

UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Proyecto Semestral

Servidor Minecraft en red local

Despliegue automatizado y observabilidad

Fecha: Noviembre de 2026
Integrantes: Alonso Toro
Paulo Vargas
Asignatura: Administración de Redes de Computadores
Profesor: Alexis Gaete Olivares

Índice

1. Introducción **3**

2. Problemática y objetivos **3**
2.1. Problemática . . . 3
2.2. Objetivo general . . . 3
2.3. Objetivos específicos . . . 3

3. Arquitectura de la solución **3**
3.1. Distribución de contenedores e imágenes OCI . . . 4
3.2. Aislamiento de red y resolución DNS embebida . . . 4
3.3. Persistencia de datos y permisos de usuario (`UID/GID`) . . . 5

4. Implementación y automatización (8 Scripts) **5**
4.1. Fase 1 de aprovisionamiento: `dependencia.sh` . . . 5
4.2. Fase 2 de aprovisionamiento: `configuracion.sh` . . . 5
4.3. Fase 3 de aprovisionamiento: `despliegue.sh` . . . 7
4.4. Fase 4 de aprovisionamiento: `backups.sh` . . . 8
4.5. Scripts de operación y administración diaria (Día 2) . . . 8

5. Monitoreo y administración del servicio **9**

6. Pruebas de funcionamiento y resultados esperados **9**

7. Evidencias de implementación y funcionamiento **9**
7.1. Estado de los servicios . . . 10
7.2. Conexión del cliente al servidor . . . 10
7.3. Participación de jugadores y carga de skins . . . 11
7.4. Verificación interna de salud y objetivos en Prometheus . . . 11
7.5. Disponibilidad de paneles aprovisionados . . . 12
7.6. Visualización de Grafana (Métricas de Minecraft) . . . 12
7.7. Indicadores del sistema Linux . . . 13
7.8. Comportamiento durante actividad en el servidor . . . 14
7.9. Generación del respaldo . . . 14

8. Consideraciones de seguridad y limitaciones **15**

9. Conclusiones **15**

Índice de figuras

1. Arquitectura lógica del servicio Minecraft y de su sistema de observabilidad en LAN. . . . 4
2. Estado y consumo de recursos de los servicios Docker tras el despliegue de la plataforma. . . . 10
3. Conexión de un cliente Minecraft al servidor publicado en la red local. . . . 10
4. Sesión activa en el servidor Minecraft dentro de la red local. . . . 11
5. Actualizacion en el dashboard con 2 jugadores . . . 11
6. Comprobación de salud y objetivos activos directamente desde el contenedor Prometheus. . . . 12
7. Paneles aprovisionados automáticamente y alimentados por Prometheus. . . . 12
8. Dashboard de Minecraft disponible desde la interfaz web de Grafana. . . . 13
9. Dashboard estadísticas de usuarios disponible desde la interfaz web de Grafana. . . . 13
10. Panel de monitoreo de los recursos del host Linux mediante Node Exporter. . . . 14
11. Variación de indicadores del servidor registrada en las gráficas de series temporales. . . . 14
12. Archivo comprimido generado mediante el procedimiento automatizado de respaldo del mundo. . . . 15

1. Introducción

La habilitación de un servidor de videojuegos en una red local parece una tarea sencilla, pero su administración implica desafíos similares a los presentes en otros servicios informáticos: instalación de dependencias, configuración reproducible, disponibilidad, supervisión de recursos y resguardo de información. Cuando el servicio se instala manualmente, cada nuevo despliegue puede requerir repetir pasos y resolver problemas de configuración. Además, una vez iniciado, es difícil reconocer la causa de una caída de rendimiento sin contar con métricas del sistema y de la aplicación.

Este proyecto utiliza Minecraft como caso práctico para abordar dichas tareas bajo un enfoque de Infraestructura como Código (IaC). La solución se implementa sobre una máquina virtual Debian, la misma usada en todas las experiencias del laboratorio, y emplea Docker Compose para ejecutar un servidor PaperMC junto con servicios de monitoreo multicapa (Prometheus, Node Exporter y Grafana). Se automatizan la preparación del entorno, el aprovisionamiento de configuraciones, el despliegue, la programación de respaldos y la operación diaria mediante ocho scripts modulares de shell (cuatro de aprovisionamiento base y cuatro de operación administrativa). De este modo, el énfasis del proyecto no está únicamente en permitir una partida multijugador, sino en administrar un servicio resiliente y observable accesible desde la misma red Wi-Fi.

2. Problemática y objetivos

2.1. Problemática

Un grupo de usuarios desea habilitar rápidamente un servidor Minecraft para jugar desde distintos computadores conectados a una misma red local. La instalación manual exige preparar el sistema, configurar el servidor y resolver la conectividad. Posteriormente surgen nuevas necesidades: comprobar si el servicio sigue funcionando, identificar qué ocurre cuando existe retraso (*lag*) durante la partida y conservar los datos del mundo ante un reinicio o falla.

La problemática abordada consiste, por tanto, en **simplificar el despliegue y mejorar la administración operativa de un servicio multijugador en LAN**, utilizando herramientas estándar de administración de redes y sistemas.

2.2. Objetivo general

Implementar una plataforma reproducible para alojar un servidor Minecraft en una red local, automatizando su despliegue y habilitando monitoreo en tiempo real del sistema anfitrión y de la aplicación.

2.3. Objetivos específicos

- Automatizar la preparación de Debian y la instalación de Docker y Docker Compose v2 mediante scripts de aprovisionamiento.
- Desplegar los servicios de Minecraft, Prometheus, Node Exporter y Grafana en contenedores interconectados bajo una red privada aislada.
- Habilitar el acceso de múltiples jugadores desde la LAN (incorporando soporte de skins locales mediante `SkinsRestorer`) y visualizar indicadores de funcionamiento desde un navegador web con cuatro paneles preconfigurados.
- Configurar recopilación periódica de métricas, almacenamiento persistente mediante montajes enlazados (*bind mounts*) y asignación estricta de permisos de usuario (`UID/GID`).
- Programar copias de respaldo automatizadas con rotación de antigüedad, proveer scripts de operación diaria y documentar pruebas de funcionamiento mediante evidencias.

3. Arquitectura de la solución

La plataforma utiliza una máquina virtual Debian conectada mediante un adaptador de red en modo puente (*Bridged Adapter*). En este modo de conmutación en Capa 2, la máquina virtual participa directamente en la subred local Wi-Fi y obtiene su propia dirección IP (`IP_VM`, por ejemplo `192.168.1.41` o `192.168.1.45` según la asignación DHCP del entorno), permitiendo que múltiples clientes de la misma LAN $(1, 2, \ldots, N)$ se conecten directamente sin doble traducción de direcciones (NAT).

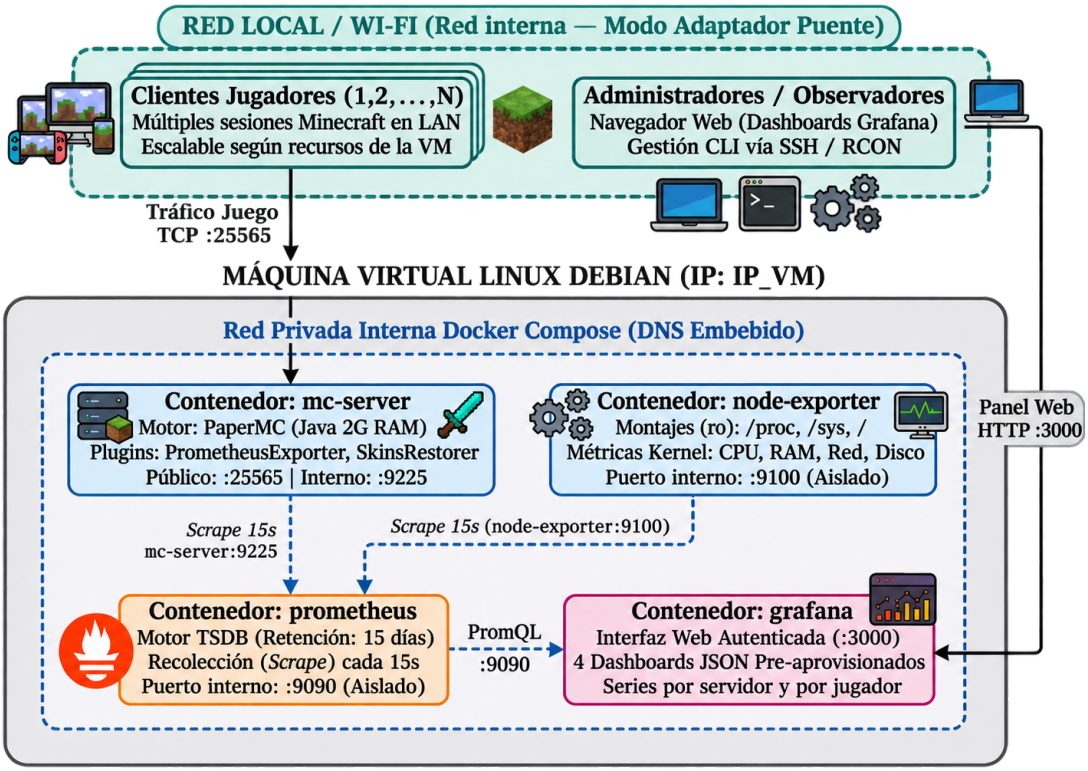


Figura 1: Arquitectura lógica del servicio Minecraft y de su sistema de observabilidad en LAN.

3.1. Distribución de contenedores e imágenes OCI

En lugar de instalar Java, bases de datos y servidores web directamente sobre el sistema anfitrión, la solución descarga cuatro imágenes oficiales desde registros públicos y las ejecuta como contenedores aislados mediante espacios de nombres (*namespaces*) y grupos de control (*cgroups*) del kernel Linux:

Servicio	Imagen base	Función en la arquitectura	Acceso / puerto
mc-server	`itzg/minecraft-server`	Servidor PaperMC (Java) con plugins `PrometheusExporter` y `SkinsRestorer`.	TCP 25565 (LAN) / 9225 (interno)
node-exporter	`prom/node-exporter`	Expone métricas del kernel Linux montando `/proc`, `/sys` y `/` en solo lectura (`:ro`).	TCP 9100 (interno)
prometheus	`prom/prometheus`	Recolecta series temporales cada 15 s y las almacena en disco por 15 días.	TCP 9090 (interno)
grafana	`grafana/grafana`	Visualiza 4 dashboards pre-aprovisionados conectados a Prometheus.	TCP 3000 (LAN)

Tabla 1: Servicios, imágenes y puertos configurados en Docker Compose.

3.2. Aislamiento de red y resolución DNS embebida

Para minimizar la superficie de exposición en la red local, únicamente se publican dos puertos hacia el exterior mediante reglas DNAT del anfitrión: **TCP 25565** para los clientes del juego y **TCP 3000** para la interfaz web de Grafana. Los puertos de telemetría (**9100** de Node Exporter, **9225** del exportador de Minecraft y **9090** de Prometheus) no poseen directiva `ports:` hacia el anfitrión, quedando completamente inaccesibles desde la red Wi-Fi e incluso desde el `localhost` de la propia máquina virtual.

La comunicación interna ocurre dentro de la red privada tipo *bridge* de Docker Compose utilizando el servidor **DNS embebido** del motor Docker (`127.0.0.11`). De esta forma, Prometheus resuelve por nombre de contenedor a `mc-server:9225` y `node-exporter:9100`, mientras que Grafana consulta a `http://prometheus:9090`, manteniendo la operatividad aunque cambien las direcciones IP privadas internas tras un reinicio.

3.3. Persistencia de datos y permisos de usuario (UID/GID)

Dado que el sistema de archivos interno de un contenedor es efímero, todos los datos de estado se desacoplan hacia el directorio `/opt/minecraft-project` del anfitrión Debian mediante montajes enlazados (*bind mounts*). Si un contenedor se detiene, se elimina o se recrea, el mundo de juego, las métricas históricas y los paneles sobreviven intactos en el disco de la máquina virtual.

Por seguridad, las imágenes oficiales no ejecutan sus procesos internos como `root` (`UID 0`), sino bajo cuentas sin privilegios. Como el contenedor y el anfitrión comparten el mismo kernel Linux, el script de configuración asigna explícitamente la propiedad numérica (`chown`) de cada directorio en el host para coincidir con el usuario interno de cada contenedor:

- `1000:1000` sobre `mc-data/` para el usuario interno del contenedor `mc-server`.
- `65534:65534` sobre `prometheus-data/` para el usuario `nobody` de Prometheus.
- `472:472` sobre `grafana-data/` y `grafana-provisioning/` para el usuario `grafana`.

4. Implementación y automatización (8 Scripts)

Toda la infraestructura se centraliza en `/opt/minecraft-project`. El diseño se divide en dos grupos de scripts: **cuatro scripts de aprovisionamiento** (que transforman un sistema Debian limpio en una plataforma funcional) y **cuatro scripts de operación diaria** (que simplifican la administración del servicio en producción).

4.1. Fase 1 de aprovisionamiento: `dependencia.sh`

Prepara el sistema operativo base actualizando repositorios APT e instalando herramientas de transferencia, empaquetado y planificación (`curl`, `wget`, `ca-certificates`, `gnupg`, `cron` y `tar`). Verifica de forma idempotente si Docker y el plugin Docker Compose v2 están presentes; de no estarlo, los instala e incluye un mecanismo de respaldo que descarga el binario oficial desde GitHub según la arquitectura del procesador (`x86_64` o `aarch64`). Finalmente, habilita los demonios `docker` y `cron` en `systemd` (`systemctl enable -now`) para garantizar el arranque automático ante reinicios de la máquina virtual.

Listing 1: Script 1: dependencia.sh (Preparación del sistema operativo y Docker).

```
#!/bin/bash
set -e
echo "=== [1/4] Instalando dependencias del sistema y Docker ==="

apt-get update -y
apt-get install -y curl wget ca-certificates gnupg lsb-release cron tar

if ! command -v docker &> /dev/null; then
    curl -fsSL https://get.docker.com -o get-docker.sh
    sh get-docker.sh
    rm -f get-docker.sh
fi

if ! docker compose version &> /dev/null; then
    apt-get update -y
    apt-get install -y docker-compose-plugin || {
        DOCKER_CONFIG=${DOCKER_CONFIG:-/usr/local/lib/docker}
        mkdir -p $DOCKER_CONFIG/cli-plugins
        ARCH=$(uname -m)
        curl -SL "https://github.com/docker/compose/releases/latest/download/docker-compose-linux-${ARCH}" \
             -o $DOCKER_CONFIG/cli-plugins/docker-compose
        chmod +x $DOCKER_CONFIG/cli-plugins/docker-compose
    }
fi

systemctl enable --now docker
systemctl enable --now cron
```

4.2. Fase 2 de aprovisionamiento: `configuracion.sh`

Constituye el núcleo de aprovisionamiento declarativo del proyecto y realiza cinco tareas automatizadas:

1. Crea el árbol de directorios persistentes (`mc-data/plugins`, `prometheus-data`, `grafana-data`, `backups` y `grafana-provisioning`), donde vivirán los archivos de datos del proyecto.
2. Genera el archivo maestro `prometheus.yml` con un intervalo de recolección de 15 s hacia tres objetivos (`prometheus`, `node-exporter:9100` y `mc-server:9225`), inyectando la etiqueta `server_name: 'mc-server'` necesaria para que funcionen las variables dinámicas de los paneles de jugadores.

3. Configura el aprovisionamiento automático de Grafana generando el origen de datos (`prometheus-ds.yml`), el proveedor de paneles (`dashboards.yml`) y descargando los cuatro archivos JSON (*Minecraft General*, *Minecraft Server*, *Minecraft Players* y *Node Exporter Full ID 1860*), reemplazando automáticamente la variable `${DS_-PROMETHEUS}` por el identificador `prometheus`.

4. Consulta la API de GitHub para descargar las últimas versiones de los plugins `PrometheusExporter` (configurando su `config.yml` en el puerto `9225` con métricas de memoria JVM, recolector de basura, hilos, entidades, chunks y duración de ticks) y `SkinsRestorer` (permitiendo visualizar apariencias personalizadas aun operando en modo local `ONLINE_MODE=FALSE`).

5. Aplica los permisos de propietario (`chown`) correspondientes a cada contenedor, para evitar errores por falta de permisos.

Listing 2: Script 2: configuracion.sh (Aprovisionamiento de métricas, dashboards, plugins y permisos).

```
#!/bin/bash
set -e
PROJECT_DIR="/opt/minecraft-project"
mkdir -p "$PROJECT_DIR"/{mc-data/plugins,prometheus-data,grafana-data,backups}
mkdir -p "$PROJECT_DIR"/grafana-provisioning/{datasources,dashboards/json}
cd "$PROJECT_DIR"

cat << 'EOF' > "$PROJECT_DIR/prometheus.yml"
global:
  scrape_interval: 15s
  evaluation_interval: 15s
scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']
  - job_name: 'node_exporter'
    static_configs:
      - targets: ['node-exporter:9100']
  - job_name: 'minecraft'
    static_configs:
      - targets: ['mc-server:9225']
        labels:
          server_name: 'mc-server'
EOF

cat << 'EOF' > "$PROJECT_DIR/grafana-provisioning/datasources/prometheus-ds.yml"
apiVersion: 1
datasources:
  - name: Prometheus
    type: prometheus
    uid: prometheus
    access: proxy
    url: http://prometheus:9090
    isDefault: true
    editable: true
EOF

cat << 'EOF' > "$PROJECT_DIR/grafana-provisioning/dashboards/dashboards.yml"
apiVersion: 1
providers:
  - name: 'Minecraft Dashboards'
    orgId: 1
    folder: 'Minecraft'
    type: file
    disableDeletion: false
    updateIntervalSeconds: 30
    options:
      path: /etc/grafana/provisioning/dashboards/json
EOF

DASHBOARD_BASE_URL="https://raw.githubusercontent.com/sladkoff/minecraft-prometheus-exporter/master/dashboards"
for db in minecraft-general-dashboard.json minecraft-server-dashboard.json minecraft-players-dashboard.json; do
    curl -fsSL "${DASHBOARD_BASE_URL}/${db}" -o "$PROJECT_DIR/grafana-provisioning/dashboards/json/${db}" || true
done
curl -fsSL "https://grafana.com/api/dashboards/1860/revisions/37/download" \
     -o "$PROJECT_DIR/grafana-provisioning/dashboards/json/node-exporter-full.json" || true

sed -i 's/\${DS_PROMETHEUS}/prometheus/g' "$PROJECT_DIR"/grafana-provisioning/dashboards/json/*.json 2>/dev/null || true

EXPORTER_URL=$(curl -fsSL https://api.github.com/repos/sladkoff/minecraft-prometheus-exporter/releases/latest \
  | grep -o 'https://github.com/sladkoff/minecraft-prometheus-exporter/releases/download/[^"]*\.jar' | head -n 1)
curl -fsSL "$EXPORTER_URL" -o "$PROJECT_DIR/mc-data/plugins/minecraft-prometheus-exporter.jar"

SKINSRESTORER_URL=$(curl -fsSL https://api.github.com/repos/SkinsRestorer/SkinsRestorer/releases/latest \
  | grep -o 'https://github.com/SkinsRestorer/SkinsRestorer/releases/download/[^"]*/SkinsRestorer\.jar' | head -n 1)
curl -fsSL "$SKINSRESTORER_URL" -o "$PROJECT_DIR/mc-data/plugins/SkinsRestorer.jar"

mkdir -p "$PROJECT_DIR/mc-data/plugins/PrometheusExporter"
cat << 'EOF' > "$PROJECT_DIR/mc-data/plugins/PrometheusExporter/config.yml"
```

```
host: 0.0.0.0
port: 9225
enable_metrics:
  jvm_threads: true
  jvm_gc: true
  jvm_memory: true
  players_online_total: true
  player_online: true
  entities_total: true
  loaded_chunks_total: true
  tps: true
  tick_duration_seconds: true
  tick_duration_average: true
  tick_duration_min: true
  tick_duration_max: true
  player_statistic: true
EOF

chown -R 1000:1000 "$PROJECT_DIR/mc-data"
chown -R 65534:65534 "$PROJECT_DIR/prometheus-data"
chown -R 472:472 "$PROJECT_DIR/grafana-data" "$PROJECT_DIR/grafana-provisioning"
```

4.3. Fase 3 de aprovisionamiento: `despliegue.sh`

Genera el archivo `docker-compose.yml`, descarga las imágenes y levanta el stack definido en segundo plano. Configura el motor `PAPER` (una implementación optimizada que permite gestionar el servidor de Minecraft e incorporar plugins), asigna 2 GB de memoria al Heap de Java (`MEMORY: "2G"`), habilita `ONLINE_MODE: "FALSE"` para conexiones locales en LAN y aplica la política `restart: unless-stopped` en todos los contenedores.

Listing 3: Script 3: despliegue.sh (Generación de Docker Compose y arranque de servicios).

```
#!/bin/bash
set -e
PROJECT_DIR="/opt/minecraft-project"
cd "$PROJECT_DIR"

cat << 'EOF' > "$PROJECT_DIR/docker-compose.yml"
services:
  mc-server:
    image: itzg/minecraft-server:latest
    container_name: mc-server
    restart: unless-stopped
    ports:
      - "25565:25565"
    environment:
      EULA: "TRUE"
      TYPE: "PAPER"
      MEMORY: "2G"
      ONLINE_MODE: "FALSE"
      MOTD: "Servidor Minecraft - Proyecto Admin de Redes"
      MAX_PLAYERS: "20"
      VIEW_DISTANCE: "10"
    volumes:
      - ./mc-data:/data

  node-exporter:
    image: prom/node-exporter:latest
    container_name: node-exporter
    restart: unless-stopped
    command:
      - '--path.procfs=/host/proc'
      - '--path.sysfs=/host/sys'
      - '--path.rootfs=/rootfs'
    volumes:
      - /proc:/host/proc:ro
      - /sys:/host/sys:ro
      - /:/rootfs:ro

  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    restart: unless-stopped
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.path=/prometheus'
      - '--storage.tsdb.retention.time=15d'
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml:ro
      - ./prometheus-data:/prometheus
    depends_on:
      - mc-server
      - node-exporter

  grafana:
    image: grafana/grafana:latest
```

```
    container_name: grafana
    restart: unless-stopped
    ports:
      - "3000:3000"
    environment:
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=admin
    volumes:
      - ./grafana-data:/var/lib/grafana
      - ./grafana-provisioning:/etc/grafana/provisioning
    depends_on:
      - prometheus
EOF

docker compose pull
docker compose up -d
docker compose ps
```

4.4. Fase 4 de aprovisionamiento: `backups.sh`

Crea el trabajo de respaldo `backup-job.sh`, el cual empaqueta y comprime el volumen `mc-data` en archivos `.tar.gz` fechados (`mc-backup-YYYYMMDD_HHMMSS.tar.gz`) y borra automáticamente las copias con más de 7 días de antigüedad (`find -mtime +7`). Además, registra de forma idempotente la ejecución diaria a las 03:00 AM en el `crontab` del sistema.

Listing 4: Script 4: backups.sh (Programación de respaldos diarios y política de retención).

```
#!/bin/bash
set -e
PROJECT_DIR="/opt/minecraft-project"
BACKUP_SCRIPT="$PROJECT_DIR/backup-job.sh"

cat << 'EOF' > "$BACKUP_SCRIPT"
#!/bin/bash
PROJECT_DIR="/opt/minecraft-project"
BACKUP_DIR="$PROJECT_DIR/backups"
TIMESTAMP=$(date +"%Y%m%d_%H%M%S")
BACKUP_FILE="$BACKUP_DIR/mc-backup-$TIMESTAMP.tar.gz"

mkdir -p "$BACKUP_DIR"
tar -czf "$BACKUP_FILE" -C "$PROJECT_DIR" mc-data
find "$BACKUP_DIR" -name "mc-backup-*.tar.gz" -type f -mtime +7 -exec rm -f {} \;
EOF

chmod +x "$BACKUP_SCRIPT"
CRON_JOB="0 3 * * * $BACKUP_SCRIPT >> $PROJECT_DIR/backups/backup.log 2>&1"
( crontab -l 2>/dev/null | grep -v -F "$BACKUP_SCRIPT" || true ; echo "$CRON_JOB" ) | crontab -
"$BACKUP_SCRIPT"
```

4.5. Scripts de operación y administración diaria (Día 2)

Para evitar la escritura manual de comandos extensos de Docker durante la operación cotidiana, se implementaron cuatro scripts adicionales:

- `prender.sh`: Inicia o reanuda todos los contenedores de la pila en segundo plano (`docker compose up -d`).
- `apagar.sh`: Realiza un apagado ordenado mediante `docker compose stop`. Envía la señal `SIGTERM` al contenedor `mc-server`, permitiendo que PaperMC desconecte a los jugadores, vuelque a disco los chunks pendientes en memoria RAM y cierre limpiamente las bases de datos de Prometheus y Grafana sin destruir los contenedores.
- `estado.sh`: Muestra en una sola vista el estado de ejecución y mapeo de puertos (`docker compose ps`) seguido del consumo instantáneo de CPU y memoria RAM real de cada contenedor (`docker stats -no-stream`).
- `consola.sh`: Abre una terminal interactiva contra el servidor de juego mediante el protocolo RCON interno (`docker exec -i mc-server rcon-cli`), facilitando la asignación de operadores (`op`) o ajustes del mundo sin detener el servicio.

Listing 5: Scripts 5, 6, 7 y 8: prender.sh, apagar.sh, estado.sh y consola.sh.

```
# --- Script 5: prender.sh ---
#!/bin/bash
cd /opt/minecraft-project || exit
docker compose up -d

# --- Script 6: apagar.sh ---
#!/bin/bash
```

```
cd /opt/minecraft-project || exit
docker compose stop

# --- Script 7: estado.sh ---
#!/bin/bash
cd /opt/minecraft-project || exit
echo "=== ESTADO DE LOS CONTENEDORES ==="
docker compose ps
echo ""
echo "=== CONSUMO ACTUAL (RAM/CPU) ==="
docker stats --no-stream --format "table {{.Name}}\t{{.CPUPerc}}\t{{.MemUsage}}"

# --- Script 8: consola.sh ---
#!/bin/bash
docker exec -i mc-server rcon-cli
```

5. Monitoreo y administración del servicio

El diseño considera dos niveles complementarios de observabilidad. **Node Exporter** proporciona información del sistema anfitrión Linux (uso de CPU por modo, memoria RAM física y swap, carga del sistema y tráfico de red en las interfaces), visualizada en el panel *Node Exporter Full*. Por su parte, el plugin **PrometheusExporter** extrae métricas internas de la Máquina Virtual de Java (uso del Heap, recolecciones de basura y hebras activas) y del motor PaperMC (jugadores en línea por dimensión, entidades activas, chunks cargados y duración de los ticks).

Prometheus recopila estas series temporales mediante solicitudes HTTP periódicas (*scraping*) cada 15 s y Grafana permite consultarlas mediante PromQL. Para evaluar la estabilidad de la simulación se utiliza la frecuencia nominal del bucle principal de Minecraft de 20 ticks por segundo (TPS), equivalente a un presupuesto máximo de tiempo por ciclo de 50 ms (0,05 s):

$$T_{\text{presupuesto}} = \frac{1000 \text{ ms}}{20 \text{ ticks}} = 50 \text{ ms/tick} = 0{,}05 \text{ s/tick}. \tag{1}$$

Si la carga computacional de un ciclo (por acumulación de entidades, generación de chunks en múltiples dimensiones o explosiones en cadena) hace que el tiempo de procesamiento supere los 50 ms, el servidor no puede acelerar ciclos futuros para compensar el atraso y la tasa efectiva de TPS cae por debajo de 20, manifestándose como retraso (*lag*) observable por los jugadores.

6. Pruebas de funcionamiento y resultados esperados

Las pruebas contemplan la validación del despliegue de contenedores, la conectividad de los clientes en LAN, la visualización de los paneles de telemetría y la generación de respaldos. Cada verificación se documenta en la sección siguiente con capturas reales.

Prueba	Criterio de verificación
Servicios Docker	Los cuatro contenedores aparecen activos (`Up`) y con métricas de consumo mediante `./estado.sh` / `docker compose ps`.
Acceso Minecraft	Un cliente de la misma LAN logra descubrir e ingresar a `IP_VM:25565`.
Conexión simultánea	Múltiples clientes acceden al mundo con skins visibles (`SkinsRestorer`).
Grafana	El navegador muestra la interfaz en `IP_VM:3000` con los 4 dashboards aprovisionados.
Telemetría	Se observan series temporales tanto de Minecraft (JVM, TPS, chunks) como del host Linux (CPU, RAM, red).
Prueba de carga	El desplazamiento entre dimensiones o detonación controlada de TNT refleja variaciones en chunks, entidades y duración de tick.
Respaldo	La ejecución del script genera un archivo comprimido `.tar.gz` en `/opt/minecraft-project/backups`.

Tabla 2: Plan de verificación funcional del proyecto.

7. Evidencias de implementación y funcionamiento

Las figuras de esta sección documentan la ejecución real de la plataforma. Se mantiene el encuadre del comando ejecutado, la dirección IP local utilizada y los paneles de observabilidad en funcionamiento.

7.1. Estado de los servicios

Se muestra la salida de `./estado.sh` (`docker compose ps` y `docker stats`) en Debian, donde se identifica el estado `Up` de los contenedores `mc-server`, `node-exporter`, `prometheus` y `grafana`, junto con su consumo real de memoria RAM y CPU.

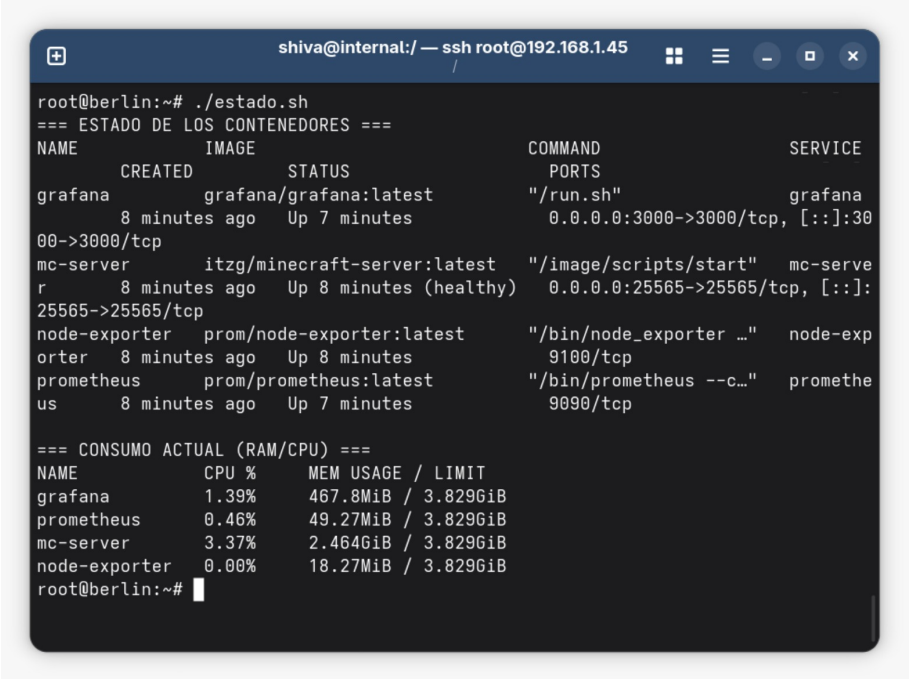


Figura 2: Estado y consumo de recursos de los servicios Docker tras el despliegue de la plataforma.

7.2. Conexión del cliente al servidor

Se presenta la conexión desde el cliente Minecraft apuntando a la dirección IP local de la máquina virtual mediante el puerto TCP 25565, que no es necesario especificar pues es el puerto por defecto de Minecraft.



Figura 3: Conexión de un cliente Minecraft al servidor publicado en la red local.

7.3. Participación de jugadores y carga de skins

Se muestra el ingreso al mundo de Minecraft verificando la sesión activa en la red local y la correcta aplicación de apariencias mediante el plugin `SkinsRestorer`.



Figura 4: Sesión activa en el servidor Minecraft dentro de la red local.

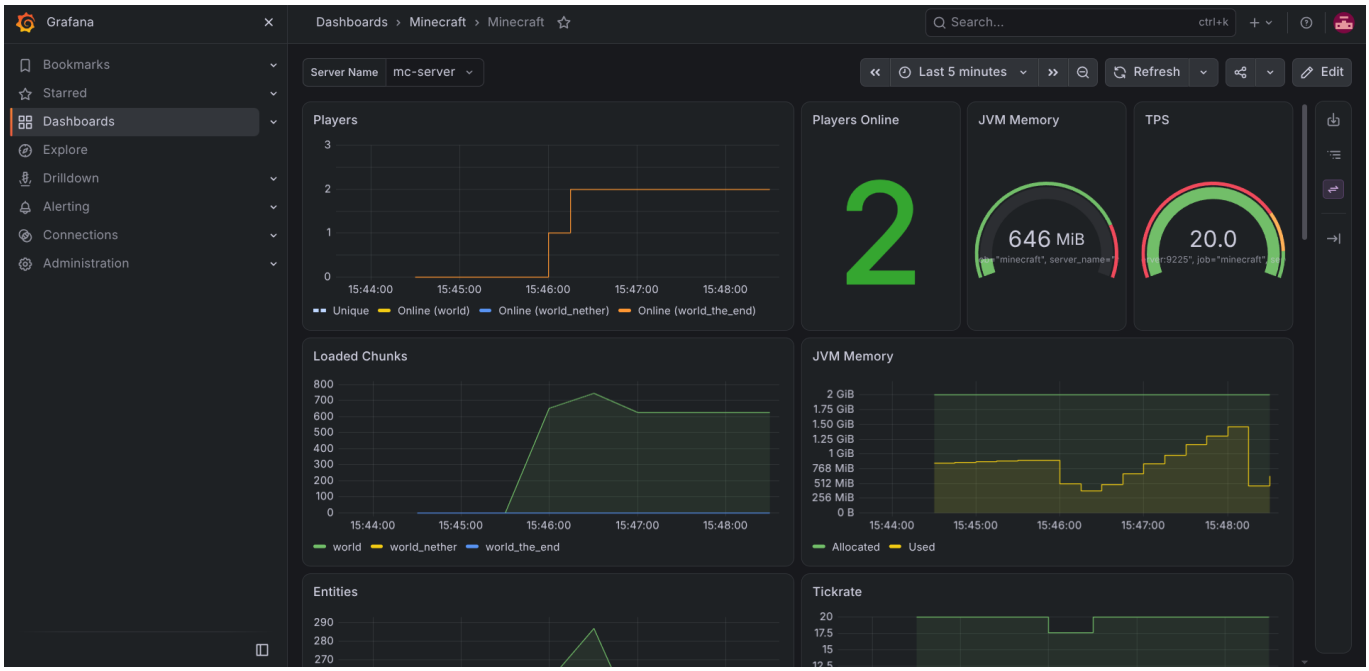


Figura 5: Actualizacion en el dashboard con 2 jugadores

7.4. Verificación interna de salud y objetivos en Prometheus

Se muestra la consulta interna al contenedor de Prometheus verificando su estado saludable (`Healthy`) y la disponibilidad de recolección (`up`) de los exportadores sin exponer el puerto 9090 al anfitrión.

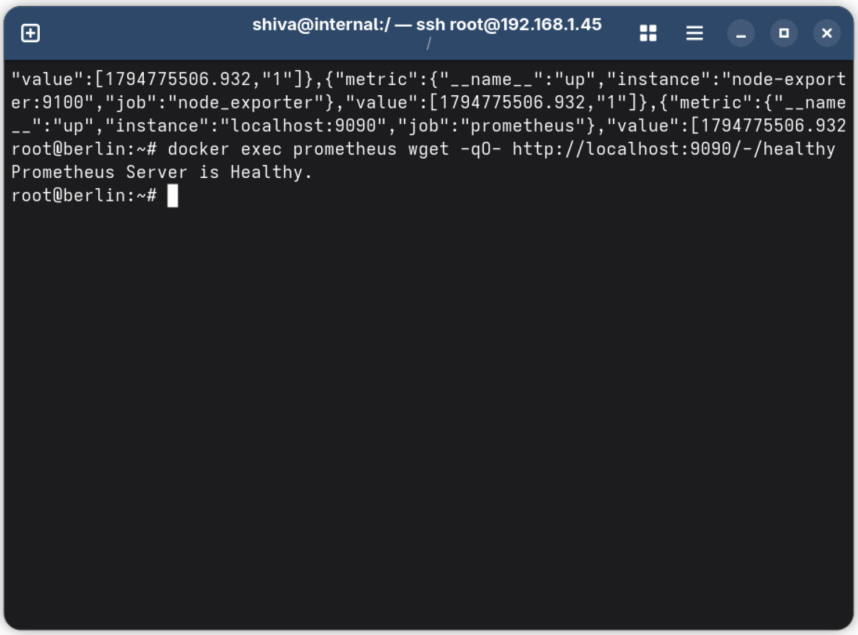


Figura 6: Comprobación de salud y objetivos activos directamente desde el contenedor Prometheus.

7.5. Disponibilidad de paneles aprovisionados

Se muestra el directorio de paneles en Grafana confirmando el aprovisionamiento automático de los cuatro tableros configurados para Minecraft y el sistema anfitrión.

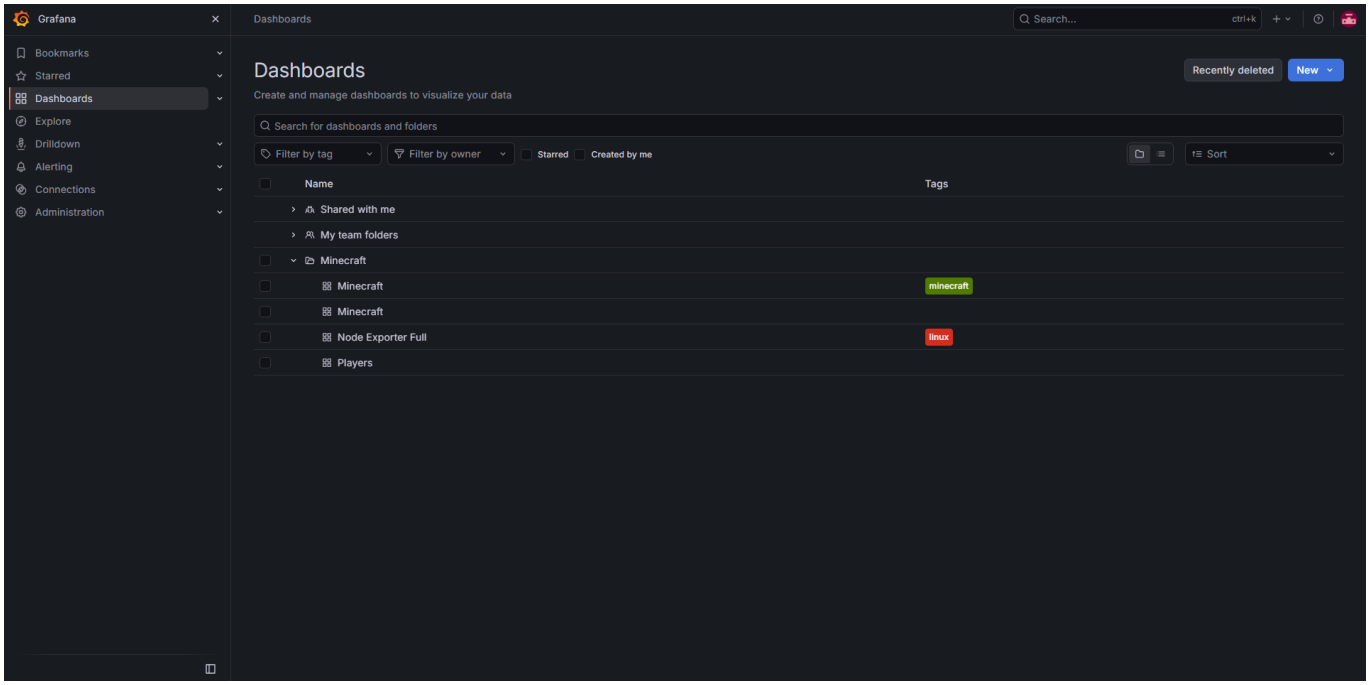


Figura 7: Paneles aprovisionados automáticamente y alimentados por Prometheus.

7.6. Visualización de Grafana (Métricas de Minecraft)

Se presenta la interfaz web de Grafana con el dashboard de Minecraft mostrando jugadores en línea, uso de memoria de la JVM (1,33 GiB), tasa de 20,0 TPS, chunks cargados y entidades.

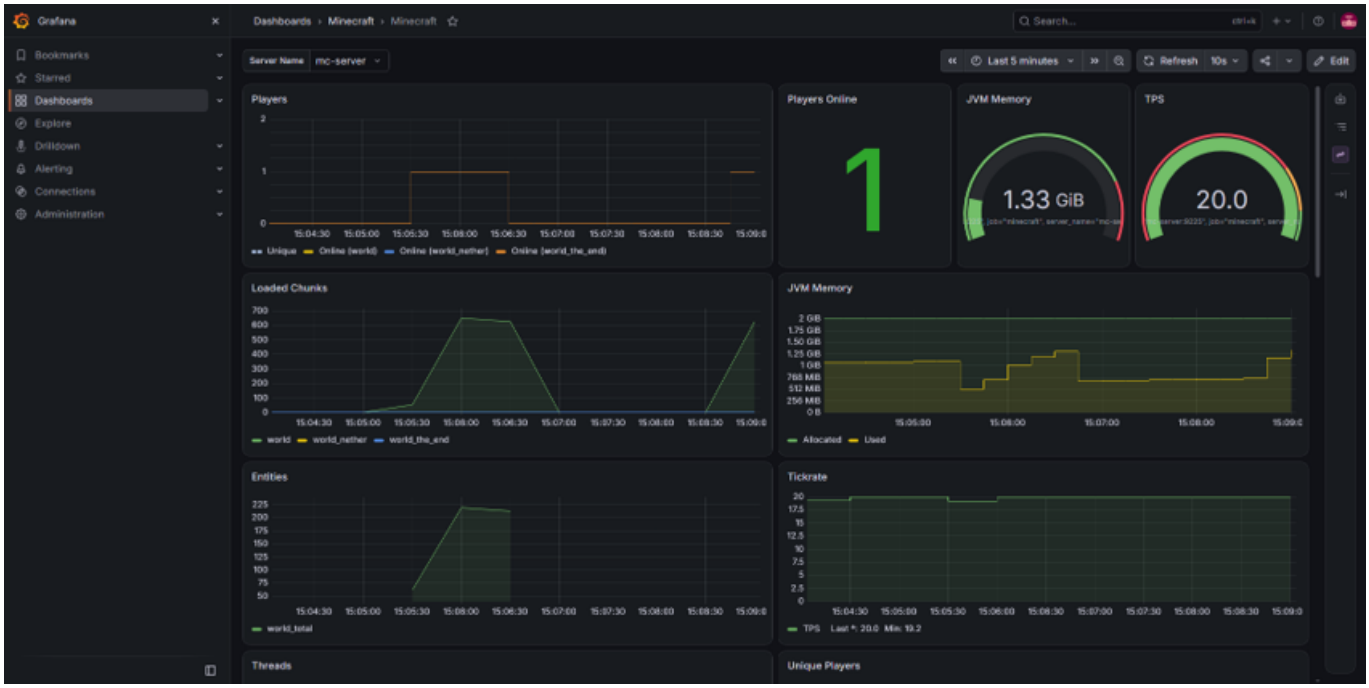


Figura 8: Dashboard de Minecraft disponible desde la interfaz web de Grafana.

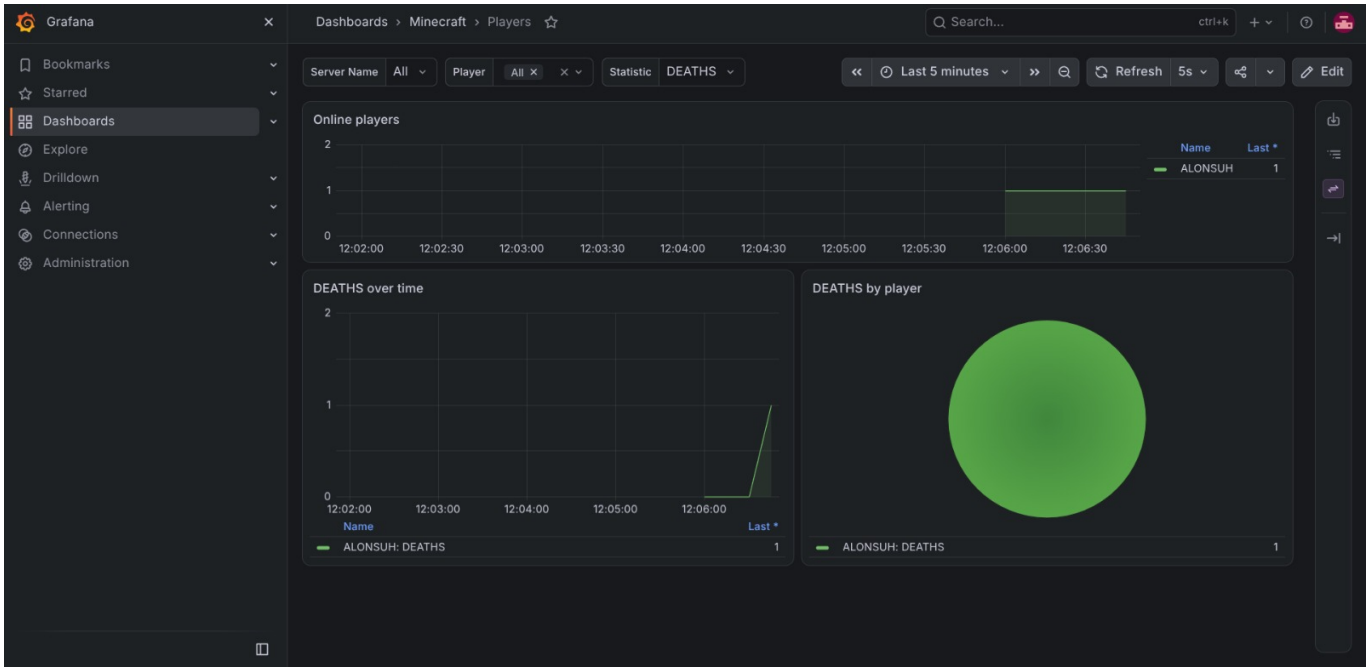


Figura 9: Dashboard estadísticas de usuarios disponible desde la interfaz web de Grafana.

7.7. Indicadores del sistema Linux

Se muestra el dashboard *Node Exporter Full*, utilizado para consultar indicadores de CPU (4 núcleos), memoria RAM (4 GiB), espacio en disco y tráfico de red de la máquina virtual Debian.

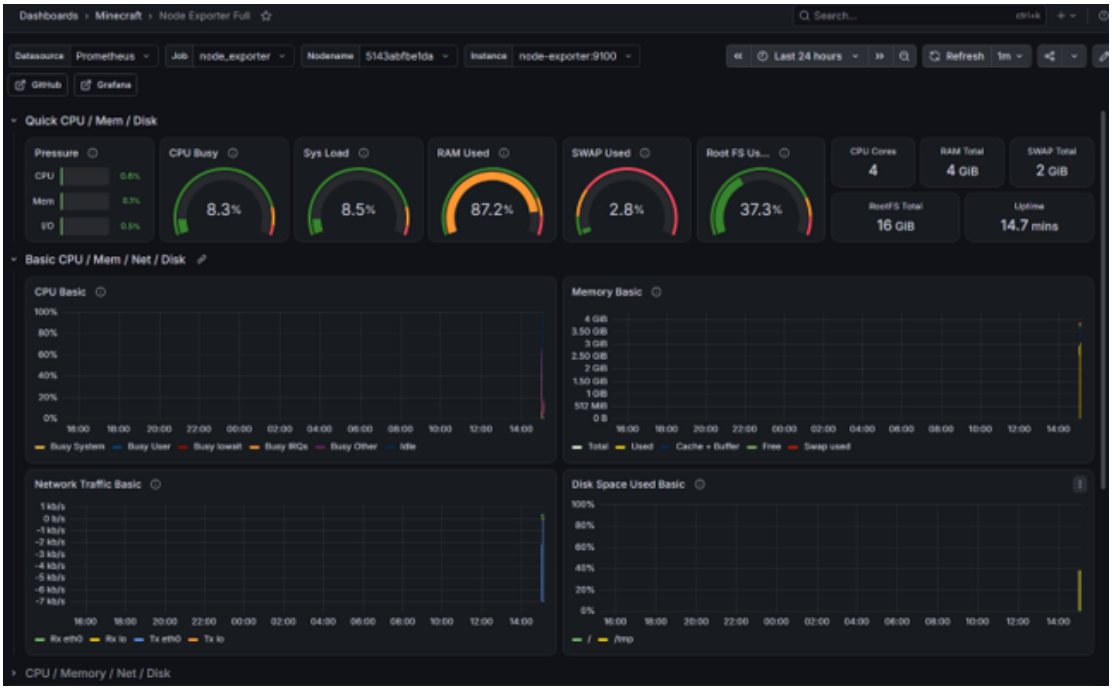


Figura 10: Panel de monitoreo de los recursos del host Linux mediante Node Exporter.

7.8. Comportamiento durante actividad en el servidor

Se incorpora la evolución temporal registrada en Grafana ante cambios de carga de chunks, entidades y memoria dentro del servidor de juego. En específico, se detonó una gran cantidad de TNT para generar múltiples eventos simultáneos en pocos segundos, saturando la capacidad de procesamiento y repercutiendo en una caída de los TPS (ya que el hilo principal demora más tiempo en resolver cada tick al registrar las modificaciones del mundo).

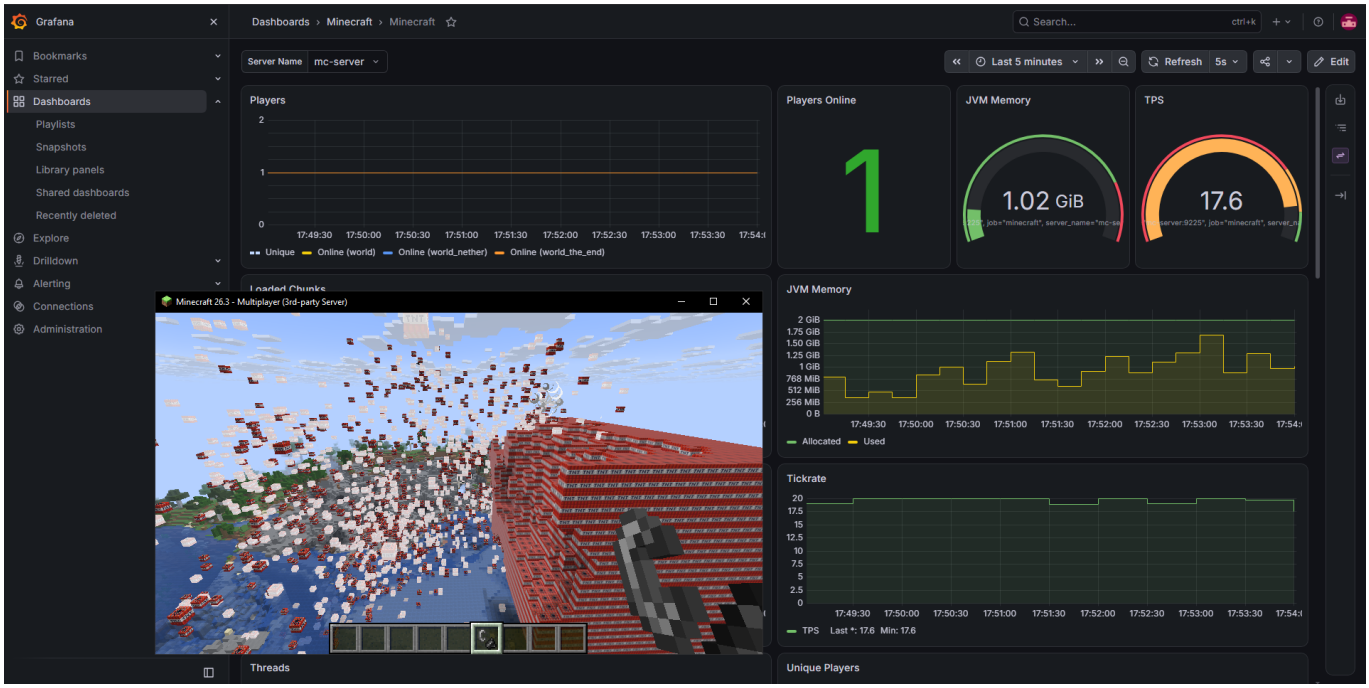


Figura 11: Variación de indicadores del servidor registrada en las gráficas de series temporales.

7.9. Generación del respaldo

Se presenta la terminal con el listado del directorio `/opt/minecraft-project/backups`, evidenciando la creación del archivo comprimido `mc-backup-YYYYMMDD_HHMMSS.tar.gz` junto con su tamaño y marca de tiempo.

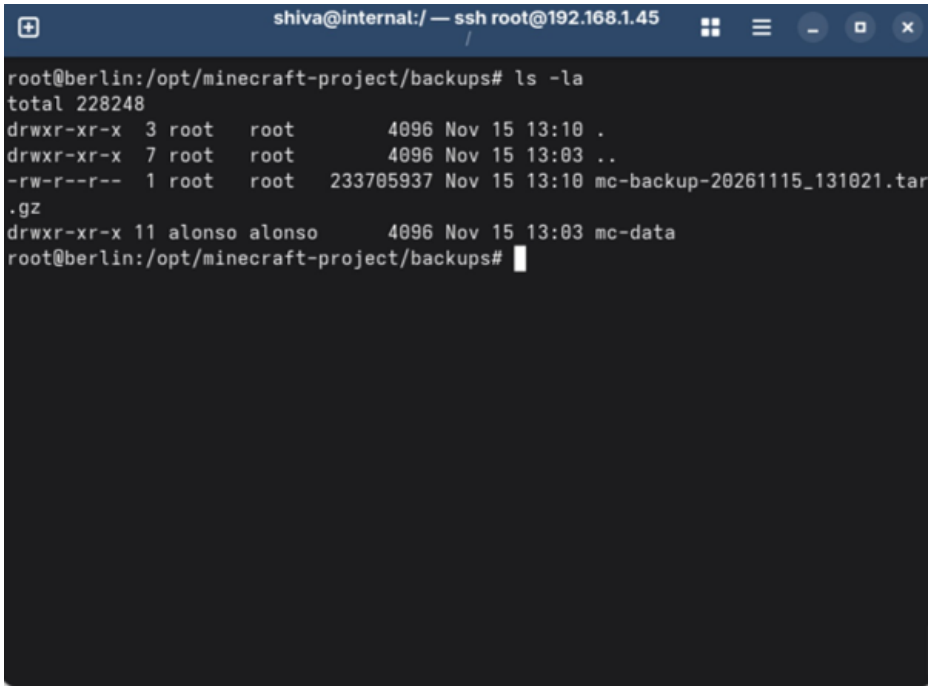


Figura 12: Archivo comprimido generado mediante el procedimiento automatizado de respaldo del mundo.

8. Consideraciones de seguridad y limitaciones

La solución se ha diseñado como una demostración operativa en una red local controlada. La configuración `ONLINE_MODE=FALSE` permite operar en una LAN aislada sin depender de los servidores externos de autenticación de Microsoft/Mojang (complementada visualmente con `SkinsRestorer`), pero facilita la suplantación de nombres de usuario si no se acompaña de un plugin de autenticación local por contraseña en redes no confiables.

Asimismo, el archivo de despliegue define credenciales administrativas iniciales (`admin/admin`) para Grafana y utiliza etiquetas de imagen `latest`. En un entorno de producción expuesto, es necesario gestionar credenciales mediante variables de entorno externas (`.env`), fijar versiones inmutables de imágenes (*pinning*) e incorporar comandos RCON (`save-off`, `save-all` y `save-on`) dentro de `backup-job.sh` para garantizar respaldos atómicos bajo alta concurrencia de escritura.

9. Conclusiones

El proyecto traslada un caso conocido, como alojar un mundo de Minecraft para una partida en Wi-Fi, a un escenario riguroso de administración de redes y sistemas. La solución implementada integra aprovisionamiento automatizado en Bash, aislamiento de red mediante contenedores Docker, persistencia de estado con control de permisos `UID/GID`, observabilidad multicapa en tiempo real con Prometheus y Grafana, y continuidad operativa mediante respaldos programados en `cron` y scripts de gestión diaria.

La principal contribución consiste en transformar un despliegue manual propenso a errores en una arquitectura reproducible en minutos y completamente observable. De esta forma, Minecraft funciona como un banco de pruebas representativo para aplicar metodologías de automatización, seguridad de red y monitoreo transferibles a cualquier servicio crítico de infraestructura.