UNIVERSITÀ DEGLI STUDI DI CAGLIARI

FACOLTÀ DI SCIENZE

Corso di Laurea Magistrale in Informatica

Progetto BIG DATA: Creazione di un cluster in Microk8s

Studenti
Nicola Figus
Francesco Pasqui

Professore
Diego Reforgiato Recupero

A.A 2025/2026

Indice

Capitolo 1

Requisiti

Di seguito elencati i requisiti di sistema per la creazione del cluster:
Si raccomanda sufficiente memoria RAM e core di CPU per le operazioni del cluster.

1.1 Setup del Nodo Master

- Computer con architettura **ARM64 (aarch64)**, dunque qualunque PC Apple con CPU da M1 in poi
  - Virtual Machine con Sistema Operativo **Ubuntu 25.04**
  - Configurazione di rete della VM in modalità **Bridge**

1.2 Setup dei Nodi Slave

- Uno o più PC (qualsiasi architettura) per la configurazione multinodo
  - Virtual Machine con Sistema Operativo **Ubuntu 25.04**
  - Configurazione di rete della VM in modalità **Bridge**

Capitolo 2

Guida Operativa

Questo capitolo fornisce una guida completa all'installazione e alla configurazione di un ambiente Kubernetes locale basato su **MicroK8s**, integrato con **MinIO** come sistema di storage compatibile con S3 e **Apache Spark** per l'elaborazione distribuita. Oltre alla sequenza di comandi, ogni sezione include approfondimenti teorici sulle motivazioni e implicazioni delle operazioni svolte, al fine di garantire una comprensione consapevole delle scelte architetturali.

2.1 Preparazione dell'Ambiente di Lavoro

Questa fase è volta a predisporre un sistema operativo pulito e stabile, aggiornato nelle librerie di base e pronto all'esecuzione di container e servizi Kubernetes. La disabilitazione dello swap, l'installazione di tool diagnostici e il riavvio finale assicurano stabilità e compatibilità del sistema con MicroK8s.

Aggiornamento Librerie Il comando sotto riportato aggiorna i pacchetti del sistema, assicurando la compatibilità con le dipendenze richieste da Kubernetes e Snap, in modo da prevenire conflitti dovuti a versioni obsolete di librerie di sistema.

```
# Aggiornamenti Librerie
sudo apt update && sudo apt -y upgrade
```

Disabilitazione Swap Kubernetes impone che la memoria swap sia disabilitata per evitare che il kernel sposti in memoria virtuale parti dei processi dei pod, compromettendo la prevedibilità delle prestazioni. La rimozione permanente dallo `/etc/fstab` evita che lo swap venga riattivato dopo il riavvio.

```
# Disabilitazione dello swap (Kubernetes lo richiede)
sudo swapoff -a
sudo sed -i.bak '/\sswap\s/d' /etc/fstab
```

Altre installazioni e riavvio macchina :

```
# Installazione di Tool Utili
sudo apt -y install curl wget git unzip jq net-tools ca-certificates

# Riavvio della macchina (consigliato)
sudo reboot
```

2.2 Configurazione Operativa

2.2.1 Installazione e Setup di microk8s

MicroK8s è una distribuzione Kubernetes leggera fornita come pacchetto Snap, che offre un cluster completo in un singolo nodo. L'opzione `-classic` consente al servizio di operare senza le restrizioni del confinement Snap, necessario per l'accesso ai socket di rete e ai percorsi del filesystem.

```
# Installazione MicroK8s
sudo snap install microk8s --classic
```

Creazione Privilegi Utente MicroK8s crea un gruppo dedicato per gestire i permessi d'accesso al demone Kubernetes interno. Aggiungendo l'utente a tale gruppo, si consente l'interazione con il cluster senza dover ricorrere a privilegi amministrativi.

```
# Privilegi utente per utilizzarlo senza sudo
sudo usermod -aG microk8s $USER
mkdir -p ~/.kube
sudo chown -R $USER:$USER ~/.kube

# Riavvio sessione per applicare i privilegi utente
newgrp microk8s
```

Controllo dello stato Il cluster viene verificato fino alla completa disponibilità (`--wait-ready`).
Vengono poi abilitati:

- `dns`: necessario per la risoluzione dei nomi dei servizi all'interno del cluster;
- `storage`: fornisce uno storage locale basato su *hostpath*;
- `helm3`: gestore di pacchetti Kubernetes moderno.

Infine, vengono creati alias per usare `kubectl` e `helm` senza prefissi per semplificare l'utilizzo.

```
# Controllo dello stato - attendere finchè diventa running
microk8s status --wait-ready
microk8s enable dns
microk8s enable storage
microk8s enable helm3
sudo snap alias microk8s.kubectl kubectl
sudo snap alias microk8s.helm helm

# Assicurarsi che il nodo sia ready
kubectl get nodes
```

2.2.2 Installazione e Setup di MinIO

MinIO fornisce un servizio di object storage compatibile con l'API Amazon S3, ideale per simulare ambienti cloud in contesti locali o di ricerca. In questa sezione verranno descritti i comandi necessari all'installazione e setup di MinIO sul cluster, attraverso file di configurazione YAML.

Primo setup Si prepara una directory dedicata per mantenere i file YAML e le configurazioni di MinIO, in modo da separare logicamente le componenti del cluster.
In Kubernetes, un *namespace* isola logicamente le risorse. Questo approccio favorisce modularità e chiarezza nel deployment, utile per scenari multi-servizio come Spark + MinIO.
Il pre-pull evita ritardi nel primo deploy, scaricando preventivamente l'immagine nel registry container locale, specificando l'architettura ARM64.

```
# Creazione Cartella di MinIO
mkdir -p /home/<NOME_UTENTE>/k8s-local/minio
cd /home/<NOME_UTENTE>/k8s-local/minio

# Creazione del namespace minio
kubectl create namespace minio

# (FACOLTATIVO ma consigliato):
# Pre-pull dell'immagine da Quay per ARM64
sudo microk8s ctr -n k8s.io images pull --platform linux/arm64
quay.io/minio/minio:latest
```

dove `<NOME_UTENTE>` rappresenta lo username dell'account Linux.

File di configurazione YAML Il manifest YAML definisce i componenti fondamentali:

- `Secret`: per le credenziali di amministrazione (gestione sicura tramite variabili d'ambiente);
- `PersistentVolumeClaim`: per garantire persistenza ai dati salvati;
- `Deployment`: che specifica il container MinIO, con readiness e liveness probe per assicurare disponibilità;
- `Service`: esposto in modalità `NodePort` per consentire accesso esterno stabile.

Applicazione file di configurazione Scaricare dal seguente link il file "minio-standalone.yaml" e inserirlo dentro la cartella k8s-local/minio/.
Il comando applica il manifest YAML al cluster, generando le risorse descritte. In questo modo il cluster entra nello stato desiderato.

```
# Applica il file di configurazione
kubectl apply -f minio-standalone.yaml
```

Visualizzazione stato e informazioni di configurazione Il flag `-w` permette di osservare in tempo reale l'evoluzione dello stato del pod fino a quando risulta "Running" e "Ready 1/1". L'uso di `jsonpath` consente di estrarre informazioni strutturate direttamente dal manifest Kubernetes, utile per ottenere porte e endpoint.

```
# Visualizzazione dello stato
# Attendere finchè non diventa Running
kubectl -n minio get pods -w
```

```
# Stampa le porte configurate (console e API)
kubectl -n minio get svc minio -o jsonpath='{range
.spec.ports[*]}{.name}={.nodePort}{"\n"}{end}'

# Riepilogo completo con IP della VM
VM_IP=$(hostname -I | awk '{print $1}')
echo "MinIO Console: http://$VM_IP:30081"
echo "MinIO S3 API:  http://$VM_IP:30080"
echo "Credenziali:   minioadmin / minioadmin123"
```

2.2.3 Installazione e Setup Spark

Analogamente a MinIO, si crea un namespace dedicato. La separazione di namespace consente una gestione modulare delle applicazioni e un isolamento funzionale.
Il pre-pull dell'immagine ufficiale di Spark garantisce che l'immagine Spark corretta per l'architettura locale sia disponibile immediatamente.

```
# Creazione cartella spark
cd
mkdir -p k8s-local/spark
cd k8s-local/spark

# crea il namespace
kubectl create namespace spark

# PRE-PULL dell'immagine ufficiale Spark (ARM64)
sudo microk8s ctr -n k8s.io images pull --platform linux/arm64
docker.io/library/spark:3.5.1
```

File di configurazione YAML Il file YAML definisce due entità principali:

- `StatefulSet` per il master (garantisce identità stabile del nodo principale);
- `Deployment` per i worker (scalabilità dinamica).

Questa architettura segue il paradigma master–worker, con comunicazione interna su porte dedicate (`7077` e `8080`), conforme al design distribuito di Spark.

Applicazione configurazione e controllo stato Scaricare dal seguente link il file "spark-standalone.yaml" e inserirlo dentro la cartella k8s-local/spark/.
Si applica la configurazione e si osservano i pod fino al loro avvio, garantendo che l'intero cluster Spark sia operativo.

```
# Applica il manifest
kubectl apply -f spark-standalone.yaml

# Segui lo stato finché i pod sono Running
kubectl -n spark get pods -w
```

Visualizzazione informazioni di configurazione :

```
# Visualizzazione informazioni utili di configurazione
VM_IP=$(hostname -I | awk '{print $1}')
CONSOLE_PORT=$(kubectl -n minio get svc minio -o
jsonpath='{.spec.ports[?(@.name=="console")].nodePort}')
API_PORT=$(kubectl -n minio get svc minio -o
jsonpath='{.spec.ports[?(@.name=="api")].nodePort}')
```

```
echo "MinIO Console:  http://$VM_IP:$CONSOLE_PORT"
echo "MinIO S3 API:   http://$VM_IP:$API_PORT"
```

Per connettersi alla dashboard minio, è possibile utilizzare il seguente comando:

```
http://localhost:<PORTA_MINIO_CONSOLE>
```

```
# CREDENZIALI
# USER:     minioadmin
# PASSWORD: minioadmin123
```

Creazione del test-bucket su MinIO :

Apertura dashboard MinIO Per prima cosa, aprire la dashboard, digitando sul browser il seguente url:

```
http://localhost:30081
```

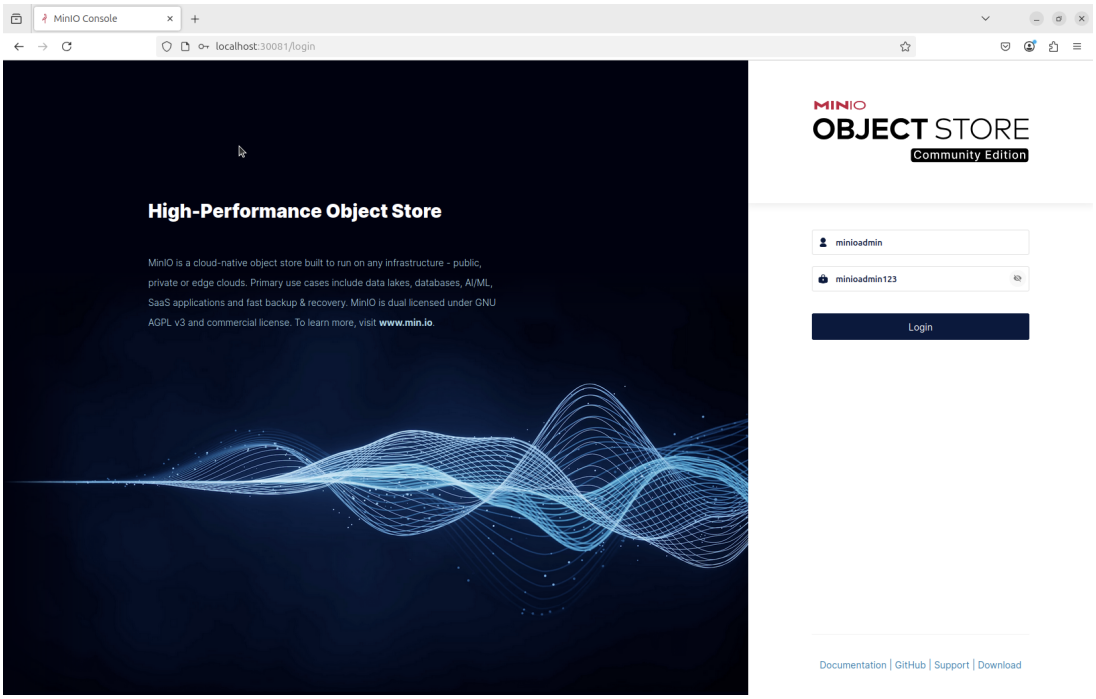

Figura 2.1: Passo 1

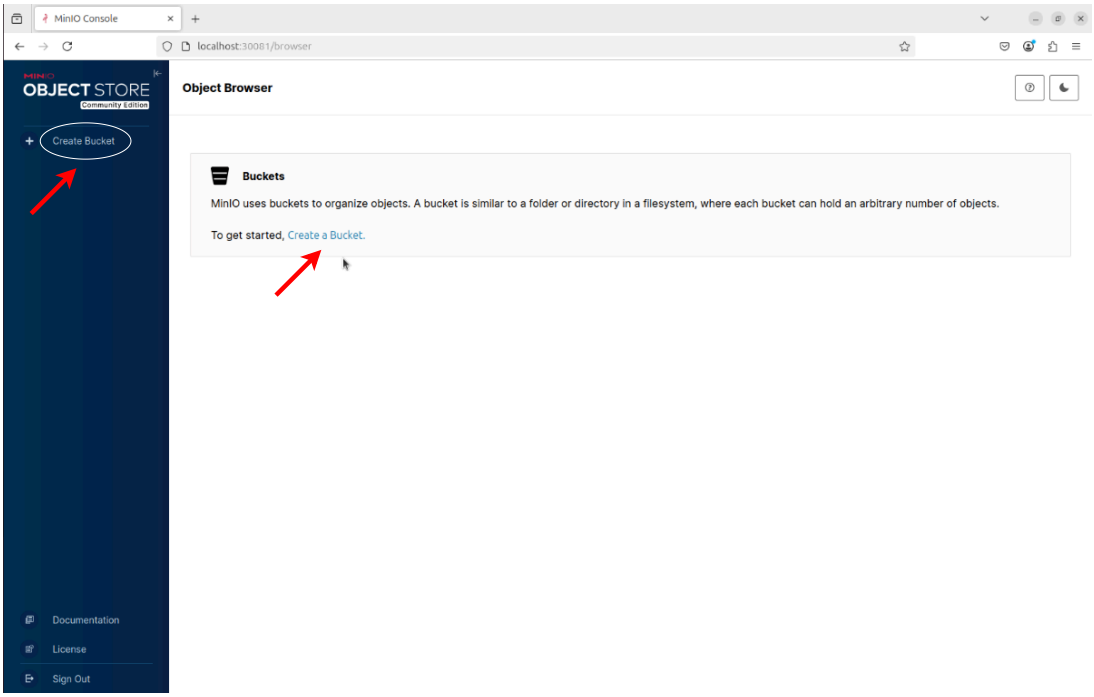

Figura 2.2: Passo 2

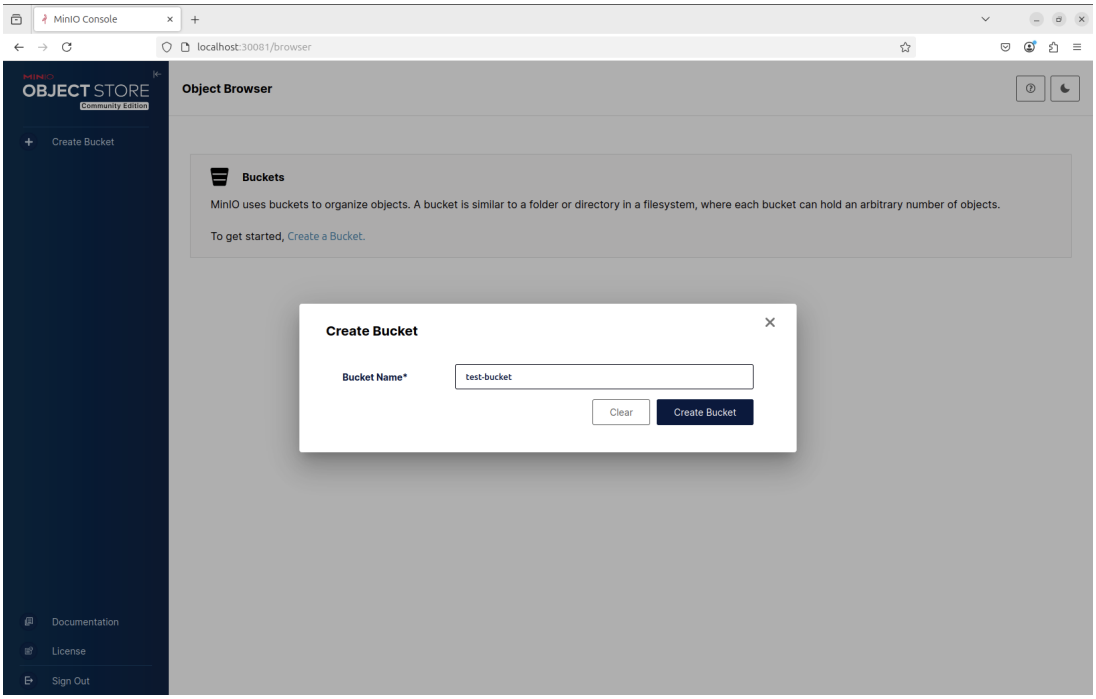

Figura 2.3: Passo 3

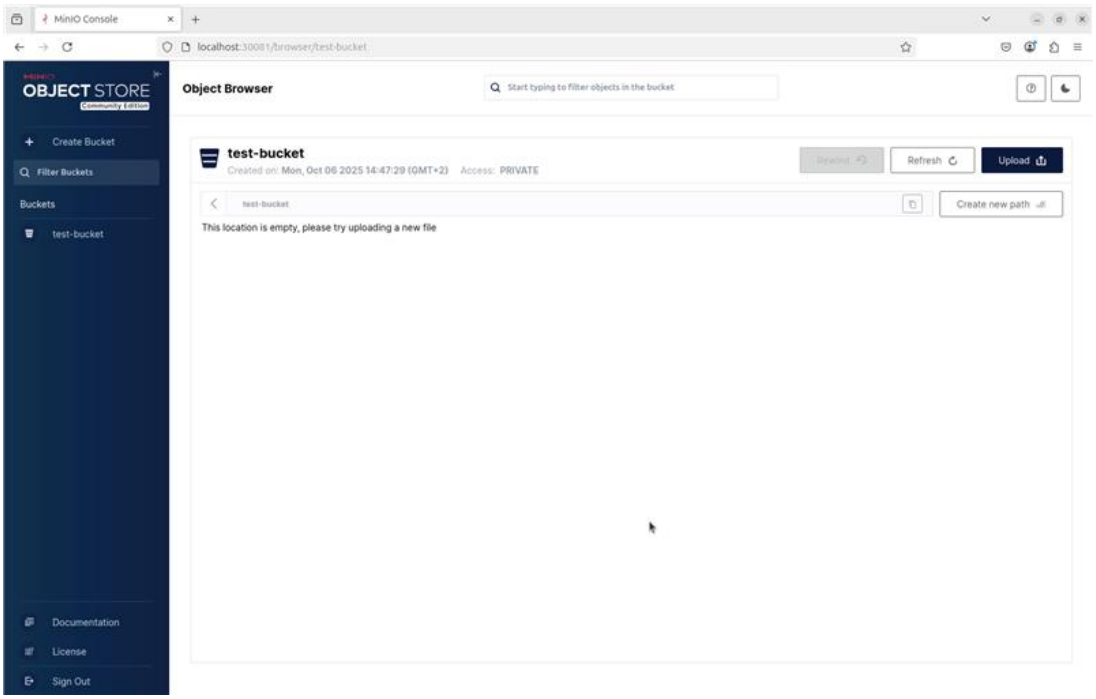

Figura 2.4: Passo 4

2.2.4 Creazione Cache Ivy

Spark utilizza Ivy per la gestione delle dipendenze client che vengono risolte dinamicamente. Creare una cache scrivibile all'interno del pod evita problemi di permessi durante la risoluzione dei pacchetti al primo avvio, migliorando la riproducibilità dell'ambiente.

```
# Cambio directory: posizionarsi su k8s-local/spark:
cd
cd k8s-local/spark

# Crea una cache Ivy scrivibile nel pod (usiamo /tmp)
kubectl -n spark exec -it spark-master-0 -- bash -lc 'mkdir -p /tmp/.ivy2
&& ls -la /tmp'
```

2.2.5 Setup e Connessione nodi al Cluster

Setup nodi worker (slave)

Da eseguire in ogni nodo slave.

```
sudo apt update && sudo apt -y upgrade
sudo swapoff -a
sudo sed -i.bak '/\sswap\s/d' /etc/fstab
sudo apt -y install curl wget git unzip jq net-tools ca-certificates
sudo snap install microk8s --classic
sudo usermod -aG microk8s $USER && newgrp microk8s
microk8s status --wait-ready
microk8s enable dns
microk8s enable storage
microk8s enable helm3
sudo snap alias microk8s.kubectl kubectl
sudo snap alias microk8s.helm helm
```

Connessione nodi al Cluster

Dal master eseguire:

```
microk8s add-node
```

Verrà generata una stringa simile:

```
microk8s join ... --worker
```

Sul secondo PC (worker), eseguire questo comando con –worker per non promuoverlo a control-plane.

2.3 Utilities

2.3.1 Comandi per l'esecuzione dei programmi

Per l'esecuzione dei programmi spark, è sufficiente:

1. Caricare le librerie di hadoop e java sul driver come indicato nella Sezione 2.3.1
2. Caricare il file python dentro al bucket di MinIO tramite la UI di MinIO
3. Eseguire il programma di riferimento come indicato nella Sezione 2.3.1

Caricamento librerie sul driver master :

```
# Da eseguire ogni volta che si riavviano i nodi
# 0) JAR per il master
kubectl -n spark exec -it spark-master-0 -- bash -lc '
  set -e
  cd /opt/spark/jars
  test -f hadoop-aws-3.3.4.jar || curl -fLO https://repo1.maven.org/mave
  n2/org/apache/hadoop/hadoop-aws/3.3.4/hadoop-aws-3.3.4.jar
  test -f aws-java-sdk-bundle-1.12.262.jar || curl -fLO
  https://repo1.maven.org/maven2/com/amazonaws/aws-java-sdk-bundle/1.12.
  262/aws-java-sdk-bundle-1.12.262.jar
'
# 1) Assicurarsi che i JAR siano presenti su *tutti* i worker
for p in $(kubectl -n spark get pods -l role=worker -o name); do
  kubectl -n spark exec -it "$p" -- bash -lc '
    set -e
    cd /opt/spark/jars
    test -f hadoop-aws-3.3.4.jar || curl -fLO https://repo1.maven.org/mav
    en2/org/apache/hadoop/hadoop-aws/3.3.4/hadoop-aws-3.3.4.jar
    test -f aws-java-sdk-bundle-1.12.262.jar || curl -fLO
    https://repo1.maven.org/maven2/com/amazonaws/aws-java-sdk-bundle/1.1
    2.262/aws-java-sdk-bundle-1.12.262.jar
  '
done
```

Esecuzione programmi :

```
cd
cd k8s-local/spark

# Esegui il job
kubectl -n spark exec -it spark-master-0 -- bash -lc '
  export HOME=/tmp
  /opt/spark/bin/spark-submit \
    --master spark://spark-master-hs.spark.svc.cluster.local:7077 \
    --conf spark.deploy.defaultCores=8 \
    --conf spark.cores.max=8 \
    --conf spark.executor.cores=2 \
    --conf spark.executor.memory=4g \
    --conf spark.executor.memoryOverhead=512m \
    --conf spark.default.parallelism=128 \
    --conf spark.dynamicAllocation.enabled=false \
    --conf spark.scheduler.minRegisteredResourcesRatio=0.5 \
    --conf spark.locality.wait=0s \
    --conf spark.pyspark.python=/usr/bin/python3 \
    --conf spark.pyspark.driver.python=/usr/bin/python3 \
    --conf spark.fileserver.port=18080 \
    --conf spark.driver.port=7078 \
    --conf spark.blockManager.port=7079 \
    --conf spark.port.maxRetries=0 \
    --conf spark.driver.bindAddress=0.0.0.0 \
    --conf spark.driver.host=spark-master-0.spark-master-hs.spark.svc.cl
    uster.local \
    --conf spark.driver.port=7078 \
    --conf spark.blockManager.port=7079 \
    --jars local:///opt/spark/jars/hadoop-aws-3.3.4.jar,local:///opt/spa
    rk/jars/aws-java-sdk-bundle-1.12.262.jar \
    --conf spark.hadoop.fs.s3a.endpoint=http://minio.minio.svc.cluster.l
    ocal:9000 \
    --conf spark.hadoop.fs.s3a.access.key=minioadmin \
    --conf spark.hadoop.fs.s3a.secret.key=minioadmin123 \
```

```
    --conf spark.hadoop.fs.s3a.path.style.access=true \
    --conf spark.hadoop.fs.s3a.connection.ssl.enabled=false \
    --conf
    spark.hadoop.fs.s3a.impl=org.apache.hadoop.fs.s3a.S3AFileSystem \
    --conf spark.hadoop.fs.s3a.aws.credentials.provider=org.apache.hadoo
    p.fs.s3a.SimpleAWSCredentialsProvider \
    s3a://<NOME_BUCKET>/<NOME_FILE>.py \
    --samples-per-partition 10000000 \
    --partitions 128 \
    --output s3a://<NOME_BUCKET>/benchmarks/bench_pi_scaling
'
```

dove:

- <NOME_BUCKET> rappresenta il nome del Bucket di MinIO
- <NOME_FILE> rappresenta il nome del file Python da eseguire

Per visualizzare l'esecuzione del programma con i relativi worker abilitati connettersi alla Spark UI al seguente link:

```
http://localhost:30085
```

2.3.2 Riavvio sistema

```
#Posizionarsi nella cartella home utente
cd

# Privilegi Utente
sudo usermod -a -G microk8s <NOME_UTENTE>
sudo chown -R <NOME_UTENTE> ~/.kube
newgrp microk8s

# Avvia e controlla microk8s
microk8s status || sudo microk8s start
microk8s status --wait-ready
```

```
# Verifica stati dei Pods:

kubectl get nodes
kubectl -n minio get pods
kubectl -n spark get pods

cd k8s-local/spark
# Esegui i programmi
```

dove <NOME_UTENTE> rappresenta lo username dell'account Linux.

2.3.3 Programma Python: Test di Benchmark

Per testare il funzionamento è stato eseguito il file python scaricabile al seguente link. Di seguito mostrati i risultati del benchmark eseguendo il programma con 1, 2 ed infine 3 nodi worker.
Come previsto, la parallelizzazione dell'esecuzione ha ridotto ampiamente i tempi di esecuzione.

Numero di Nodi Worker	Tempo di Esecuzione
1	72s
2	39s
3	28s