Task 1

Review materials from the session

Task 2.1

Take any docker-compose application consisting of UI and backend database (for example, Prometheus+Grafana) and convert it to set of Kubernetes objects.

Task 2.2

Provide these artifacts for acceptance:

- Screenshot with running Pods and existing services, secrets, configmap
- Link to Github with k8s yaml files
- Screenshot of browser, showing that connection to Ui is successful

Task 2

Checking if kubectl is installed

```
anastasiiamazur@bttrm-amazur-pro16 ~ % kubectl version --client

Client Version: v1.30.5
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
```

Installation of Minikube

```
anastasiiamazur@bttrm-amazur-pro16 ~ % brew install minikube
anastasiiamazur@bttrm-amazur-pro16 ~ % minikube version

minikube version: v1.34.0
commit: 210b148df93a80eb872ecbeb7e35281b3c582c61
```

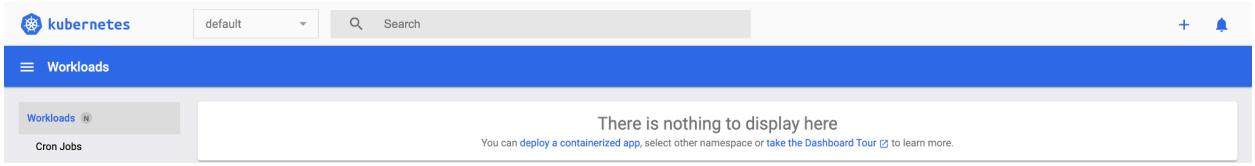
Starting the Minikube cluster and checking the cluster status

```
anastasiiamazur@bttrm-amazur-pro16 ~ % minikube start --driver=docker

😄  minikube v1.34.0 on Darwin 14.4.1 (arm64)
✨  Using the docker driver based on user configuration
📌  Using Docker Desktop driver with root privileges
👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.45 ...
💾  Downloading Kubernetes v1.31.0 preload ...
    > preloaded-images-k8s-v18-v1...:  307.61 MiB / 307.61 MiB  100.00% 25.28 M
    > gcr.io/k8s-minikube/kicbase...:  441.45 MiB / 441.45 MiB  100.00% 24.36 M
🔥  Creating docker container (CPUs=2, Memory=4000MB) ...
🐳  Preparing Kubernetes v1.31.0 on Docker 27.2.0 ...
    ▪ Generating certificates and keys ...
    ▪ Booting up control plane ...
    ▪ Configuring RBAC rules ...
🔗  Configuring bridge CNI (Container Networking Interface) ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Enabled addons: default-storageclass, storage-provisioner
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
anastasiiamazur@bttrm-amazur-pro16 ~ % kubectl get nodes

NAME       STATUS   ROLES           AGE   VERSION
minikube   Ready    control-plane   7s    v1.31.0
```

Emojis are amazing 🎉
I think everything is working fine 👋

Installing the Kubernetes Dashboard

```
anastasiiamazur@bttrm-amazur-pro16 ~ % minikube dashboard

🔌  Enabling dashboard ...
    ▪ Using image docker.io/kubernetesui/dashboard:v2.7.0
    ▪ Using image docker.io/kubernetesui/metrics-scraper:v1.0.8
💡  Some dashboard features require the metrics-server addon. To enable all features please run:

        minikube addons enable metrics-server

🤔  Verifying dashboard health ...
🚀  Launching proxy ...
🤔  Verifying proxy health ...
🎉  Opening http://127.0.0.1:58054/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/ in your default browser...
```



Web interfaces can already make work more fun 🙂

Minikube node stop

```
anastasiiamazur@bttrm-amazur-pro16 ~ % minikube stop

✋  Stopping node "minikube"  ...
🛑  Powering off "minikube" via SSH ...
🛑  1 node stopped.
```

I'm still in awe of the emoji... But let's get back to the task at hand. I'd like to try to do this task for the Minecraft app (just because it's quite amusing and I love this game)!

Initial compose.yaml

compose.yaml

```yaml
services:
  minecraft:
    image: itzg/minecraft-server
    ports:
      - "25565:25565"
    environment:
      EULA: "TRUE"
    deploy:
      resources:
        limits:
          memory: 1.5G
    volumes:
      - "~/minecraft_data:/data"
```

Installing and using kompose

```
anastasiiamazur@bttrm-amazur-pro16 minecraft-k8s % brew install kompose

==> Downloading https://formulae.brew.sh/api/formula.jws.json
######################################################################################################################################################## 100.0%
==> Downloading https://formulae.brew.sh/api/cask.jws.json
######################################################################################################################################################## 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/kompose/manifests/1.34.0
######################################################################################################################################################## 100.0%
==> Fetching kompose
==> Downloading https://ghcr.io/v2/homebrew/core/kompose/blobs/sha256:77c3111025da4a5bbca60e21ff63e519da381f4706bb180a6bf6ab706c9b58f
######################################################################################################################################################## 100.0%
==> Pouring kompose--1.34.0.arm64_sonoma.bottle.tar.gz
==> Caveats
zsh completions have been installed to:
  /opt/homebrew/share/zsh/site-functions
==> Summary
🍺  /opt/homebrew/Cellar/kompose/1.34.0: 9 files, 20.9MB
==> Running `brew cleanup kompose`...
Disable this behaviour by setting HOMEBREW_NO_INSTALL_CLEANUP.
Hide these hints with HOMEBREW_NO_ENV_HINTS (see `man brew`).
anastasiiamazur@bttrm-amazur-pro16 minecraft-k8s % kompose convert -f compose.yaml
WARN File don't exist or failed to check if the directory is empty: stat /Users/anastasiiamazur/minecraft_data: no such file or direc
tory
WARN Volume mount on the host "/Users/anastasiiamazur/minecraft_data" isn't supported - ignoring path on the host
INFO Kubernetes file "minecraft-service.yaml" created
INFO Kubernetes file "minecraft-deployment.yaml" created
INFO Kubernetes file "minecraft-claim0-persistentvolumeclaim.yaml" created
```

`kompose` simplifies the migration process and is an excellent starting point, but manual refinement is needed for production-grade deployments.

Kompose results overview

minecraft-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    kompose.cmd: kompose convert -f compose.yaml
    kompose.version: 1.34.0 (HEAD)
  labels:
    io.kompose.service: minecraft
  name: minecraft
spec:
  replicas: 1
  selector:
    matchLabels:
      io.kompose.service: minecraft
  strategy:
    type: Recreate
  template:
    metadata:
      annotations:
        kompose.cmd: kompose convert -f compose.yaml
        kompose.version: 1.34.0 (HEAD)
      labels:
        io.kompose.service: minecraft
    spec:
      containers:
        - env:
            - name: EULA
              value: "TRUE"
          image: itzg/minecraft-server
          name: minecraft
          ports:
            - containerPort: 25565
              protocol: TCP
          resources:
            limits:
              memory: "1610612736"
          volumeMounts:
            - mountPath: /data
              name: minecraft-claim0
      restartPolicy: Always
      volumes:
        - name: minecraft-claim0
          persistentVolumeClaim:
            claimName: minecraft-claim0
```

minecraft-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  annotations:
    kompose.cmd: kompose convert -f compose.yaml
    kompose.version: 1.34.0 (HEAD)
  labels:
    io.kompose.service: minecraft
  name: minecraft
spec:
  ports:
    - name: "25565"
      port: 25565
      targetPort: 25565
  selector:
    io.kompose.service: minecraft
```

minecraft-claim0-persistentvolumeclaim.yaml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  labels:
    io.kompose.service: minecraft-claim0
  name: minecraft-claim0
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
```

The generated files are not perfect but provide a solid starting point for migrating to Kubernetes. Here are the identified points to improve my final manifest files:

- remove unnecessary kompose annotations for cleaner YAML files and replace default names with descriptive and consistent identifiers
- adjust replicas for scalability
- change strategy to RollingUpdate for smoother updates
- move hardcoded environment variables to a ConfigMap
- define both memory and CPU requests/limits for better resource management and increase storage size
- change Service type to NodePort or LoadBalancer for external access
- use descriptive names for volumes and claims
- add probes to monitor container availability
- specify a namespace to isolate application resources

Create project structure

```
anastasiiamazur@bttrm-amazur-pro16 UCU_DevOps_Course % cd minecraft-k8s
anastasiiamazur@bttrm-amazur-pro16 minecraft-k8s % cd manifests
cd: no such file or directory: manifests
anastasiiamazur@bttrm-amazur-pro16 minecraft-k8s % mkdir manifests
anastasiiamazur@bttrm-amazur-pro16 minecraft-k8s % cd manifests
anastasiiamazur@bttrm-amazur-pro16 manifests % touch minecraft-deployment.yaml minecraft-service.yaml minecraft-configmap.yaml minecraft-pvc.yaml
```

Finalized YAML files

minecraft-configmap.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: minecraft-config
  namespace: minecraft # consistent namespace
data:
  EULA: "TRUE" # agree to minecraft's end user license ag
```

minecraft-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: minecraft
  namespace: minecraft # Ccnsistent namespace with deployment
spec:
  type: NodePort # expose the service externally
  ports:
  - port: 25565 # port exposed by the service
    targetPort: 25565 # port mapped inside the pod
    nodePort: 30000 # NodePort for external access
  selector:
    app: minecraft # route traffic to pods with this label
```

minecraft-pvc.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: minecraft-pv # PersistentVolume for data storage
  namespace: minecraft
spec:
  capacity:
    storage: 5Gi # total storage capacity
  accessModes:
    - ReadWriteOnce # allow single node to read/write
  hostPath:
    path: /data/minecraft # path on the Kubernetes node for storage
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: minecraft-pvc # PersistentVolumeClaim to request storage
  namespace: minecraft
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi # storage request size
```

minecraft-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: minecraft
  namespace: minecraft # isolate resources in a dedicated namespace
spec:
  replicas: 1 # number of pod replicas
  strategy:
    type: RollingUpdate # smooth updates without downtime
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  selector:
    matchLabels:
      app: minecraft # match pods with the specified label
  template:
    metadata:
      labels:
        app: minecraft
    spec:
      containers:
      - name: minecraft
        image: itzg/minecraft-server # minecraft server image
        ports:
        - containerPort: 25565 # standard minecraft server port
        envFrom:
        - configMapRef:
            name: minecraft-config # load environment variables from ConfigMap
        resources:
          requests:
            memory: "1Gi" # minimum memory required
            cpu: "500m" # minimum CPU required
          limits:
            memory: "1.5Gi" # maximum memory allowed
            cpu: "1" # maximum CPU allowed
        volumeMounts:
        - mountPath: /data # mount the data directory in the container
          name: minecraft-data
      volumes:
      - name: minecraft-data
        persistentVolumeClaim:
          claimName: minecraft-pvc # link to PersistentVolumeClaim
```

Add namespace file

```
anastasiiamazur@bttrm-amazur-pro16 manifests % touch minecraft-namespace.yaml
```

minecraft-namespace.yaml

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: minecraft
```

And then an incredibly long epic of 4 hours began, as I tried to fight with ports and public access... And now I will tell you this story

Create Kubernetes resources

```
kubectl apply -f minecraft-deployment.yaml
kubectl apply -f minecraft-service.yaml
kubectl apply -f minecraft-configmap.yaml
kubectl apply -f minecraft-pvc.yaml
```

Verify Kubernetes resources

```
anastasiiamazur@bttrm-amazur-pro16 manifests % kubectl get all -n minecraft

NAME                             READY   STATUS    RESTARTS   AGE
pod/minecraft-597545bcd7-xrhbr   1/1     Running   0          71m

NAME                 TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)           AGE
service/minecraft    LoadBalancer   10.106.113.31   127.0.0.1     25565:30000/TCP   98m

NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/minecraft    1/1     1            1           98m

NAME                                   DESIRED   CURRENT   READY   AGE
replicaset.apps/minecraft-597545bcd7   1         1         1       98m
```

Debugging pod and service

Get pods info

```
anastasiiamazur@bttrm-amazur-pro16 manifests % kubectl get pods -n minecraft
NAME                         READY   STATUS    RESTARTS   AGE
minecraft-597545bcd7-xrhbr   1/1     Running   0          73m
```

To see if the Minecraft server started properly

```
anastasiiamazur@bttrm-amazur-pro16 manifests % kubectl logs pod/minecraft-597545bcd7-xrhbr -n minecraft
[init] Running as uid=1000 gid=1000 with /data as 'drwxrwxrwx 7 1000 1000 4096 Dec  6 02:27 /data'
[init] Resolving type given VANILLA
[init] Resolved version given LATEST into 1.21.4
[mc-image-helper] 02:27:51.310 INFO  : Created/updated 1 property in /data/server.properties
[init] Setting initial memory to 1G and max to 1G
[init] Starting the Minecraft server...
Starting net.minecraft.server.Main
[02:27:58] [ServerMain/INFO]: Environment: Environment[sessionHost=https://sessionserver.mojang.com, servicesHost=https://api.minecraftserv
OD]
[02:28:01] [ServerMain/INFO]: Loaded 1370 recipes
[02:28:01] [ServerMain/INFO]: Loaded 1481 advancements
[02:28:01] [Server thread/INFO]: Starting minecraft server version 1.21.4
[02:28:01] [Server thread/INFO]: Loading properties
[02:28:01] [Server thread/INFO]: Default game type: SURVIVAL
[02:28:01] [Server thread/INFO]: Generating keypair
[02:28:01] [Server thread/INFO]: Starting Minecraft server on 0.0.0.0:25565
[02:28:02] [Server thread/INFO]: Using epoll channel type
[02:28:02] [Server thread/INFO]: Preparing level "world"
[02:28:02] [Server thread/INFO]: Preparing start region for dimension minecraft:overworld
[02:28:09] [Worker-Main-1/INFO]: Preparing spawn area: 0%
[02:28:09] [Worker-Main-1/INFO]: Preparing spawn area: 0%
[02:28:09] [Worker-Main-1/INFO]: Preparing spawn area: 0%
[02:28:09] [Worker-Main-1/INFO]: Preparing spawn area: 0%
[02:28:09] [Worker-Main-1/INFO]: Preparing spawn area: 0%
[02:28:09] [Worker-Main-1/INFO]: Preparing spawn area: 0%
[02:28:09] [Worker-Main-1/INFO]: Preparing spawn area: 0%
[02:28:09] [Worker-Main-1/INFO]: Preparing spawn area: 0%
[02:28:09] [Worker-Main-1/INFO]: Preparing spawn area: 0%
[02:28:09] [Worker-Main-1/INFO]: Preparing spawn area: 0%
[02:28:09] [Worker-Main-1/INFO]: Preparing spawn area: 0%
[02:28:09] [Worker-Main-1/INFO]: Preparing spawn area: 0%
[02:28:09] [Worker-Main-1/INFO]: Preparing spawn area: 0%
[02:28:09] [Worker-Main-1/INFO]: Preparing spawn area: 0%
[02:28:09] [Worker-Main-1/INFO]: Preparing spawn area: 0%
[02:28:09] [Server thread/INFO]: Time elapsed: 7417 ms
[02:28:09] [Server thread/INFO]: Done (7.752s)! For help, type "help"
[02:28:09] [Server thread/INFO]: Starting GS4 status listener
[02:28:09] [Server thread/INFO]: Thread Query Listener started
[02:28:09] [Query Listener #1/INFO]: Query running on 0.0.0.0:25565
[02:28:09] [Server thread/INFO]: Starting remote control listener
[02:28:09] [Server thread/INFO]: Thread RCON Listener started
[02:28:09] [Server thread/INFO]: RCON running on 0.0.0.0:25575
[02:29:09] [Server thread/INFO]: Server empty for 60 seconds, pausing
[02:43:40] [Server thread/WARN]: handleDisconnection() called twice
[02:48:40] [Server thread/WARN]: Can't keep up! Is the server overloaded? Running 2026ms or 40 ticks behind
[03:06:27] [User Authenticator #1/INFO]: UUID of player skajeniy is 6ed8f124-3650-4912-9939-57ad1fb15c1a
[03:06:28] [Server thread/INFO]: skajeniy[/10.244.0.1:23371] logged in with entity id 6 at (-17.5, 67.0, 248.5)
[03:06:28] [Server thread/INFO]: skajeniy joined the game
[03:08:19] [Server thread/INFO]: skajeniy lost connection: Disconnected
[03:08:19] [Server thread/INFO]: skajeniy left the game
[03:08:32] [User Authenticator #2/INFO]: UUID of player skajeniy is 6ed8f124-3650-4912-9939-57ad1fb15c1a
[03:08:33] [Server thread/INFO]: skajeniy[/10.244.0.1:9105] logged in with entity id 524 at (-22.24353096520244, 67.0, 247.46100680568765)
[03:08:33] [Server thread/INFO]: skajeniy joined the game
[03:09:15] [Server thread/INFO]: skajeniy lost connection: Disconnected
[03:09:15] [Server thread/INFO]: skajeniy left the game
```

To verify if the service is correctly configured and exposes the necessary ports

```
anastasiiamazur@bttrm-amazur-pro16 manifests % kubectl describe svc minecraft -n minecraft

Name:                     minecraft
Namespace:                minecraft
Labels:                   <none>
Annotations:              <none>
Selector:                 app=minecraft
Type:                     LoadBalancer
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.106.113.31
IPs:                      10.106.113.31
LoadBalancer Ingress:     127.0.0.1
Port:                     <unset>  25565/TCP
TargetPort:               25565/TCP
NodePort:                 <unset>  30000/TCP
Endpoints:                10.244.0.5:25565
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

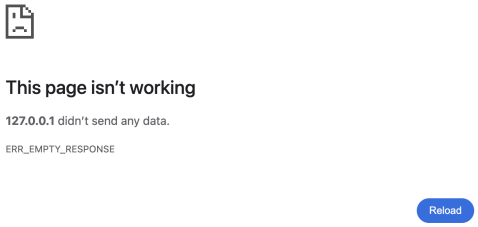
To process PersistentVolumeClaim check

```
anastasiiamazur@bttrm-amazur-pro16 manifests % kubectl get pvc -n minecraft

NAME            STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
minecraft-pvc   Bound    pvc-b374198b-8ace-4dca-bd3e-7516a3f71d3c   5Gi        RWO            standard       <unset>                 114m
```

Getting access to the Minecraft server

```
anastasiiamazur@bttrm-amazur-pro16 manifests % minikube service minecraft -n minecraft

|-----------|-----------|-------------|---------------------------|
| NAMESPACE |   NAME    | TARGET PORT |            URL            |
|-----------|-----------|-------------|---------------------------|
| minecraft | minecraft |       25565 | http://192.168.49.2:30000 |
|-----------|-----------|-------------|---------------------------|
🏃  Starting tunnel for service minecraft.
|-----------|-----------|-------------|------------------------|
| NAMESPACE |   NAME    | TARGET PORT |          URL           |
|-----------|-----------|-------------|------------------------|
| minecraft | minecraft |             | http://127.0.0.1:50226 |
|-----------|-----------|-------------|------------------------|
🎉  Opening service minecraft/minecraft in default browser...
❗  Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
```

Minikube has created a tunnel to the Minecraft service on port 59669 for local access (127.0.0.1:59669).
I also have direct access via IP 192.168.49.2:30000 (NodePort).
BUT the Minecraft server is not an HTTP service, so accessing it through a browser (HTTP request) will not display data.



The server is running on port 25565, and the Minecraft client expects traffic through this port. Accessing it through a browser doesn't work because Minecraft is a TCP protocol, not HTTP. So I needed to check the connection through the Minecraft client or test the port manually.

And I've tried that so many times via https://mcsrvstat.us/ and https://mcstatus.io/, but that what But I kept getting access problems or messages about the server being inactive. This is what it looks like:

Adjust server.properties

I checked and modified the server.properties file to ensure:

- server-ip=0.0.0.0 (bind to all network interfaces)
- enable-query=true (allows status queries)
- query.port=25565 (default Minecraft query port)
- prevent-proxy-connections=false (to set up stable external connections)

Port forwarding for local access

To test the server locally, I forwarded the pod's port to the local machine

```
anastasiiamazur@bttrm-amazur-pro16 manifests % kubectl port-forward pod/minecraft-597545bcd7-8b6lv 25566:25565 -n minecraft

Forwarding from 127.0.0.1:25566 -> 25565
Forwarding from [::1]:25566 -> 25565
```

LoadBalancer in Minikube

Minikube cannot directly expose LoadBalancer services without additional addons (minikube tunnel was partially functional)

```
anastasiiamazur@bttrm-amazur-pro16 manifests % minikube tunnel

✅  Tunnel successfully started

📌  NOTE: Please do not close this terminal as this process must stay alive for the tunnel to be accessible ...

🏃  Starting tunnel for service minecraft.
```

Before trying the LoadBalancer, I explored other methods like NodePort and kubectl port-forward, which didn't fully work.
Minikube assigns the cluster an internal IP (e.g., 192.168.49.2), which is not directly accessible from external networks without additional configuration.
External access requires minikube tunnel, but even with tunnel, NodePort didn't always work due to local routing issues or firewall restrictions (AND FIREWALL IS ACTUALLY THE MAIN ANTAGONIST OF THE STORY).

After many failed retries and approaches testing I've managed to get external-ip and pass some local checks:

```
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)           AGE
service/minecraft   LoadBalancer   10.106.113.31   127.0.0.1     25565:30000/TCP   98m
```

You can see below the application (10.244.0.3:25565) is running correctly, and the backend is accessible within the cluster and LoadBalancer setup in Minikube is not properly exposing the service due to limitations in the environment and configuration.

```
anastasiiamazur@bttrm-amazur-pro16 manifests % kubectl describe svc minecraft -n minecraft

Name:                     minecraft
Namespace:                minecraft
Labels:                   <none>
Annotations:              <none>
Selector:                 app=minecraft
Type:                     LoadBalancer
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.106.113.31
IPs:                      10.106.113.31
LoadBalancer Ingress:     127.0.0.1
Port:                     <unset>  25565/TCP
TargetPort:               25565/TCP
NodePort:                 <unset>  30000/TCP
Endpoints:                10.244.0.3:25565
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
anastasiiamazur@bttrm-amazur-pro16 manifests % minikube ssh

docker@minikube:~$ nc -zv 10.244.0.3 25565
Connection to 10.244.0.3 25565 port [tcp/*] succeeded!
docker@minikube:~$ nc -zv 192.168.49.2 30000
Connection to 192.168.49.2 30000 port [tcp/*] succeeded!
```

The issue arises because the LoadBalancer in Minikube incorrectly binds to 127.0.0.1 (localhost), limiting external access, while the Minecraft server is reachable internally but doesn't respond to HTTP queries as it uses a custom protocol.

```
docker@minikube:~$ curl 10.244.0.3:25565
curl: (52) Empty reply from server
docker@minikube:~$ exit
```

Solution (well, finally)

After conducting extensive research on how to expose a local Minecraft server running in a Minikube environment to external users, I identified several approaches, including configuring a NodePort service, using a LoadBalancer, or employing a tunneling solution. While NodePort exposes services on a specific port of the Minikube host and LoadBalancer works well with cloud providers, both methods faced limitations in Minikube due to its design and lack of direct public IP exposure.

The main challenge, however, was my **firewall restrictions**, which prevented me from modifying its settings or allowing external traffic directly to the required ports. This added complexity because even with an exposed NodePort or LoadBalancer, connections were being blocked at the system level, making the server inaccessible to users outside my local network.

One effective solution I implemented was **ngrok**, a tunneling tool that creates a secure, temporary public endpoint for local applications. Ngrok establishes a connection to its servers and forwards incoming traffic to the specified local port, bridging the gap between the local Kubernetes environment and external networks.

1. Ngrok provides a public URL or address that can be shared with clients, bypassing Minikube's inability to assign an external IP to a LoadBalancer.
2. It supports non-HTTP protocols like TCP, which is essential for the Minecraft server as it communicates using its own protocol on port 25565.

```
manifests % brew install ngrok/ngrok/ngrok
==> Auto-updating Homebrew...
Adjust how often this is run with HOMEBREW_AUTO_UPDATE_SECS or disable with
HOMEBREW_NO_AUTO_UPDATE. Hide these hints with HOMEBREW_NO_ENV_HINTS (see `man brew`).
==> Tapping ngrok/ngrok
Cloning into '/opt/homebrew/Library/Taps/ngrok/homebrew-ngrok'...
remote: Enumerating objects: 241, done.
remote: Counting objects: 100% (88/88), done.
remote: Compressing objects: 100% (55/55), done.
remote: Total 241 (delta 22), reused 60 (delta 19), pack-reused 153 (from 1)
Receiving objects: 100% (241/241), 49.53 KiB | 757.00 KiB/s, done.
Resolving deltas: 100% (55/55), done.
Tapped 1 cask (13 files, 62.7KB).
==> Downloading https://bin.equinox.io/a/eb5fgv4hujc/ngrok-v3-3.18.4-darwin-arm64.zip
######################################################################################################################## 100.0%
==> Installing Cask ngrok
==> Linking Binary 'ngrok' to '/opt/homebrew/bin/ngrok'
🍺  ngrok was successfully installed!
```

```
anastasiiamazur@bttrm-amazur-pro16 manifests % ngrok tcp 25565
```

```
manifests — docker@minikube: ~ — ngrok tcp 25565 — 122×46
ngrok                                                                    (Ctrl+C to quit)

🍾 Found a bug? Let us know: https://github.com/ngrok/ngrok

Session Status                online
Account                       nastiamazur.v@gmail.com (Plan: Free)
Version                       3.18.4
Region                        Europe (eu)
Latency                       35ms
Web Interface                 http://127.0.0.1:4040
Forwarding                    tcp://7.tcp.eu.ngrok.io:12948 -> localhost:25565

Connections                   ttl     opn     rt1     rt5     p50     p90
                              32      0       0.00    0.00    0.07    30.17
```

Unlike the above methods, ngrok worked because:

- It initiates an outbound connection to its cloud servers, bypassing firewall restrictions
- Ngrok provided a publicly accessible address (7.tcp.eu.ngrok.io:12948), which routed traffic to my local server
- No need to configure Minikube networking, port forwarding, or my firewall

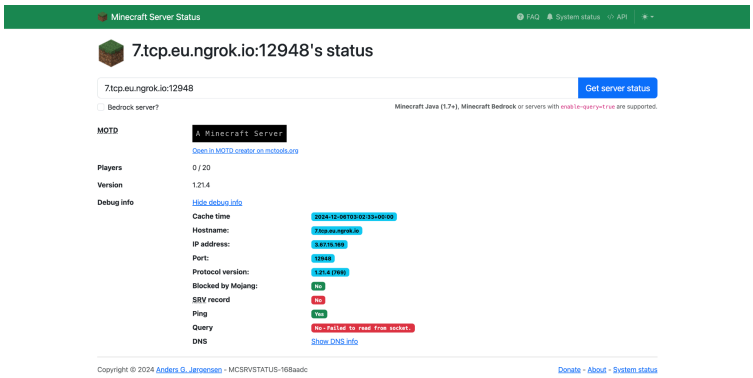
Visual results overview

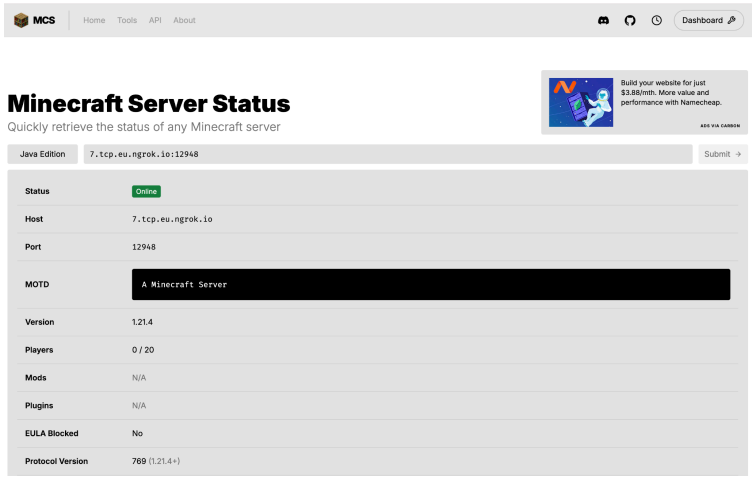
This was my first attempt to connect another user to the server with the help of friends (how good that they don't sleep at night either)

Obviously, it didn't work 🙂

And here are the actual results of working system 👋

Results of checking server access on two different resources:



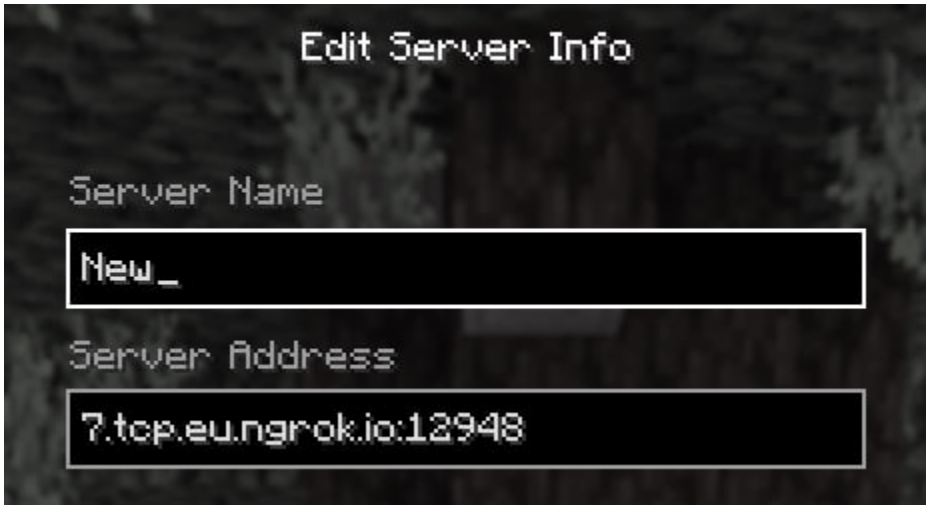


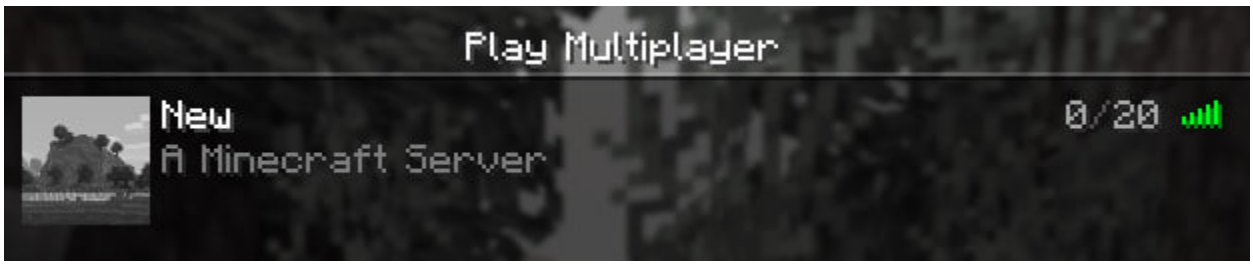
And via a web request as well:

```
[11] import socket

     server = '7.tcp.eu.ngrok.io'
     port = 12948

     try:
         sock = socket.create_connection((server, port), timeout=5)
         print("Connection to Minecraft server succeeded!")
         sock.close()
     except Exception as e:
         print(f"Failed to connect: {e}")
```

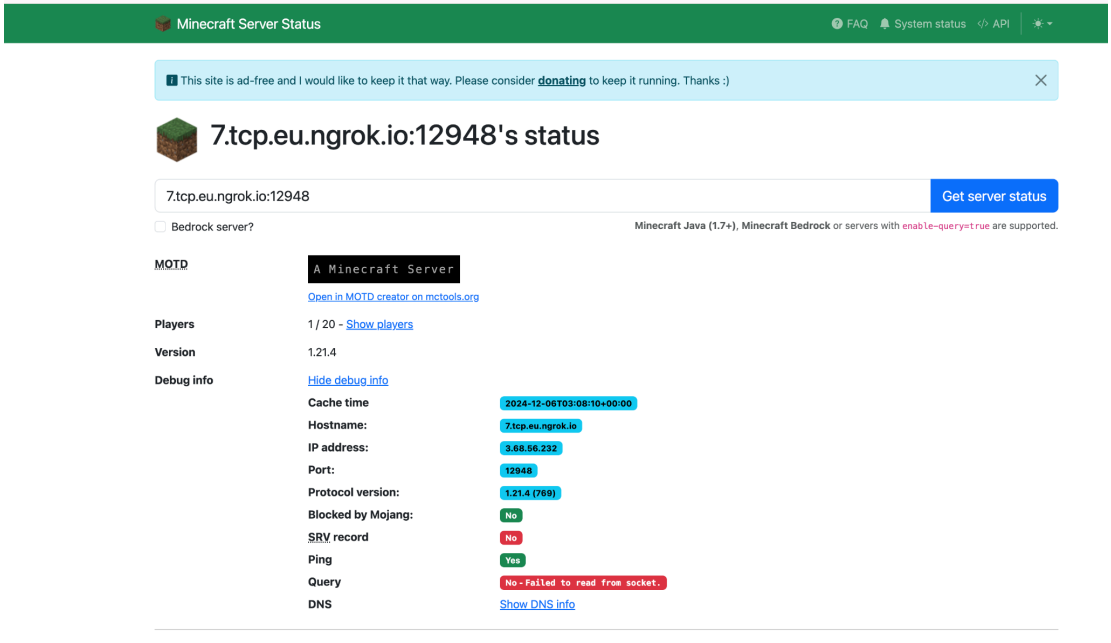
```
Connection to Minecraft server succeeded!
```

What my assistant saw and was able to do on the server:





While he was on the server, I, for my part, was able to see the existing player through the web resource, as well as in the system logs of pod:




```
[03:06:27] [User Authenticator #1/INFO]: UUID of player skajeniy is 6ed8f124-3650-4912-9939-57ad1fb15c1a
[03:06:28] [Server thread/INFO]: skajeniy[/10.244.0.1:23371] logged in with entity id 6 at (-17.5, 67.0, 248.5)
[03:06:28] [Server thread/INFO]: skajeniy joined the game
[03:08:19] [Server thread/INFO]: skajeniy lost connection: Disconnected
[03:08:19] [Server thread/INFO]: skajeniy left the game
[03:08:32] [User Authenticator #2/INFO]: UUID of player skajeniy is 6ed8f124-3650-4912-9939-57ad1fb15c1a
[03:08:33] [Server thread/INFO]: skajeniy[/10.244.0.1:9105] logged in with entity id 524 at (-22.24353096520244, 67.0, 247.46100680568765)
[03:08:33] [Server thread/INFO]: skajeniy joined the game
[03:09:15] [Server thread/INFO]: skajeniy lost connection: Disconnected
[03:09:15] [Server thread/INFO]: skajeniy left the game
```

System results overview

List Running Pods

```
anastasiiamazur@bttrm-amazur-pro16 manifests % kubectl get pods -n minecraft

NAME                         READY   STATUS    RESTARTS   AGE
minecraft-597545bcd7-xrhbr   1/1     Running   0          132m
```

List Existing Services

```
anastasiiamazur@bttrm-amazur-pro16 manifests % kubectl get svc -n minecraft

NAME        TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)           AGE
minecraft   LoadBalancer   10.106.113.31   127.0.0.1     25565:30000/TCP   159m
```

List ConfigMaps

```
anastasiiamazur@bttrm-amazur-pro16 manifests % kubectl get configmaps -n minecraft

NAME               DATA   AGE
kube-root-ca.crt   1      160m
minecraft-config   1      159m
```

Link to Github:
https://github.com/idgafd/UCU_DevOps_Course/tree/main/minecraft-k8s

It was... at least interesting, but now Minecraft doesn't seem like such a fun game to me.