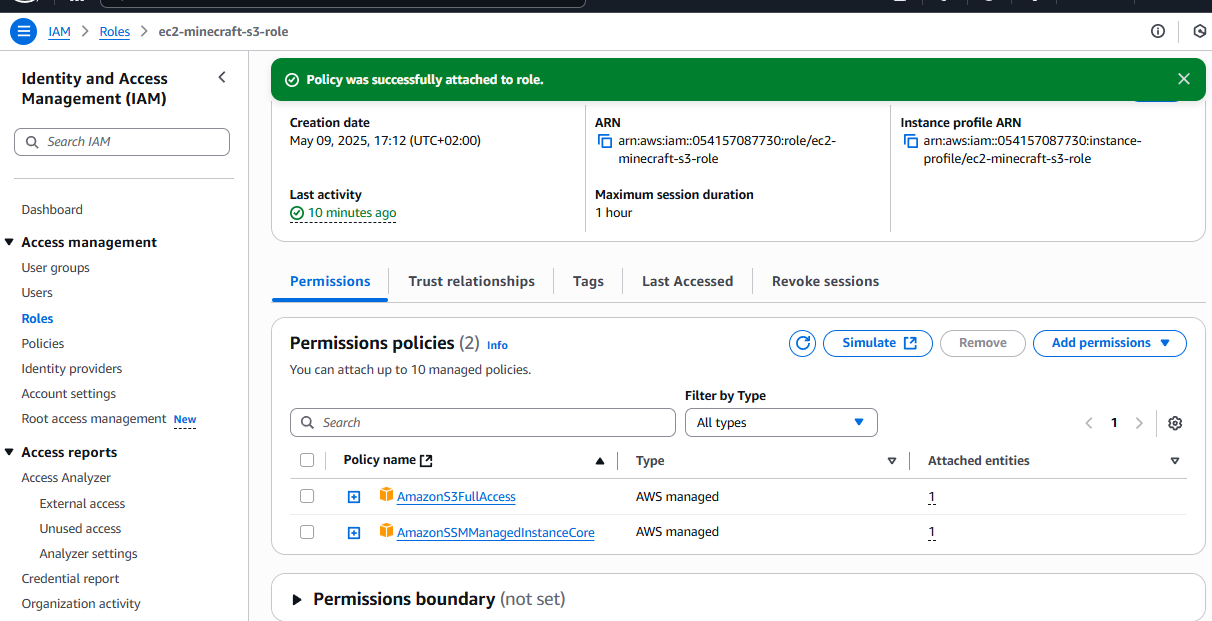
Slika : Rezultat ročne izvedbe skripte in uspešno kopiranje .tar.gz arhivov v S3.

**4. Pregled arhiviranih datotek z vklopljenim verzioniranjem**



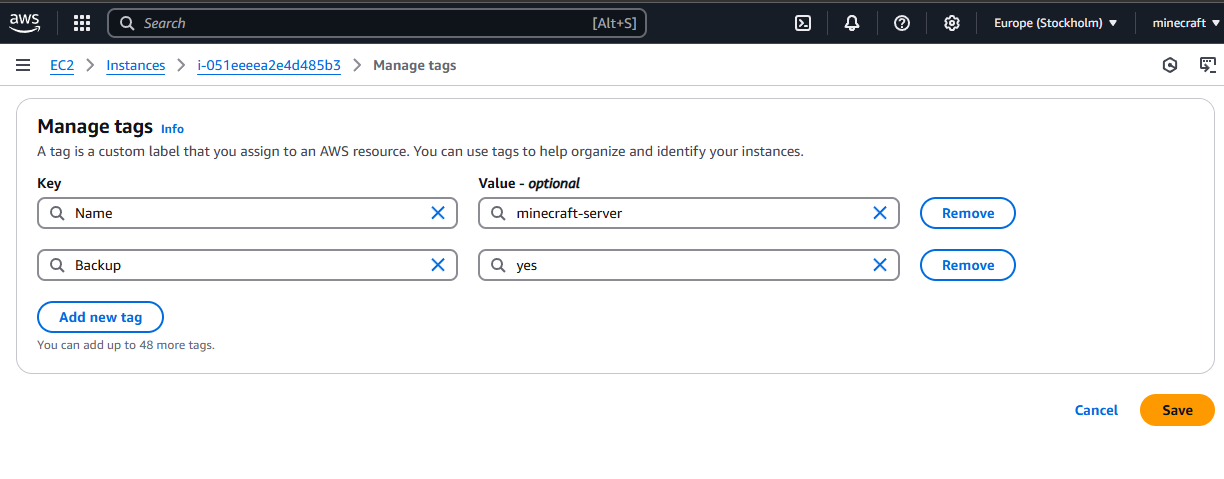
Slika : V S3 bucketu so vidne datoteke, ki vsebujejo različne dnevne verzije varnostnih kopij svetov.

**5. IAM vloga posodobljena za SSM dostop**



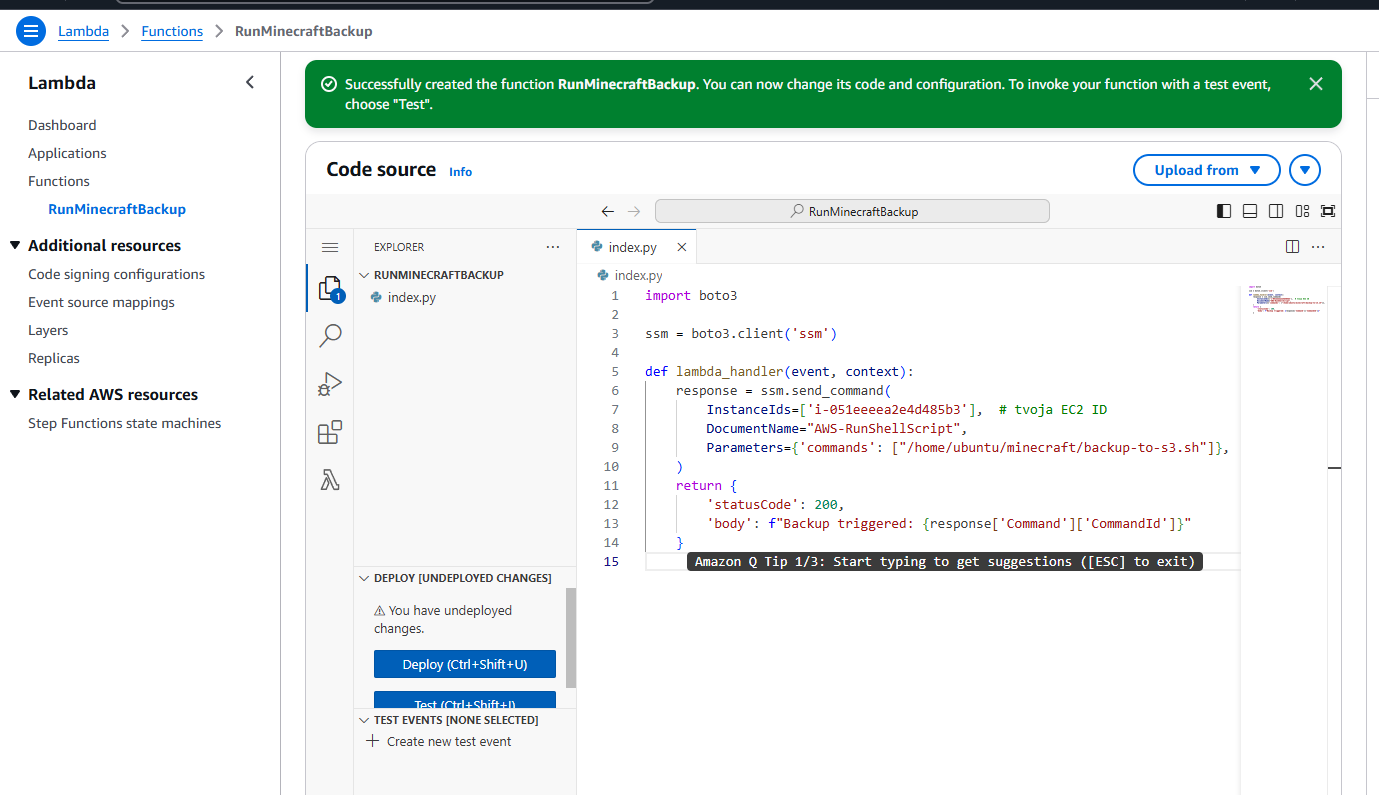
Slika : IAM vlogi ec2-minecraft-s3-role so bile dodane dodatne pravice (AmazonSSMManagedInstanceCore), kar omogoča upravljanje z EC2 preko SSM.

**6. Dodani oznaki (tags) na instanci za lažje upravljanje**



Slika : Instanci minecraft-server sta bili dodani oznaki (Name: minecraft-server, Backup: yes), kar omogoča enostavno filtriranje.

**7. Lambda funkcija zaženeta SSM ukaz na EC2**

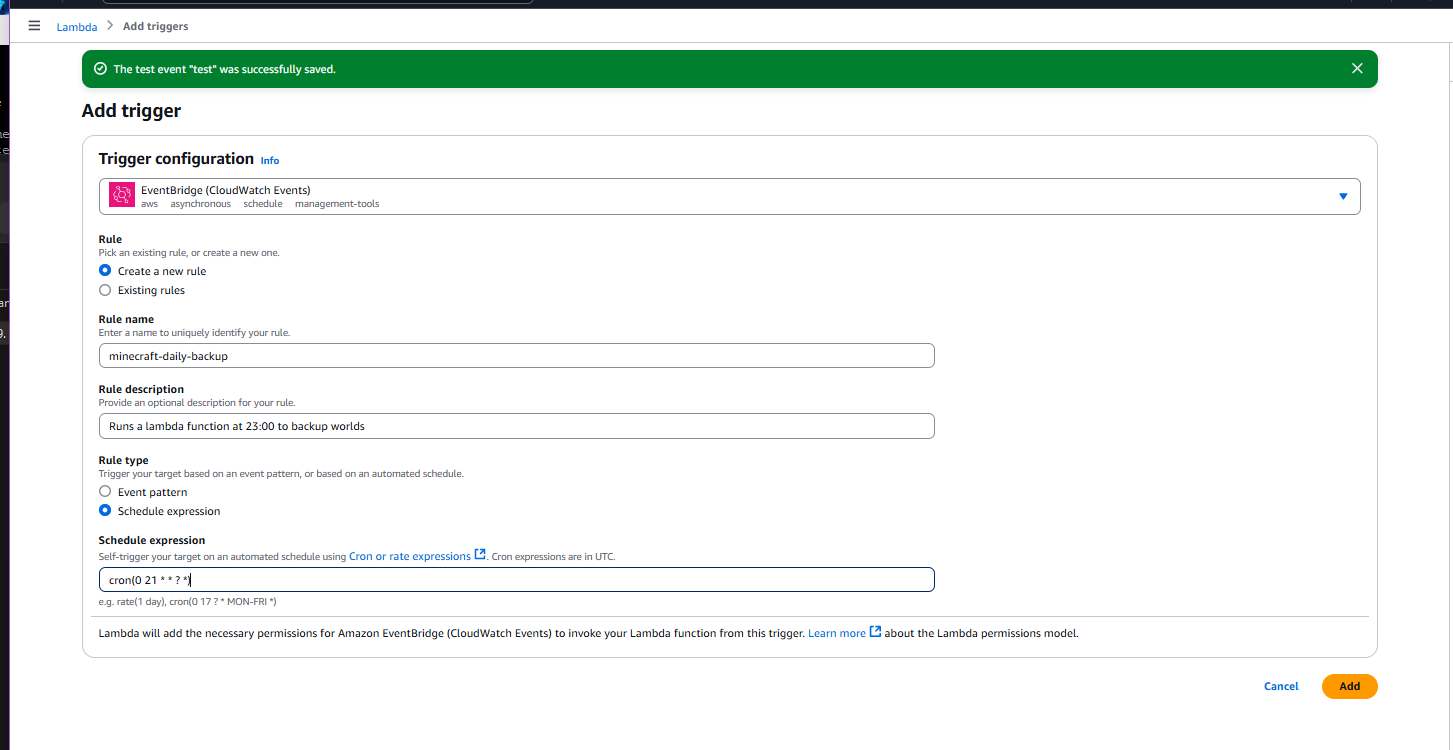


Slika : Koda Lambda funkcije RunMinecraftBackup, ki zažene skripto backup-to-s3.sh preko SSM z uporabo boto3.

# USTVARJANJE EVENTBRIDGE ČASOVNIKA ZA BACKUP

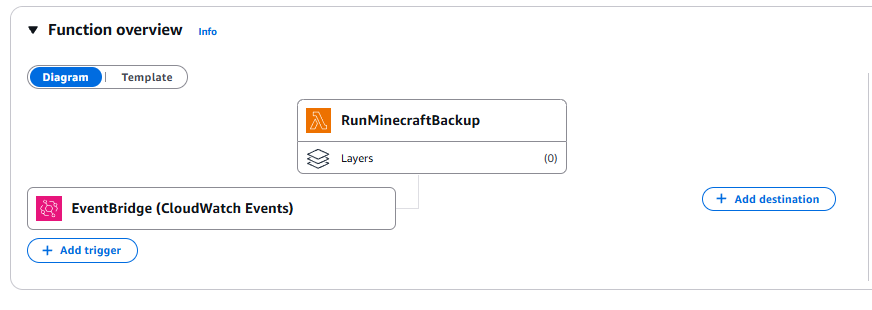
Za zagotavljanje vsakodnevne avtomatizacije izvajanja Lambda funkcije RunMinecraftBackup, smo uporabili storitev **Amazon EventBridge**, ki omogoča časovno prožene dogodke. Cilj je bil nastaviti pravilo, ki vsak dan ob 23:00 samodejno sproži backup vseh treh svetov (survival, creative, custom).

**1. Ustvarjanje novega pravila (rule) z uporabo Cron izraza**



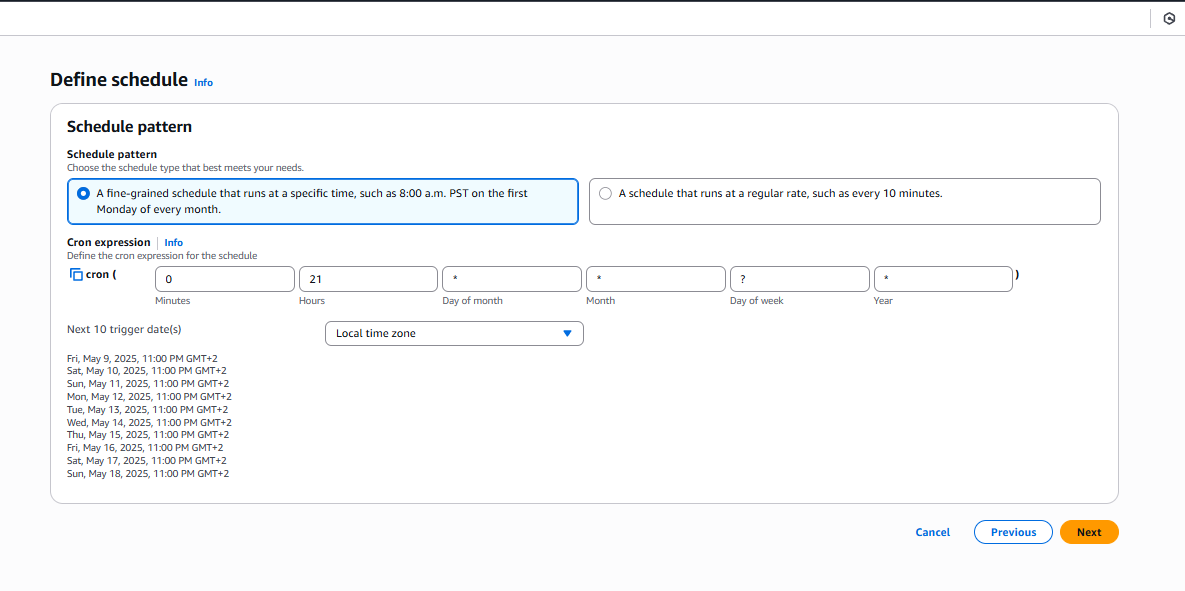
Slika : Nastavitev novega EventBridge pravila z imenom minecraft-daily-backup, ki vsak dan ob 23:00 UTC+2 sproži Lambda funkcijo.

**2. Pregled funkcijske povezave med EventBridge in Lambda funkcijo**



Slika : Pregled povezave – EventBridge sproži funkcijo RunMinecraftBackup vsakokrat, ko se pravilo izvede.

**3. Določitev časovnega razporeda z uporabo cron izraza**

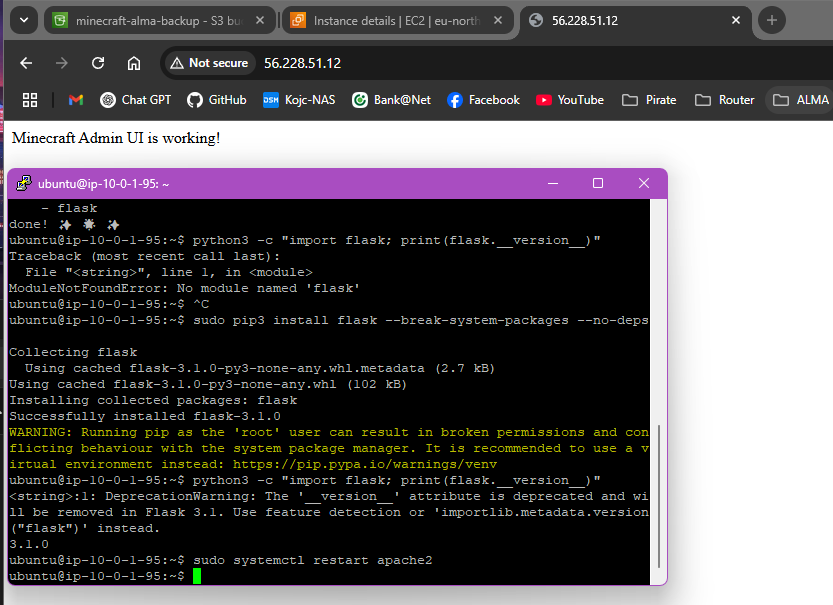


Slika : Definicija časovnega razporeda s cron izrazom cron(0 21 \* \* ? \*), ki pomeni 21:00 UTC (oz. 23:00 lokalno).

# NAMESTITEV FLASK APLIKACIJE NA APACHE STREŽNIK

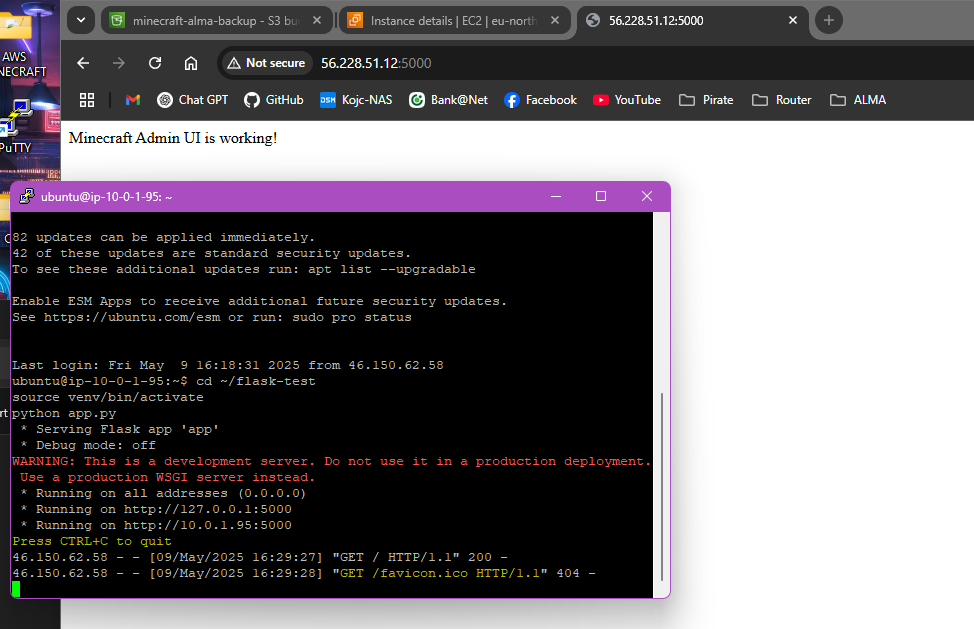
Za prikaz statusa in nadzora nad strežnikom Minecraft smo ustvarili osnovno Flask aplikacijo, ki se gostuje na isti EC2 instanci kot administrativni vmesnik. Flask je bil integriran z Apache strežnikom z uporabo mod\_wsgi modula.

**1. Namestitev knjižnice Flask z uporabo pip**



Slika : Namestitev knjižnice Flask in preverjanje njene različice prek ukazne vrstice.

**2. Zagon aplikacije v razvojnem načinu**



Slika : Flask aplikacija se zažene na lokalnem naslovu 0.0.0.0:5000 in postreže testno stran.

**3. Vključitev aplikacije v Apache s pomočjo mod\_wsgi**

Za produkcijsko delovanje aplikacije smo ustvarili .wsgi datoteko in ustrezno nastavili Apache konfiguracijo z WSGIDaemonProcess, WSGIScriptAlias in konfiguracijo virtualnega gostitelja (VirtualHost). To omogoča, da je Flask aplikacija dostopna prek standardnega HTTP porta (80), brez dodatnih nastavitev s strani uporabnika.

# FUNKCIONALNOSTI ADMIN UI APLIKACIJE

Flask aplikacija, nameščena na instanci minecraft-admin, deluje kot centralni uporabniški vmesnik za upravljanje platforme. Aplikacija ponuja naslednje funkcionalnosti:

* **Pregled in upravljanje whitelist**: Vmesnik pridobi seznam dovoljenih igralcev iz baze Aurora RDS in omogoča dodajanje novih.
* **Zagon strežnikov na zahtevo**: Administrator lahko preko gumba sproži HTTP zahtevek, ki aktivira AWS Lambda funkcijo za vklop glavne EC2 instance z Minecraft strežniki.

Komunikacija poteka preko:

* **Aurora RDS** za dostop do podatkov uporabnikov,
* **EC2 SSH (v prihodnosti GitHub Actions)** za posodobitev aplikacije.

# SAMODEJNI IZKLOP STREŽNIKA OB NEDEJAVNOSTI

Da bi dodatno zmanjšali stroške, je bila načrtovana funkcionalnost, ki bi samodejno izklopila Minecraft strežnik, če na njem ni igralcev.

Predvidena logika vključuje:

* periodično izvajanje skripte z ukazom mcstatus za pridobivanje števila aktivnih igralcev,
* beleženje zadnje aktivnosti igralcev,
* v primeru, da ni povezav dlje od 30 minut, se sproži aws ec2 stop-instances za izklop instance.

Čeprav funkcionalnost še ni implementirana, predstavlja pomemben korak v smeri avtomatskega upravljanja virov in jo je možno dopolniti s pomočjo cron opravil ali EventBridge pravil.

# AVTOMATIZIRANO NAMEŠČANJE APLIKACIJE PREK GITHUB ACTIONS

Za nadzor nad spremembami Flask aplikacije je predvidena uporaba sistema za nadzor različic GitHub. Koda je nameščena v zasebnem repozitoriju, kjer je konfiguriran CI/CD workflow z GitHub Actions.

Ob vsakem **push-u na vejo main**:

* se sproži GitHub Action,
* poveže se z EC2 instanco prek SSH,
* potegne najnovejšo kodo,
* ponovno zažene Flask aplikacijo z ukazom systemctl restart flask-admin.service.

name: Deploy Flask App

on:

push:

branches: [main]

jobs:

deploy:

runs-on: ubuntu-latest

steps:

- name: Checkout repo

uses: actions/checkout@v2

- name: Deploy to EC2

uses: appleboy/ssh-action@master

with:

host: ${{ secrets.EC2\_HOST }}

username: ubuntu

key: ${{ secrets.EC2\_SSH\_KEY }}

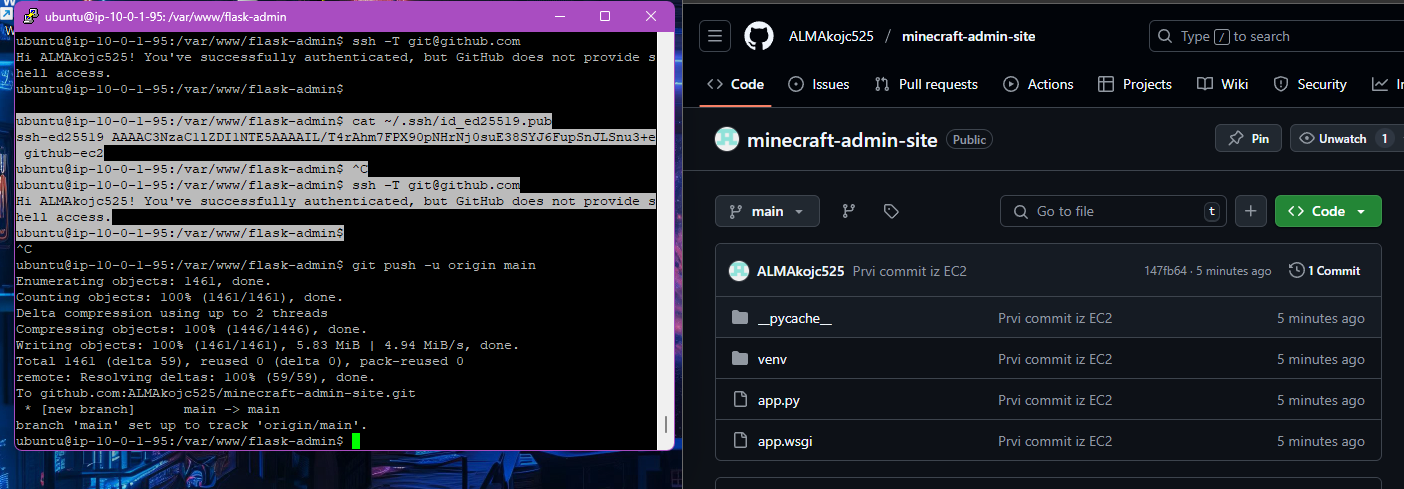
script: |

cd /home/ubuntu/flask-admin

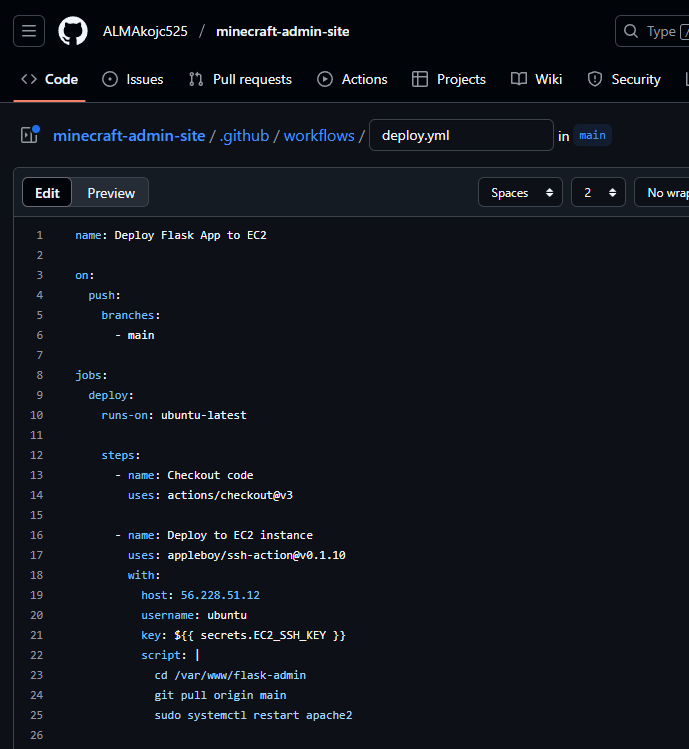
git pull origin main

sudo systemctl restart flask-admin.service

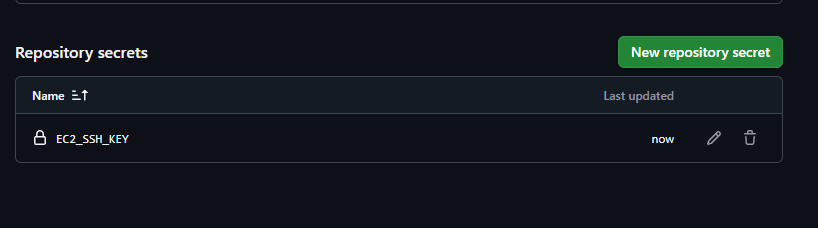
# DEPLOYMENT APLIKACIJE Z GITHUB REPOZITORIJA

V tej fazi projekta je bil predstavljen celoten postopek za avtomatsko nalaganje aplikacije iz GitHub repozitorija na EC2 instanco. S tem zagotavljamo, da so vse spremembe v kodi hitro in brez ročnega posredovanja prenesene v produkcijsko okolje.

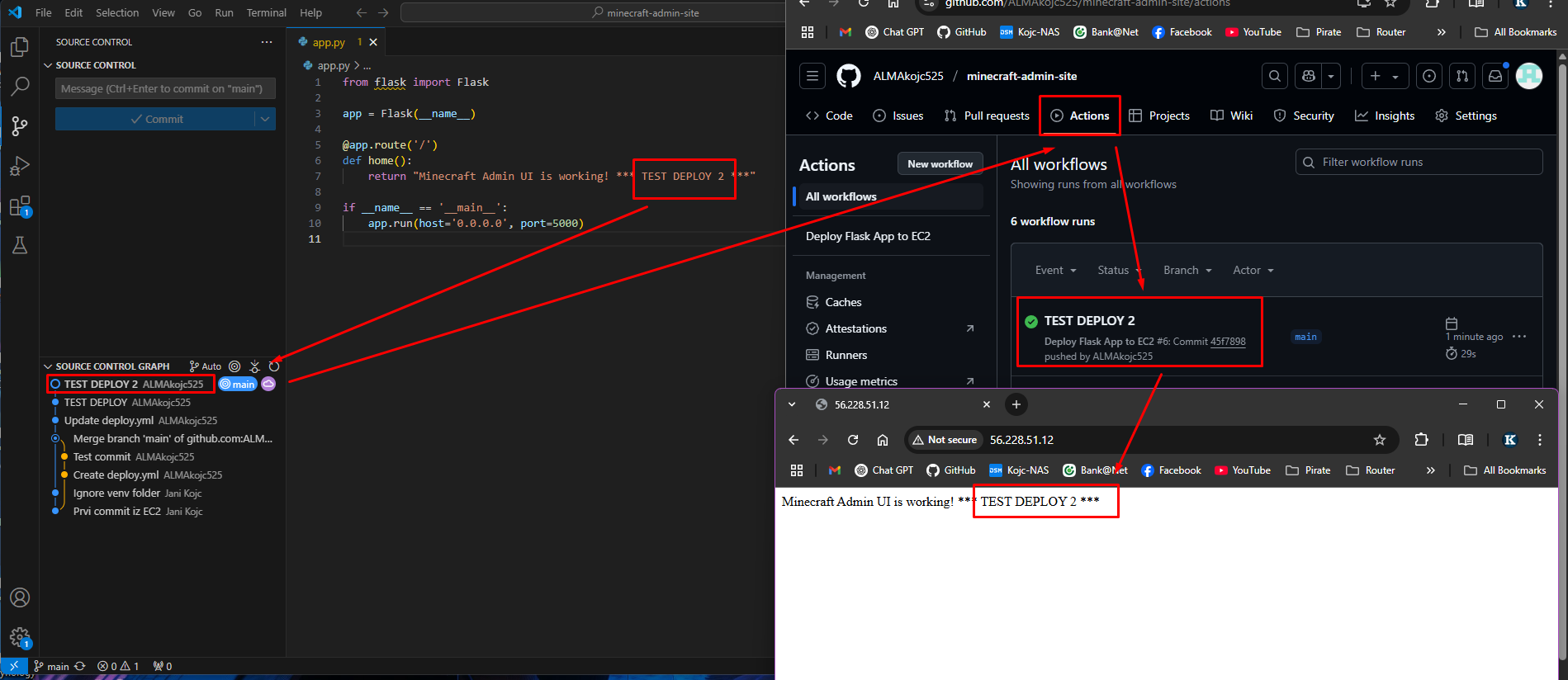
Slika :Prikazuje prvi commit iz EC2 strežnika v repozitorij »minecraft-admin-site«, kar potrjuje povezavo med lokalnim okoljem in oddaljenim GitHub repozitorijem



Slika :YAML datoteka deploy.yml, ki konfigurira GitHub Action za avtomatski deploy Flask aplikacije na EC2 ob vsakem push-u na vejo main.



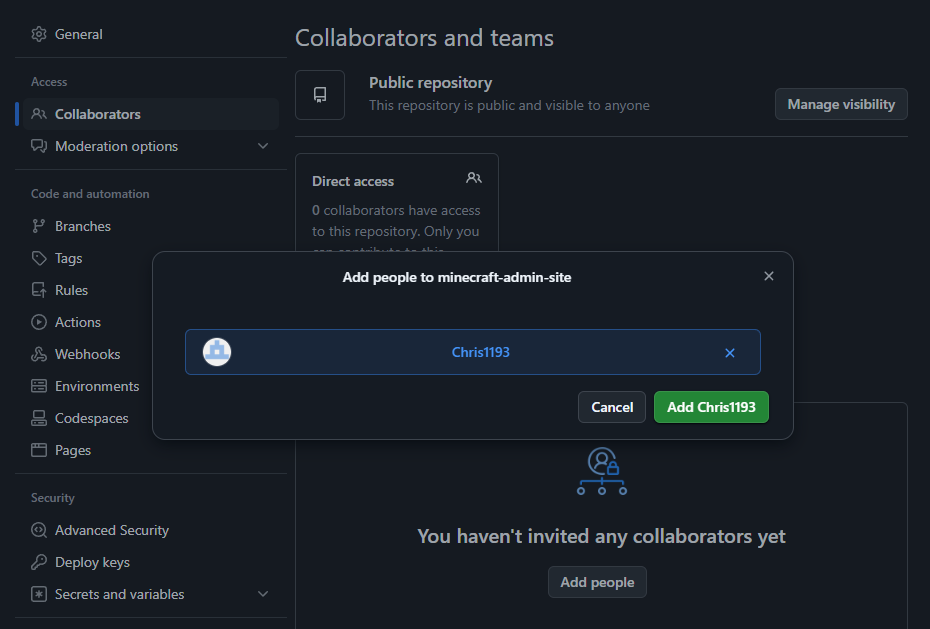
Slika : V GitHub repozitoriju je konfiguriran EC2\_SSH\_KEY kot skrivnost, kar omogoča varno povezavo prek SSH za dostop do instance.



Slika :Prikazuje uspešen test GitHub Action procesa — sprememba v kodi (tekst TEST DEPLOY 2) je bila uspešno prenesena in prikazana v aplikaciji.

# DODAJANJE UPORABNIKA NA GITHUB REPOZITORIJ

Za omogočanje sodelovanja več razvijalcev pri razvoju spletne aplikacije, je bil repozitorij »minecraft-admin-site« konfiguriran kot javni GitHub repozitorij z omogočenim dostopom za dodatne uporabnike. Dodajanje uporabnika poteka prek menija Settings > Collaborators, kjer se določi uporabniško ime in pravice dostopa.

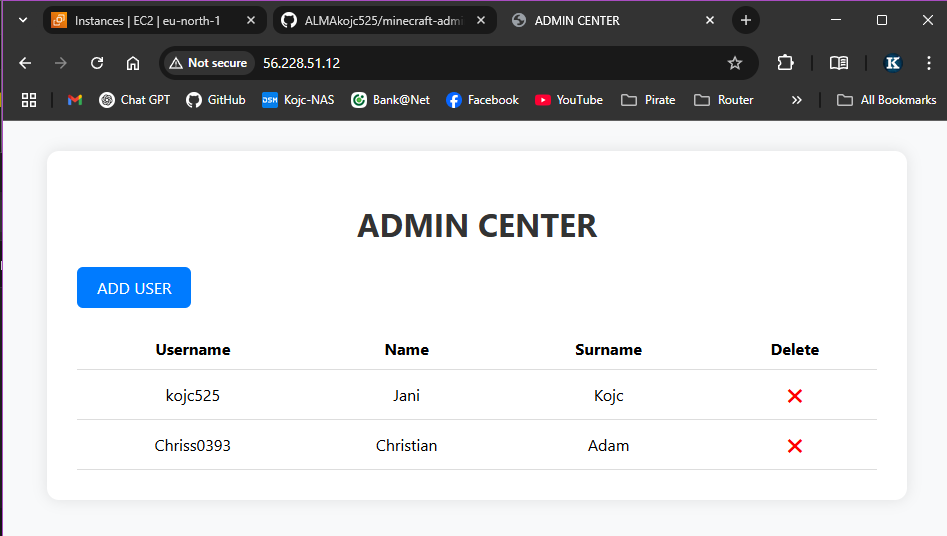


Slika : Prikazuje postopek dodajanja novega uporabnika Chris1193 kot sodelavca na GitHub repozitorij.

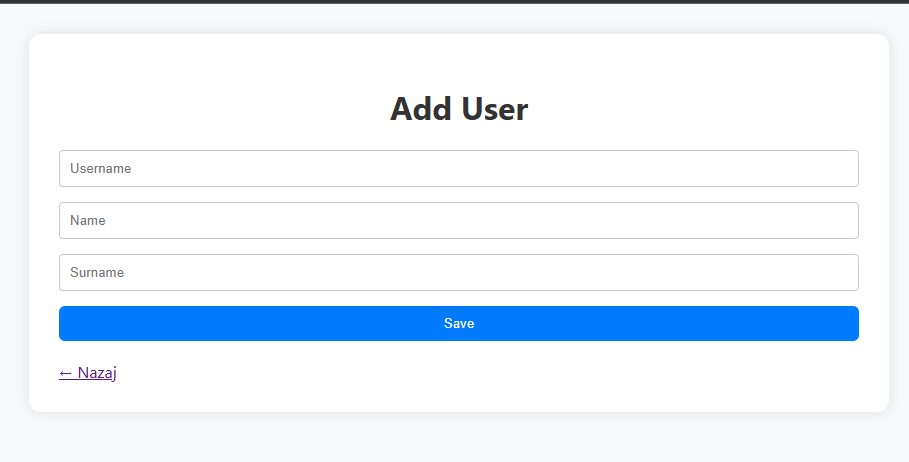
S tem je bil omogočen skupinski razvoj, kjer lahko oba razvijalca urejata kodo, izvajata push spremembe in spremljata avtomatski deploy s pomočjo GitHub Actions.

# ADMIN CENTER – UPRAVLJANJE UPORABNIKOV

Znotraj Flask aplikacije je bil razvit preprost grafični vmesnik za upravljanje uporabnikov, ki omogoča pregled, dodajanje in brisanje zapisov iz baze uporabnikov (Aurora RDS). Funkcionalnost je ključna za dinamično urejanje whiteliste igralcev, ki imajo dostop do Minecraft strežnika.



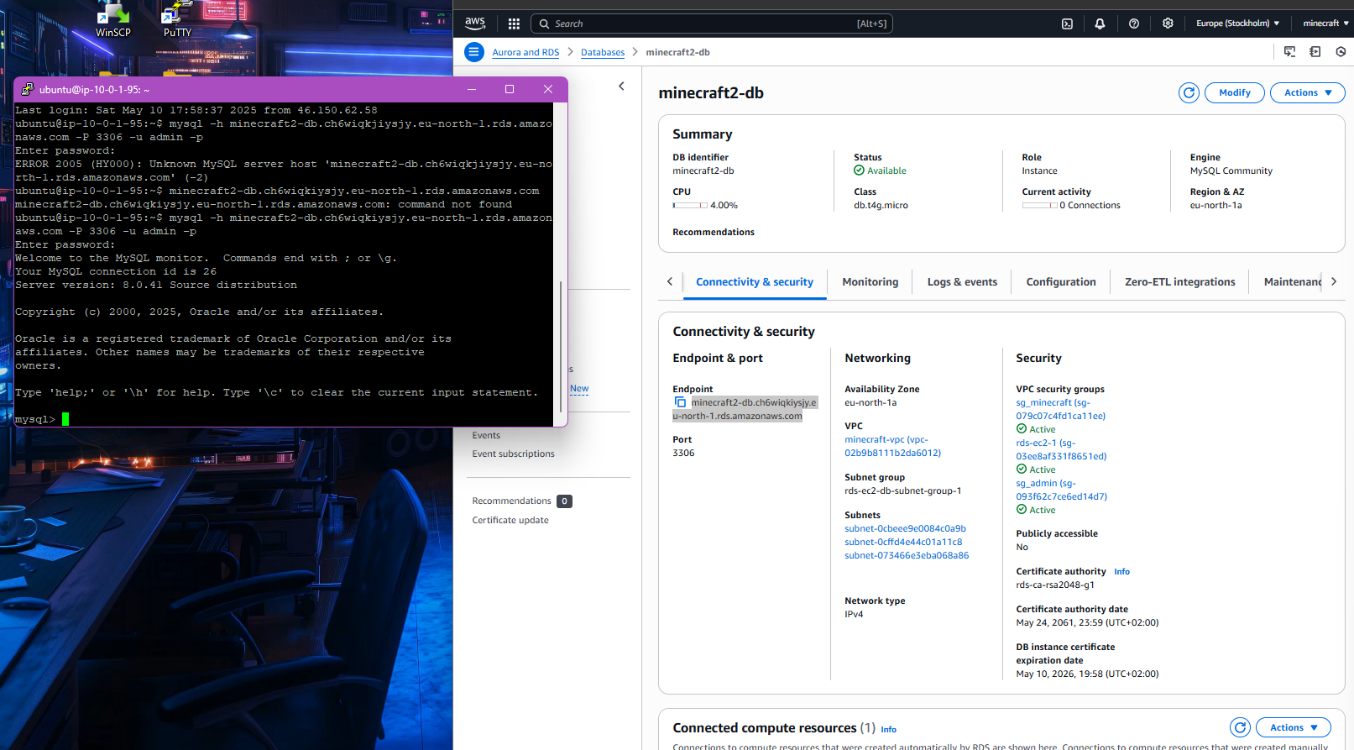
Slika : Osrednji prikaz administrativnega vmesnika (Admin Center) s seznamom uporabnikov in možnostjo brisanja.



Slika : Obrazec za dodajanje novega uporabnika v sistem. Polja vključujejo uporabniško ime, ime in priimek.

# PODATKOVNA BAZA UPORABNIKOV Z MYSQL NA AWS RDS

Za shranjevanje podatkov o uporabnikih je bila uporabljena storitev Amazon RDS z nameščenim MySQL strežnikom. Povezava do baze poteka prek javnega endpointa, dostop pa je omogočen samo instancam znotraj ustrezne varnostne skupine. V bazi »minecraft\_admin« je bila ustvarjena tabela »users«, ki vsebuje stolpce za uporabniško ime, ime in priimek.



Slika : Prikaz uspešne vzpostavitve povezave iz EC2 admin strežnika na RDS instanco in prikaz informacij o povezljivosti iz konzole AWS.



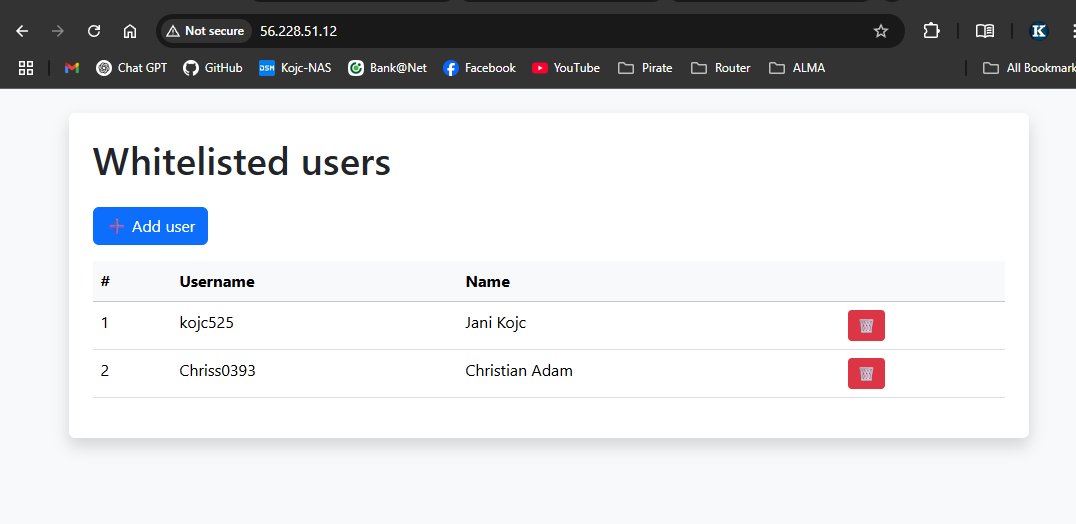
Slika : Postopek ustvarjanja baze in tabele »users« ter vnos testnega uporabnika Notch, z uspešnim prikazom vsebine tabele.

# PREVERJANJE POVEZAVE APLIKACIJE S PODATKOVNO BAZO

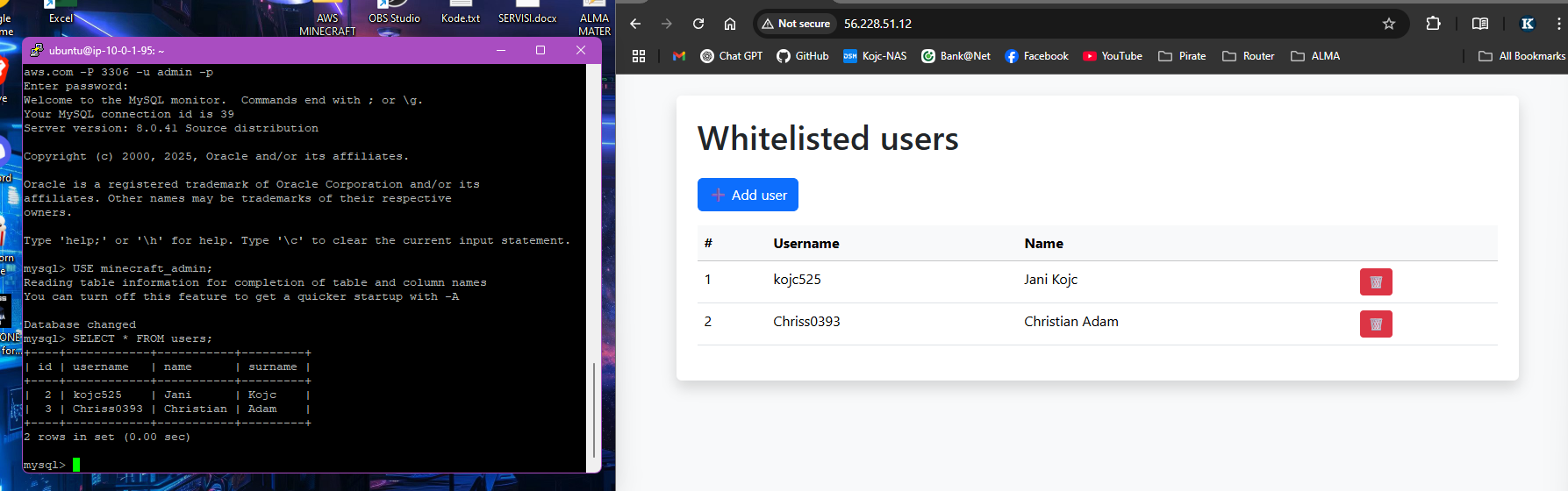
Po uspešni vzpostavitvi podatkovne baze in konfiguraciji povezave v Flask aplikaciji smo preverili, ali se vnos podatkov preko uporabniškega vmesnika dejansko odraža v MySQL bazi. Namen tega koraka je bil zagotoviti konsistenco podatkov in funkcionalno povezanost uporabniškega vmesnika s podatkovno plastjo.

Na spodnji sliki je prikazan terminalski izpis znotraj EC2 admin instance, kjer se je z uporabo SQL ukaza SELECT \* FROM users; pridobil trenutni seznam uporabnikov iz baze. Vzporedno je na desni strani prikazan uporabniški vmesnik, ki prikazuje popolnoma enak seznam — kar pomeni, da so podatki sinhronizirani in se uspešno zapisujejo v RDS bazo.

V nadaljevanju je bil vmesnik še dodatno osvežen z vizualnimi izboljšavami, z uporabo CSS stilov in bolj sodobnega oblikovanja. Nova različica vmesnika jasno prikazuje uporabniške vnose, skupaj z možnostjo njihovega brisanja prek intuitivnega gumba.



Slika : Prenovljena spletna stran z bolj sodobnim prikazom in funkcionalnostjo za brisanje posameznih vnosov iz baze.

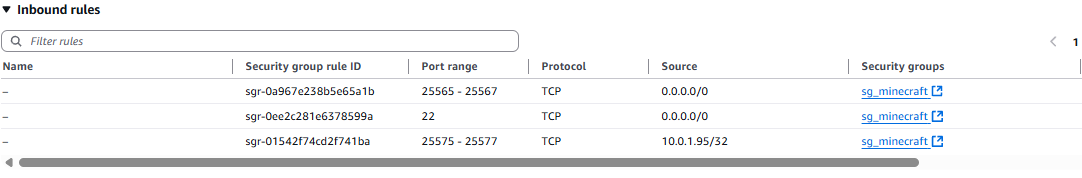


Slika : Prikaz delujoče povezave med MySQL bazo in spletno aplikacijo.

# TESTIRANJE RCON POVEZAVE IN POŠILJANJE UKAZOV

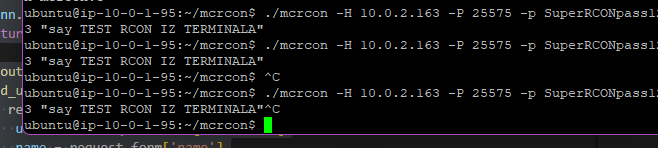
Za izboljšanje avtomatizacije in odpravo potrebe po prenosu datoteke *whitelist.json* sma se odločila za integracijo protokola **RCON (Remote Console)**. Prek tega vmesnika lahko Flask aplikacija pošilja neposredne ukaze na Minecraft strežnike.

Prvi korak je bil odprtje ustreznih vrat (25575–25577) v **security group** za instanco *minecraft-server*, s čimer smo omogočili dostop zgolj iz admin strežnika.



Slika : Nastavitve varnostnih pravil v AWS za odpiranje vrat RCON povezave (TCP porti 25575–25577)

Nato sma izvedli prvi test ukazov s pomočjo orodja mcrcon, kjer smo s terminala iz admin strežnika poslali ukaz say TEST RCON IZ TERMINALA. Spodnja slika prikazuje uspešen odziv.



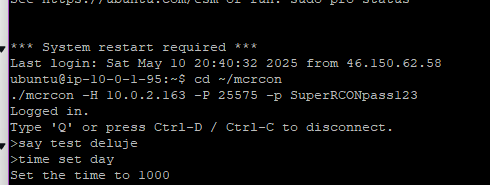
Slika : Vizualni prikaz prejetega ukaza v odjemalcu igre Minecraft, poslanega z RCON



Slika : Ukaz »say TEST RCON IZ TERMINALA« poslan prek terminala s pomočjo mcrcon.

V igri se je ta ukaz prikazal v pogovornem oknu vseh povezanih igralcev.

V nadaljevanju sma preverila tudi, ali je mogoče pošiljati dodatne ukaze, kot npr. nastavitev časa dneva, kar je bilo potrjeno z ukazom time set day.

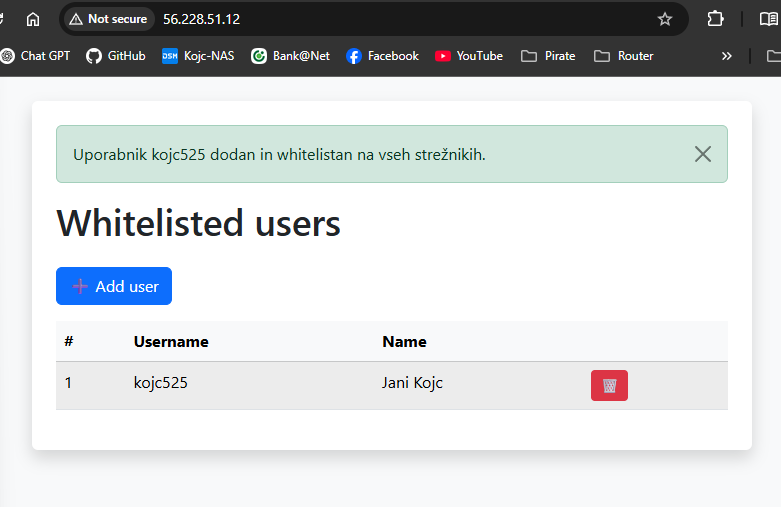


Slika : Pošiljanje več ukazov (say in time set day) z uporabo mcrcon v interaktivni seji.

Z uspešno vzpostavljeno povezavo sma s tem omogočila naslednji razvojni korak — povezavo Flask aplikacije z mcrcon, kar omogoča avtomatizirano whitelistanje uporabnikov neposredno preko spletnega vmesnika brez potrebe po ročnem posodabljanju datotek ali ponovnem zagonu strežnikov.

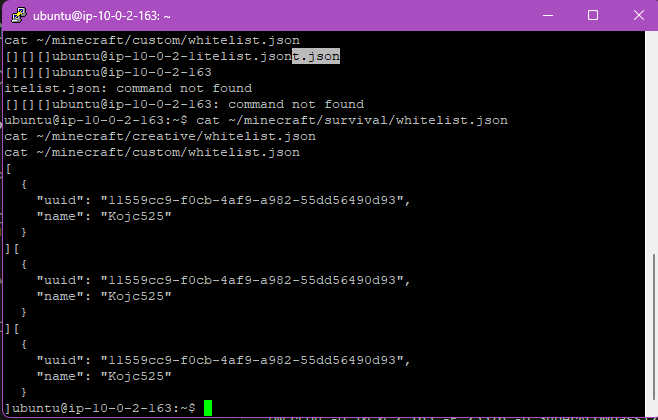
# WHITELIST UPORABNIKOV PREKO FLASK APLIKACIJE

Po vzpostavitvi RCON dostopa smo razširili Flask aplikacijo tako, da omogoča dodajanje in odstranjevanje uporabnikov iz whitelist seznama na vseh treh Minecraft strežnikih (survival, creative, custom).

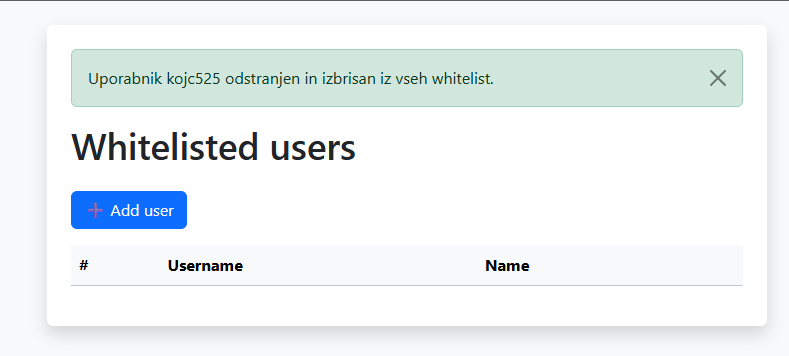


Slika :Uspešno dodan uporabnik na strani.png«: Uporabnik je uspešno dodan prek Flask spletnega vmesnika.

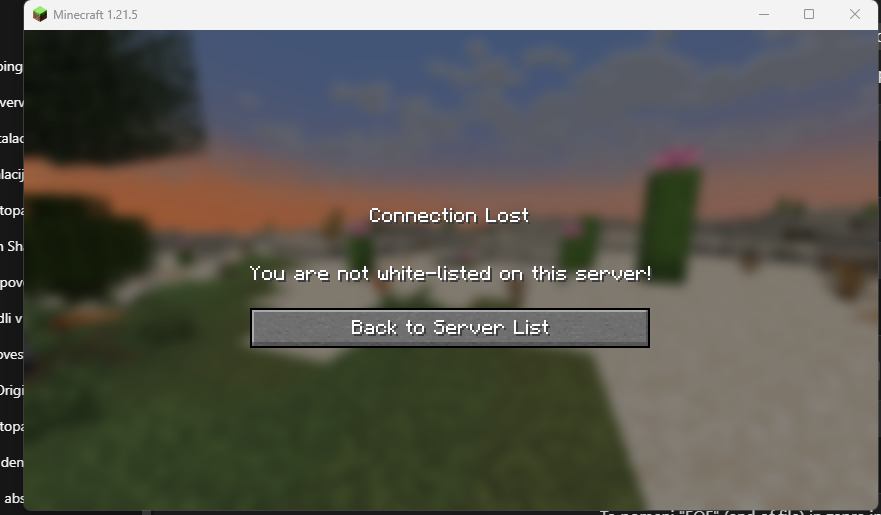
INSERT 2 spremembe vidne na strežniku uspešno dodane.png



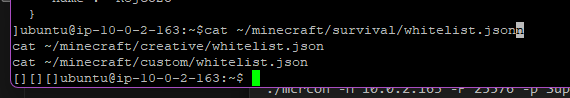
Slika : Spremembe vidne na strežniku uspešno dodane.png«: Prikaz whitelist.json datoteke z dodanim uporabnikom na vseh treh strežnikih.



Slika :Uporabnik uspešno odstranjen.png«: Obvestilo, da je uporabnik odstranjen iz baze in vseh whitelist.



Slika : Uporabnika vrže iz igre ko ga brišeš.png«: Minecraft obvestilo, da uporabnik ni več whitelistan in je bil odstranjen iz igre.



Slika : Uporabnika več ni na whitelistah od minecraft strežnikov.png«: Vsebina vseh treh whitelist.json datotek brez izbrisanega uporabnika.

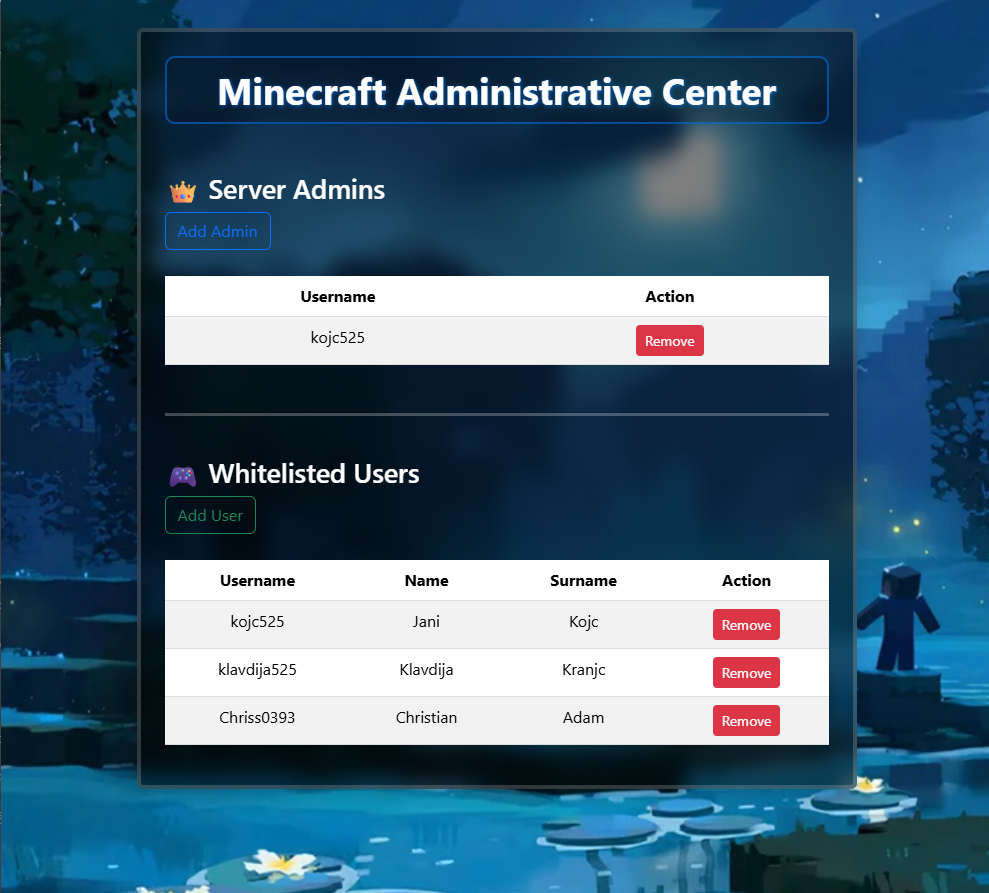
S tem je bila uspešno izvedena popolna integracija uporabniškega vmesnika z upravljanjem whitelist funkcionalnosti na več Minecraft strežnikih hkrati. Rešitev omogoča dinamično in varno upravljanje igralcev brez ročnega poseganja v strežniške datoteke.

# VIZUALNA PRENOVA ADMIN APLIKACIJE

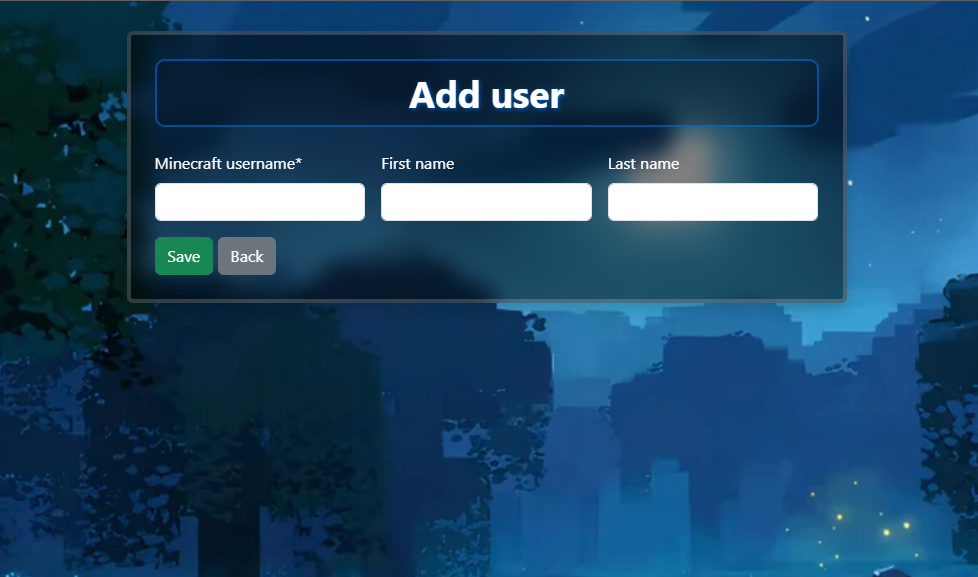
Za izboljšanje uporabniške izkušnje in boljšo ločenost vlog znotraj spletne aplikacije smo prenovili grafični vmesnik nadzorne plošče. Stran je bila oblikovana v sodobnem, temnem slogu z vizualnimi poudarki in ločenimi sekcijami za **administracijo strežnika** in **upravljanje whiteliste**.

Vmesnik omogoča:

* ločeno dodajanje admin uporabnikov,
* dodajanje splošnih igralcev (whitelist uporabnikov),
* ločeno prikazovanje in odstranjevanje obeh vrst uporabnikov.



Slika : Nova nadzorna plošča z ločenima seznamoma za administratorje in whitelistane uporabnike.



Slika : Obrazec za dodajanje uporabnika s polji za uporabniško ime, ime in priimek.



Slika : Obrazec za dodajanje novega admin uporabnika

S prenovo je aplikacija postala bolj intuitivna, pregledna in primerna za uporabo tudi na mobilnih napravah.

# ZAKLJUČEK

V seminarski nalogi smo uspešno zasnovali in vzpostavili infrastrukturno rešitev za gostovanje Minecraft strežnikov v okolju AWS. Uporabili smo dobre prakse varovanja, modularnosti in avtomatizacije. Z uporabo AWS storitev, kot so EC2, VPC, IAM, S3, Lambda in EventBridge, smo implementirali robusten in razširljiv sistem.

Nadaljnji koraki lahko vključujejo avtomatizirano skaliranje, obveščanje ob napakah (npr. z SNS), ter infrastrukturo kot kodo (Infrastructure as Code) z uporabo Terraform ali AWS CloudFormation. Projekt demonstrira praktično vrednost oblačnih storitev pri razvoju kompleksnih rešitev z minimalnimi začetnimi vložki.

Projekt je bil razvit v celoti samostojno, vključno s pisanjem aplikacijske kode, pripravo infrastrukture, testiranjem in dokumentiranjem vseh korakov. Vsaka komponenta je bila dejansko izvedena, kar je dokazano s slikovnim in funkcionalnim prikazom. Uporabljena je bila tudi dobra praksa varnega dostopa, ločevanja vlog in CI/CD procesa z GitHub Actions.

V prihodnosti bi bilo smiselno dodati funkcije, kot so integracija z Discord API za obveščanje, avtomatski monitoring obremenitve (npr. z CloudWatch) ter Terraform skripte za Infrastructure as Code.