**Documentazione Docker ClassifAI Multichain ed Explorer Dashboard**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Autore/i: Arifur Rahman** |  |  | **Roma, lì** | | 05/10/2022 |
| **Rivisto da : Damiano Landi** |  |  | **Roma, lì** | 14/10/2022 | |
| **Accettato da:** |  |  | **Roma, lì** | |  |
| **Referente:** |  |  | **Roma, lì** | |  |

**SOMMARIO**

[**1.**](#_heading=h.2et92p0) **Introduzione 3**

[**2.**](#_heading=h.2et92p0) **Architettura della soluzione 3**

[**3.**](#_heading=h.2et92p0) **Documentazione Dockerfile 5**

**4**[**.**](#_heading=h.2et92p0) **Documentazione docker-compose.yml 8**

[**5.**](#_heading=h.2et92p0) **Procedura per lanciare l’applicazione 13**

[**6.**](#_heading=h.2et92p0) **Sistema Operativo e librerie utilizzate nel progetto 14**

**7**[**.**](#_heading=h.2et92p0) **Implementazione Terraform e Kubernetes 17**

1. **Introduzione**

**Perché utilizzare docker**

Questo documento illustra la procedura per creare un'immagine Docker della nostra soluzione multichain ed annesso explorer e come lanciarlo in ambiente locale, in modo da renderlo facilmente accessibile ed utilizzabile su varie piattaforme e contesti diversi in maniera più efficiente e veloce rispetto ad una classica virtual machine.

1. **ARCHITETTURA DELLA SOLUZIONE**
   1. Introduzione

La nostra soluzione comprende due fulcri principali:

* + 1. Multichain

La multichain è l’interfaccia che ci permette di relazionarci con la Blockchain e consultare i dati al suo interno, fornendoci utili comandi necessari ad interrogarla.

* + 1. Dashboard

La nostra Dashboard è l’interfaccia tramite cui l’utente potrà andare a visualizzare i dati di suo interesse. Al suo interno saranno visitabili svariate pagine legate ai vari aspetti dei dati della Blockchain, dalla quantità di dati ai dettagli degli stessi.

* 1. Struttura del File System

2.2.1: All’interno del progetto saranno presenti diversi folder:

-base

-dashboard

-master

-multichain-volume

-node

Oltre questi, saranno presenti anche un file docker-compose.yml e gli script di start e build del progetto. All’interno di tutte le cartelle menzionate sopra, eccetto multichain-volume, sarà presente un Dockerfile dedicato a gestire la specifica sezione del progetto

2.2.2: Il folder “base” e il Dockerfile in esso contenuto saranno responsabili della creazione dell’immagine basilare che farà da environment per la multichain e l’applicazione front-end di visualizzazione dei dati.

2.2.3: Il folder “dashboard” e relativo Dockerfile sono invece responsabili per la realizzazione dell’explorer dei dati. Oltre il Dockerfile saranno presenti anche i vari file che vanno a comporre l’applicazione stessa.

2.2.4: Il folder “master”, con relativo Dockerfile, sarà responsabile per la creazione e setup del master node della multichain dell’intero progetto. In aggiunta al Dockerfile sarà possibile trovare gli script runchain.sh, dedicato ad avviare la multichain stessa, e blocknotify.sh, dedicato a creare dei blocchi vuoti per inizializzare la chain.

2.2.5: All’interno della cartella “multichain-volume” non sarà inizialmente presente alcun file. Una volta avviata l’immagine Docker del progetto vi saranno conservate automaticamente le impostazioni della multichain, in modo da non aver bisogno di impostarle nuovamente ad un secondo avvio dell’immagine.

2.2.6: Il folder “node”, insieme al relativo Dockerfile, è responsabile per la creazione di un generico nodo della nostra chain, dipendente dal master node indicato al punto 2.2.4. Al suo interno, oltre appunto il Dockerfile, è presente uno script runchain.sh, dedicato ad aggiornare la multichain aggiungendo il nuovo nodo. **È possibile replicare questa cartella e, aggiornando di conseguenza il docker-compose.yml, creare nuovi nodi della catena a piacimento.**

2.2.7: Lo script “build.sh” contiene un solo comando,

docker build -t mediatica/base-multichain ./base

Questo comando va a creare l’immagine base del nostro progetto, mediatica/base-multichain, partendo dalle impostazioni del Dockerfile all’interno della directory ./base

2.2.8: Lo script “start.sh” contiene le istruzioni utili ad avviare l’immagine del nostro progetto. Il primo comando all’interno di esso è “sh build.sh”, che va a richiamare lo script di build menzionato al punto precedente. Le istruzioni successive sono

docker compose build

docker compose up

Queste istruzioni vanno a comporre e lanciare realmente l’applicazione, seguendo le informazioni dettate all’interno del file docker-compose.yml, che a sua volta si rifà ai vari Dockerfile spiegati precedentemente.

1. **Documentazione Dockerfile**

Useremo una serie di Dockerfile per creare l'immagine Docker della nostra dashboard di esposizione dei dati della multichain, e della multichain stessa. Di seguito vengono spiegati e descritti i Dockerfile. Il primo passo è quello di installare Docker Desktop sulla propria macchina.

3.0: Il primo Dockerfile che andiamo ad esplorare sarà quello contenuto all’interno della cartella “base”, che quindi costituisce le fondamenta del nostro progetto

3.0.1 : Come prima istruzione all’interno del Dockerfile si andrà ad indicare l’immagine di riferimento che andrà a costituire l’ambiente in cui verrà eseguita l’applicazione. In questo caso:

FROM ubuntu:xenial

La keyword FROM indica a Docker su quale immagine vuoi basare la tua (nell'esempio in questione, appunto, ubuntu:xenial).

3.0.2: L’istruzione successiva sarà:

ENV DEBIAN\_FRONTEND noninteractive

ENV consente di impostare variabili di ambiente, che possono essere utilizzate in seguito, nel Dockerfile stesso.

3.0.3: Successivamente andremo a descrivere le istruzioni che dovranno venire effettuate all’interno dell’ambiente dell’ immagine

RUN apt-get update \

&& apt-get upgrade -q -y \

&& apt-get dist-upgrade -q -y \

&& apt-get install -q -y wget curl \

&& apt-get clean \

&& rm -rf /var/lib/apt/lists/\* \

&& cd /tmp \

&& wget http://www.multichain.com/download/multichain-2.3.tar.gz \

&& tar -xvzf multichain-2.3.tar.gz \

&& cd multichain-2.3 \

&& mv multichaind multichain-cli multichain-util /usr/local/bin \

&& cd /tmp \

&& rm -Rf multichain\*

Tramite “apt-get update”, “apt-get upgrade -q -y” e “apt-get dist-upgrade -q -y” vengono aggiornati pacchetti e dipendenze dell’immagine. “apt-get install -q -y wget curl”, invece, va a scaricare ed installare le dipendenze wget e curl. Con “rm -rf /var/lib/apt/lists/\*” si vanno a ripulire le informazioni superflue riguardanti lo stato delle dipendenze. Tramite “ cd /tmp”, “wget <http://www.multichain.com/download/multichain-2.3.tar.gz>” e “tar -xvzf multichain-2.3.tar.gz” si vanno a scaricare ed estrarre i file ufficiali della multichain, che verranno utilizzati poi per settare l’ambiente ed inizializzare la blockchain. Con la sequenza “cd multichain-2.3”, “mv multichaind multichain-cli multichain-util /usr/local/bin” si vanno a selezionare i file utili al nostro progetto e vengono spostati all’interno del percorso /usr/local/bin. Infine con “cd /tmp” e “rm -Rf multichain\*” si rimuovono i file superflui scaricati precedentemente.

3.0.4: L’ istruzione successiva, che va a chiudere il Dockerfile base, sarà:

CMD[“/bin/bash”]

In questo modo si va a indicare la bash che verrà utilizzata all’inizio dell’esecuzione del container.

3.1: Il prossimo Dockerfile che verrà esplorato in questo documento è quello all’interno della cartella “dashboard”. Come per il Dockerfile precedente, come istruzione iniziale andremo ad indicare l’immagine di riferimento per l’ambiente. In questo caso:

FROM python:3.9-buster

3.1.1: L’istruzione successiva sarà:

WORKDIR /dashboard

L’istruzione WORKDIR viene utilizzata per impostare la directory di lavoro per tutte le successive istruzioni del Dockerfile.

3.1.2: Il prossimo passo è quello di andare a scrivere:

COPY . /dashboard

In questo modo vengono duplicati i file della directory indicata all’interno dell’immagine.

3.1.3: Si andrà poi ad installare le dependencies necessarie al funzionamento del progetto. Questo si farà tramite:

RUN pip install -r requirements.txt

Questo comando andrà a leggere dal file requirements.txt, interno alla cartella dashboard, la lista di dipendenze da installare nel progetto.

3.1.4: Il passo successivo sarà quello di indicare la porta di collegamento tra la dashboard e il resto del container Docker. Si indica tramite l’istruzione:

EXPOSE 2751

In questo caso viene impostata la porta 2751.

3.1.5: Infine bisognerà inserire nel file:

CMD ["python3", "dashboard.py"]

L'istruzione CMD specifica il programma predefinito che verrà eseguito una volta lanciato il contenitore.

3.2: Il prossimo Dockerfile che verrà trattato è quello contenuto nella cartella “master”. Come immagine di riferimento, questa volta andremo ad indicare l’immagine base creata con il Dockerfile al punto 3.0.

FROM mediatica/base-multichain

3.2.1: Successivamente saranno indicate due istruzioni COPY, le quali permetteranno di avere a disposizione determinati file all’interno dell’immagine.

COPY ./runchain.sh /root/runchain.sh

COPY ./blocknotify.sh /root/blocknotify.sh

3.2.2: Le prossime due istruzioni, legate alle due menzionate al punto precedente, saranno:

RUN chmod a+x /root/runchain.sh

RUN chmod a+x /root/blocknotify.sh

In modo da dare i permessi all’immagine di lanciare gli script copiati al punto precedente.

3.2.3: Infine il file si conclude con l’istruzione:

CMD[“/bin/bash”, “/root/runchain.sh”]

Questa istruzione va a lanciare lo script per lanciare la multichain.

3.3: A conclusione di questa sezione andremo a trattare il Dockerfile contenuto nella cartella “node”. Quest’ultimo è fondamentalmente identico al Dockerfile master, tuttavia non contiene le istruzioni relative allo script blocknotify.sh. Anche la cartella node contiene uno script runchain.sh, il quale aggiorna la multichain aggiungendo un nuovo nodo. Il file completo quindi risulta essere:

FROM mediatica/base-multichain

ADD ./runchain.sh /root/runchain.sh

RUN chmod a+x /root/runchain.sh

CMD ["/bin/bash", "/root/runchain.sh"]

1. **Documentazione docker-compose.yml**

4.0: A differenza del Dockerfile, il file docker-compose.yml servirà a descrivere le specifiche di servizi, rete, porte e altre caratteristiche che vanno a definire il contesto nel quale la nostra applicazione verrà eseguita, e le relazioni tra le varie parti dell’immagine.

4.1: Per prima cosa dovremo andare a creare un file di tipo .yml nel folder in cui si trova la nostra applicazione. Il file andrà rinominato “docker-compose”. Nei passaggi seguenti verrà spiegato nel dettaglio il contenuto dello stesso.

4.2: La prima istruzione che si dovrà inserire all’interno del docker-compose.yml sarà quella relativa alla versione di riferimento della nostra immagine. Nel nostro caso l’istruzione sarà:

version: "3.8"

4.3: Successivamente andranno specificati i “services” della nostra applicazione. Andremo ad utilizzare l’istruzione:

services:

per indicare al docker-compose che le istruzioni successive saranno da interpretare come specifiche dei services. N.B.: Le righe successive al di sotto dell’istruzione “services:” andranno indentate di un tab per permettere la corretta lettura del file .yml.

4.4: Al di sotto di “services:” andremo a definire tre macro-categorie tramite le istruzioni:

masternode:

slavenode\_1:

dashboard:

N.B.: Le istruzioni “masternode:”, “slavenode\_1” e “dashboard:” andranno allo stesso livello di indentazione, ma le istruzioni successive all’interno di queste tre macro-categorie andranno indentate ulteriormente. Inoltre, come accennato al punto 2.2.6, è possibile creare nuovi slavenode aggiungendo i relativi service.

4.5: Sotto la macro-categoria masternode si dovranno andare a definire i parametri per l’esecuzione della stessa. Per prima cosa possiamo indicare al nostro docker-compose.yml la directory di un Dockerfile esistente, in modo da ereditarne le configurazioni. Questo può essere fatto tramite l’istruzione:

build: ./master

4.6: Successivamente si andranno ad indicare le impostazioni “tty” e “stdin\_open”:

stdin\_open: true

tty: true

4.7: Il prossimo passo consiste nel definire le porte che l’immagine andrà ad utilizzare internamente per comunicare con le altre parti dell’applicazione.

expose:

- 7557

- 8002

**N.B.: Rispettare bene l’indentazione per permettere la corretta lettura del file.**

4.8: Si andranno poi a definire le variabili di ambiente dell’immagine. Molte di esse verranno lette da un file di configurazione .env da creare all’interno del folder del progetto.

environment:

CHAINNAME: $CHAINNAME\_DEV

NETWORK\_PORT: $NETWORK\_PORT

RPC\_PORT: $RPC\_PORT\_DEV

RPC\_USER: $RPC\_USER\_DEV

RPC\_PASSWORD: $RPC\_PASS\_DEV

RPC\_ALLOW\_IP: $RPC\_ALLOW\_IP

PARAM\_TARGET\_BLOCK\_SIZE: target-block-time|30

PARAM\_ANYONE\_CAN\_CONNECT: anyone-can-connect|true

PARAM\_ANYONE\_CAN\_MINE: anyone-can-mine|true

4.9: L’ultima istruzione all’interno della categoria masternode sarà quella di indicare la directory dei volumes, la stessa illustrata al punto 2.2.5:

volumes:

- ./multichain\_volume:/root/.multichain

4.10: La prima istruzione situata all’interno della sezione dedicata al service “slavenode\_1” sarà l’istruzione build, illustrata al punto 4.5:

build: ./node

4.11: Andremo successivamente a definire le impostazioni relative “tty” e “stdin\_open”, illustrate al punto 4.6:

stdin\_open: true

tty: true

4.12: In seguito andranno indicate le porte di esposizione dell’immagine, vedi 4.7:

expose:

- 7557

- 8002

4.13: Si andranno poi a definire le variabili di ambiente, in maniera analoga al punto 4.8:

environment:

CHAINNAME: $CHAINNAME\_DEV

NETWORK\_PORT: $NETWORK\_PORT

RPC\_PORT: $RPC\_PORT

RPC\_USER: $RPC\_USER

RPC\_PASSWORD: $RPC\_PASSWORD

RPC\_ALLOW\_IP: 0.0.0.0/0.0.0.0

MASTER\_NODE: masternode

4.14: Infine andranno definite le istruzioni per relazioni e gerarchia tra l’immagine del nodo corrente ed il master, rispettivamente tramite le istruzioni “links” e “depends\_on”:

links:

- masternode

depends\_on:

- masternode

4.15: Come per i nodi precedenti, la prima istruzione relativa al service “dashboard” sarà “build” (vedi 4.5):

build: ./dashboard

4.16: Successivamente si andrà a definire la porta dell’applicazione tramite l’istruzione “ports”:

ports:

- 2751:2751

4.17: Infine andranno definite le variabili d’ambiente dell’immagine (vedi 4.8):

environment:

- RPC\_HOST=$RPC\_HOST

- RPC\_PORT=$RPC\_PORT

- RPC\_USER=$RPC\_USER

- RPC\_PASS=$RPC\_PASS

- RPC\_HOST\_DEV=$RPC\_HOST\_DEV

- RPC\_PORT\_DEV=$RPC\_PORT\_DEV

- RPC\_USER\_DEV=$RPC\_USER\_DEV

- RPC\_PASS\_DEV=$RPC\_PASS\_DEV

- RPC\_HOST\_PRODUCTION=$RPC\_HOST\_PRODUCTION

- RPC\_PORT\_PRODUCTION=$RPC\_PORT\_PRODUCTION

- RPC\_USER\_PRODUCTION=$RPC\_USER\_PRODUCTION

- RPC\_PASS\_PRODUCTION=$RPC\_PASS\_PRODUCTION

1. **Procedura per lanciare l’applicazione**

5.0 : Scarica l'ultima versione di Docker Desktop se stai utilizzando un sistema Windows, oppure Docker Desktop Edge se ti trovi su un sistema Linux o MacOS.

Link : https://docs.docker.com/desktop/install/mac-install/

5.1 : Lanciare l’applicazione Docker Desktop o Docker Desktop Edge

5.2: Su una macchina Linux basterà posizionarsi tramite terminale all’interno del folder del progetto e lanciare lo script di start tramite il comando “sh start.sh”. Questo lancerà lo script illustrato al punto 2.2.8, che svolgerà automaticamente le procedure necessarie ad avviare il container.

5.3 : In caso ci si trovasse su un sistema Windows, sarà necessario lanciare individualmente i comandi dal terminale. Posizionandosi all’interno della directory del progetto si dovrà inviare il comando:

docker build -t mediatica/base-multichain ./base

Questo andrà, come spiegato al punto 2.2.7, a creare l’immagine base del nostro progetto.

5.4: Successivamente si andranno a lanciare i comandi necessari all’avvio del container tramite Docker Compose.

docker compose build

docker compose up

Questi due comandi andranno a leggere le istruzioni all’interno del file docker-compose.yml, spiegate in modo dettagliato nella sezione 3 di questo documento, e avvieranno il container dell’applicazione.

5.5: Una volta lanciato il comando al punto precedente, sarà possibile accedere all’applicazione tramite localhost, alla porta indicata precedentemente nel file docker-compose.yml nel service “dashboard”.

1. **Sistema Operativo e librerie utilizzate nel progetto**

6.0: Sistema Operativo: Ubuntu 22.04

6.1: Librerie Pip utilizzate nel progetto:

appdirs 1.4.4

async-generator 1.10

attrs 22.1.0

Babel 2.10.3

beautifulsoup4 4.11.1

bokeh 2.4.3

Brotli 1.0.9

bs4 0.0.1

cachelib 0.9.0

certifi 2022.6.15.2

cffi 1.15.1

charset-normalizer 2.1.1

click 8.1.3

colorama 0.4.5

configobj 5.0.6

cssselect2 0.7.0

dash 2.6.1

dash-core-components 2.0.0

dash-html-components 2.0.0

dash-table 5.0.0

distlib 0.3.6

exceptiongroup 1.0.0rc9

ez-setup 0.9

filelock 3.8.0

Flask 2.2.2

Flask-Compress 1.12

Flask-Session 0.4.0

h11 0.14.0

idna 3.4

itsdangerous 2.1.2

Jinja2 3.1.2

lxml 4.9.1

MarkupSafe 2.1.1

mcrpc 2.0.6.0

numpy 1.23.3

outcome 1.2.0

packaging 21.3

pandas 1.4.4

pdfkit 1.0.0

Pillow 9.2.0

pip 22.2.2

platformdirs 2.5.2

plotly 5.10.0

pycparser 2.21

pyparsing 3.0.9

PyQt5 5.15.7

PyQt5-Qt5 5.15.2

PyQt5-sip 12.11.0

PySocks 1.7.1

python-dateutil 2.8.2

python-dotenv 0.21.0

pytz 2022.2.1

PyYAML 6.0

reportlab 3.6.11

requests 2.28.1

selenium 4.5.0

setuptools 63.2.0

simplejson 3.17.6

six 1.16.0

sniffio 1.3.0

sortedcontainers 2.4.0

soupsieve 2.3.2.post1

svglib 1.4.1

tabulate 0.8.10

tenacity 8.0.1

tinycss2 1.1.1

tkcalendar 1.6.1

tornado 6.2

trio 0.22.0

trio-websocket 0.9.2

typing\_extensions 4.3.0

urllib3 1.26.12

virtualenv 20.16.5

webencodings 0.5.1

Werkzeug 2.2.2

wheel 0.37.1

wsproto 1.2.0

1. TERRAFORM FILE IMPLEMENTATION AND DEPLOY IN MINIKUBE (KUBERNETES CLUSTER):

Ho sviluppato un file di infrastruttura di terraform per distribuire il nostro Multichain-Explorer-2 nel Kubernets Cluster (Minikube).

Il file dell'infrastruttura Terraform descrive di seguito:

# Provide terraform provider from hashicorp/kubernetes

terraform {

required\_providers {

kubernetes = {

source = "hashicorp/kubernetes"

version = ">= 2.0.0"

}

}

}

#provider is minikube kubernets .kube file path

provider "kubernetes" {

config\_path = "~/.kube/config"

}

# define terraform kubernets namespace

resource "kubernetes\_namespace" "multichainexplorer-terraform-latest" {

metadata {

name = "multichainexplorer-terraform-latest"

}

}

# define terraform kubernets deployments infrastructure format

resource "kubernetes\_deployment" "multichainexplorer-terraform-latest" {

metadata {

name = "multichainexplorer-terraform-latest"

namespace = kubernetes\_namespace.multichainexplorer-terraform-latest.metadata.0.name

}

spec {

replicas = 3 # you can increase replica

selector {

match\_labels = {

app = "multichainexplorer-terraform-latest"

}

}

template {

metadata {

labels = {

app = "multichainexplorer-terraform-latest"

}

}

spec {

container {

image = "arifur43210/multichain-explorer-2:latest" # Dockerhub image to pull or Dockerfile path

name = "multichainexplorer2"

port {

container\_port = 5000

}

}

}

}

}

}

# define terraform kubernets service infrastructure format

resource "kubernetes\_service" "multichainexplorer-terraform-latest" {

metadata {

name = "multichainexplorer-terraform-latest"

namespace = kubernetes\_namespace.multichainexplorer-terraform-latest.metadata.0.name

}

spec {

selector = {

app = kubernetes\_deployment.multichainexplorer-terraform-latest.spec.0.template.0.metadata.0.labels.app

}

session\_affinity = "ClientIP"

#type = "LoadBalancer"

# or

type = "NodePort"

port {

node\_port = 30210

port = 84

target\_port = 84

}

}

}

# define terraform kubernets pods infrastructure format

resource "kubernetes\_pod" "multichainexplorer-terraform-latest" {

metadata {

name = "multichainexplorer-terraform-latest"

}

spec {

container {

image = "arifur43210/multichain-explorer-2:latest"

name = "multichainexplorer-terraform-latest"

resources {

limits = {

cpu = "0.5"

memory = "512Mi"

}

requests = {

cpu = "250m"

memory = "50Mi"

}

}

}

}

}

1. KUBERNETES MANIFEST FILE IMPLEMENTATION AND DEPLOY IN MINIKUBE (KUBERNETES CLUSTER):

Ho sviluppato un file di infrastruttura di Kubernetes manifest per distribuire il nostro Multichain-Explorer-2 nel Kubernets Cluster (Minikube).

Il file dell'infrastruttura Kubernetes manifest descrive di seguito:

# To create the Kubernetes deployments infrastructure format

apiVersion: apps/v1

kind: Deployment

metadata:

name: multichainexplorer

labels:

app: multichainexplorer

spec:

replicas: 3 # you can increase replica

selector:

matchLabels:

app: multichainexplorer

template:

metadata:

labels:

app: multichainexplorer

spec:

containers:

- name: multichainexplorer

image: arifur43210/multichain-explorer-2:latest # or your dockerhub image or dpckerfile path

ports:

- containerPort: 80

# for scaling the clusters

resources:

requests:

memory: "64Mi"

cpu: "125m"

limits:

memory: "128Mi"

cpu: "250m"

# To create the Kubernetes Service infrastructure format

---

kind: Service

apiVersion: v1

metadata:

name: multichainexplorer

spec:

selector:

app: multichainexplorer

ports:

- protocol: TCP

port: 90

targetPort: 90

nodePort: 31015

type: NodePort

#type: LoadBalancer

# To create the Kubernetes Pod infrastructure format

---

apiVersion: v1

kind: Pod

metadata:

name: multichainexplorer

spec:

containers:

- name: multichainexplorer

image: arifur43210/multichain-explorer-2:latest # or your dockerhub image or dockerfile path

ports:

- containerPort: 5000

---

# To create the Kubernetes Job infrastructure format

apiVersion: batch/v1

kind: Job

metadata:

name: multichainexplorer

spec:

template:

spec:

containers:

- name: multichainexplorer

image: arifur43210/multichain-explorer-2:latest

command: ["python", "-m", "flask", "run", "--host=0.0.0.0"]

restartPolicy: Never

backoffLimit: 4

1. COMMAND PER IMPLEMENTAZIONE E DISTRIBUZIONE:

Per prima cosa avvia il Minikube Segui il comando seguente:

1. Minikube start
2. Minikube status
3. Minikube dashboard

Successivamente, segui il comando seguente per distribuire il file manifest kubernetes :

1.  kubectl delete -f multichainexplorer.yaml

2. kubectl apply -f multichainexplorer.yaml

3. minikube service list

4. kubectl port-forward pod/multichainexplorer-terraform-n 8080:5000

1. open browser " http://localhost:8080/ "

Quindi segui il comando seguente per distribuire il file Terraform:

1.  terraform init

2. terraform plan

3. terraform apply -auto-approve

4. kubectl port-forward pod/multichainexplorer-terraform-n 8080:5000

5. open browser " http://localhost:8080/ "