

# Università della Calabria

---

Dipartimento di Ingegneria Informatica, Modellistica,  
Elettronica e Sistemistica



Corso di Laurea Magistrale in Ingegneria Informatica

*Progetto di Sistemi distribuiti e Cloud computing*

**Coordinatore distribuito con elezione**

**Docente**

Domenico Talia

Loris Belcastro

Simone Gatto 235174

# INDICE

<b>1. STRUTTURA DEL PROGETTO.....</b>	<b>3</b>
1.1. APACHE ZOOKEEPER .....	3
1.2. DOCKER.....	7
<b>2. FASE DI INSTALLAZIONE E TEST .....</b>	<b>9</b>
2.1. PRE-REQUISITI.....	9
2.2. INSTALLAZIONE E CONFIGURAZIONE APACHE ZOOKEEPER .....	10
2.3. JAVA & ZOOKEEPER API .....	14
2.4. PRIMA FASE DI TEST .....	21
2.5. INSTALLAZIONE DOCKER.....	23
<b>3. ESECUZIONE FINALE .....</b>	<b>29</b>
<b>4. CONCLUSIONI .....</b>	<b>32</b>

# 1. Struttura del Progetto

## 1.1. Apache Zookeeper

ZooKeeper è un server open-source per la coordinazione distribuita affidabile di applicazioni cloud. È un progetto dell'Apache Software Foundation.



Esso è essenzialmente un servizio per sistemi distribuiti che offre uno **store**

**gerarchico di tipo chiave-valore**, utilizzato per fornire un servizio di configurazione distribuita, un servizio di sincronizzazione e un registro di denominazione per sistemi distribuiti di grandi dimensioni. Inizialmente era un sotto-progetto di Hadoop, ma ora è diventato un progetto Apache di primo livello autonomo.

### Panoramica

L'architettura supporta l'alta disponibilità attraverso servizi ridondanti. I client possono quindi richiedere un altro leader di ZooKeeper se il primo non risponde. I nodi memorizzano i loro dati in uno spazio dei nomi gerarchico, simile a un file system o a una struttura dati ad albero. I client possono leggere e scrivere sui nodi e in questo modo avere un servizio di configurazione condiviso. Può essere visto come un sistema di trasmissione atomica, attraverso il quale gli aggiornamenti sono totalmente ordinati. Il **protocollo di trasmissione atomica di ZooKeeper (ZAB)** è il cuore del sistema.

Alcune delle **principali caratteristiche** sono:

**Sistema affidabile:** Questo sistema è molto affidabile poiché continua a funzionare anche se un nodo fallisce;

**Architettura semplice:** L'architettura è piuttosto semplice, con uno spazio dei nomi gerarchico condiviso che aiuta a coordinare i processi;

**Elaborazione veloce:** È particolarmente veloce in carichi di lavoro "dominanti in lettura";

**Scalabile:** Le prestazioni possono essere migliorate aggiungendo nodi.

### Architettura

I servizi nel cluster sono replicati e memorizzati su un insieme di server (chiamato "*ensemble*"), ognuno dei quali mantiene un database in memoria contenente l'intero albero dei dati di stato, nonché un log delle transazioni e snapshot memorizzati in modo persistente. Più applicazioni client possono connettersi a un server, e ogni client mantiene una connessione TCP attraverso la quale invia richieste e invia messaggi di stato e riceve risposte ed eventi di monitoraggio.

## Un servizio di coordinazione distribuita per applicazioni distribuite

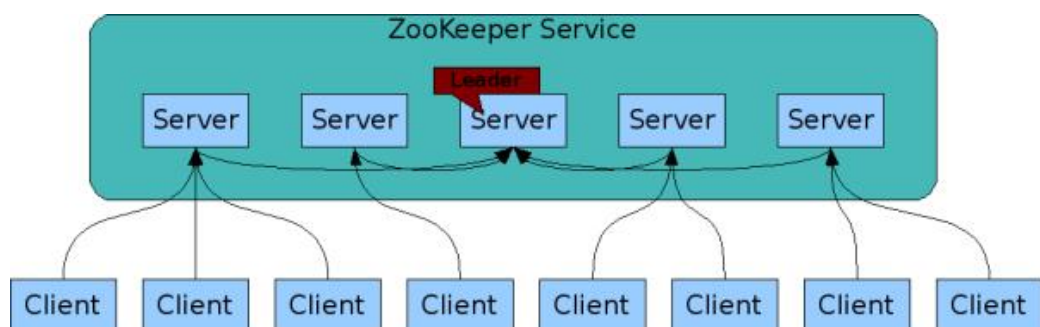
I servizi di coordinazione sono notoriamente difficili da implementare correttamente. Sono particolarmente inclini a errori come le condizioni di gara e i deadlock. La motivazione alla base di ZooKeeper è quella di alleviare alle applicazioni distribuite la responsabilità di implementare servizi di coordinazione da zero.

### Obiettivi di progettazione

Consente a processi distribuiti di coordinarsi tra loro attraverso un **namespace gerarchico condiviso**, organizzato in modo simile a un file system standard. Il namespace è costituito da **registri di dati**, chiamati **znode** e sono simili a file e directory. A differenza di un tipico file system, progettato per lo storage, i dati vengono mantenuti in memoria, il che significa che si può raggiungere elevata velocità di trasmissione e bassi valori di latenza.

L'implementazione mette un'enfasi sull'**alta performance**, l'**alta disponibilità** e l'**accesso rigorosamente ordinato**. Gli aspetti di performance significano che può essere utilizzato in sistemi distribuiti di grandi dimensioni. Gli aspetti di affidabilità impediscono che diventi un singolo punto di fallimento. L'ordinamento rigoroso significa che primitive di sincronizzazione sofisticate possono essere implementate a livello di client.

**ZooKeeper è replicato.** Come i processi distribuiti che coordina, esso stesso è destinato a essere replicato su un insieme di host chiamato **ensemble**.



I client si connettono a un singolo server ZooKeeper. Il client mantiene una connessione TCP attraverso la quale invia richieste, ottiene risposte, riceve eventi di monitoraggio e invia heartbeat. Se la connessione TCP al server si interrompe, il client si collegherà a un server diverso.

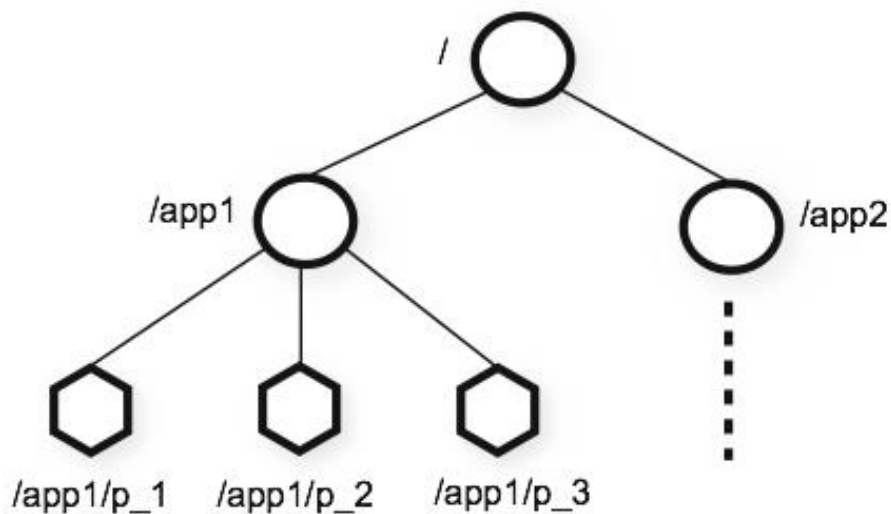
**ZooKeeper è ordinato.** ZooKeeper assegna un numero a ogni aggiornamento che riflette l'ordine di tutte le transazioni di ZooKeeper. Le operazioni successive possono utilizzare l'ordine per implementare astrazioni di livello superiore, come primitive di sincronizzazione.

**ZooKeeper è veloce.** È particolarmente veloce in carichi di lavoro "dominanti in lettura". Le applicazioni di ZooKeeper vengono eseguite su migliaia di macchine, e si comporta al meglio quando le letture sono più comuni delle scritture, con un rapporto di circa 10:1.

### Modello di dati e spazio dei nomi gerarchico

Lo spazio dei nomi fornito da ZooKeeper è molto simile a quello di un file system standard. Un nome è una sequenza di elementi di percorso separati da una barra (/). Ogni nodo nello spazio dei nomi di ZooKeeper è identificato da un percorso.

### Spazio dei nomi gerarchico di ZooKeeper



### Nodi e nodi effimeri

A differenza dei file system standard, ogni nodo in uno spazio dei nomi ZooKeeper può avere dati associati oltre ai figli. È come avere un file system che consente a un file di essere anche una directory. (ZooKeeper è stato progettato per archiviare dati di coordinamento: informazioni di stato, configurazione, informazioni sulla posizione, ecc., quindi i dati archiviati in ogni nodo sono di solito di piccole dimensioni, nell'intervallo di byte o kilobyte). Utilizziamo il termine "znode" per chiarire che stiamo parlando dei nodi dati di ZooKeeper.

Gli znode mantengono una struttura di stato che include numeri di versione per le modifiche dei dati, le modifiche delle ACL e i timestamp, per consentire la convalida della cache e gli aggiornamenti coordinati. Ogni volta che i dati di uno znode cambiano, il numero di versione aumenta. Ad esempio, ogni volta che un client recupera dati, riceve anche la versione dei dati.

I dati archiviati in ogni znode in uno spazio dei nomi vengono letti e scritti in modo atomico. Le letture ottengono tutti i byte di dati associati a uno znode e

una scrittura sostituisce tutti i dati. Ogni nodo ha una lista di **controllo degli accessi** (ACL) che limita chi può fare cosa.

ZooKeeper ha anche il concetto di nodi effimeri. Questi znode esistono finché la sessione che ha creato lo znode è attiva. Quando la sessione termina, lo znode viene eliminato.

### **Aggiornamenti condizionali e watch**

ZooKeeper supporta il concetto di watch. I client possono impostare un watch su uno znode. Un watch verrà attivato e rimosso quando lo znode cambia. Quando un watch viene attivato, il client riceve un pacchetto che indica che lo znode è cambiato. Se la connessione tra il client e uno dei server ZooKeeper viene interrotta, il client riceverà una notifica locale.

### **Qualche API**

Uno degli obiettivi di progettazione è quello di fornire un'interfaccia di programmazione molto semplice.

**create:** crea un nodo all'interno dell'albero;

**sequence Node (-s):** il nodo viene creato come sequenziale ovvero ZooKeeper al nome aggiunge un numero come suffisso, così da gestire il caso di inserimento di tanti nodi da parte di tanti client.

**ephemeral Node (-e):** ogni nodo creato con questo flag attivo verrà eliminato da ZooKeeper non appena il client che lo ha creato termina la sessione (per qualunque motivo, anche in caso di problemi di connettività del client).

**delete:** cancella un nodo;

**exists:** controlla se un nodo esiste all'interno dell'albero;

**get data:** legge i dati da un nodo;

**set data:** scrive i dati su un nodo;

**get children:** ritorna una lista di figli di un nodo.

## 1.2. Docker

Docker è un popolare software libero progettato per eseguire processi informatici in ambienti isolabili, minimali e facilmente distribuibili chiamati **container Linux** (o anche soltanto container), con l'obiettivo di semplificare i processi di deployment di applicazioni software.



Fin dal suo rilascio nel 2013, Docker non si basa sui tradizionali metodi di virtualizzazione di un intero sistema operativo, bensì sulla **OS-level virtualization** fornita dal **kernel Linux**, ovvero un singolo sistema operativo basato su Linux si occupa di orchestrare l'isolamento e le limitazioni delle risorse.

Docker implementa **API** di alto livello per gestire container che eseguono processi in ambienti isolati. Poiché utilizza delle funzionalità del kernel Linux (principalmente *cgroup* e *namespace*), un container di Docker, a differenza di una macchina virtuale, non include un sistema operativo separato. Al contrario, utilizza le funzionalità del kernel e sfrutta l'isolamento delle risorse (*CPU*, *memoria*, *I/O a blocchi*, *rete*) e i **namespace** separati per isolare ciò che l'applicazione può vedere del sistema operativo. Docker accede alle funzionalità di virtualizzazione del kernel Linux o direttamente utilizzando la libreria **libcontainer**, che è disponibile da Docker 0.9, o indirettamente attraverso **libvirt**, **LXC** o **systemd-nspawn**.

Utilizzando i container le risorse possono essere isolate, i servizi limitati e i processi avviati in modo da avere una prospettiva completamente privata del sistema operativo, col loro proprio identificativo, file system e interfaccia di rete. Più container condividono lo stesso kernel, ma ciascuno di essi può essere costretto a utilizzare una certa quantità di risorse, come la CPU, la memoria e l'I/O.

L'utilizzo di Docker per creare e gestire i container può semplificare la creazione di sistemi distribuiti, permettendo a diverse applicazioni o processi di lavorare in modo autonomo sulla stessa macchina fisica o su diverse macchine virtuali. Ciò consente di effettuare il deployment di nuovi nodi solo quando necessario, permettendo uno stile di sviluppo del tipo **platform as a service (PaaS)**. Docker inoltre semplifica la creazione e la gestione di code di lavori in sistemi distribuiti.

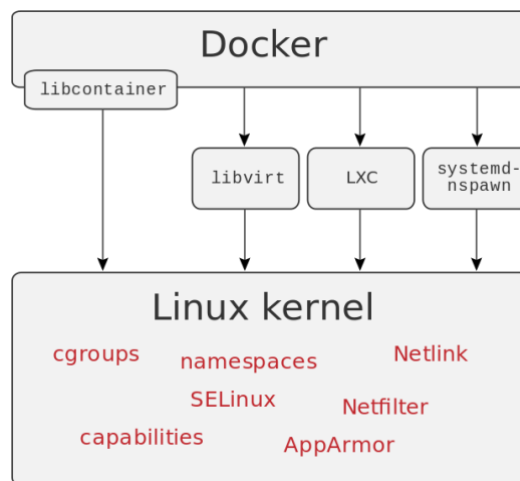
## Orchestrazione

Per far interagire tra loro più container Docker, spesso si utilizzano dei software di orchestrazione, come ad esempio Docker Swarm o Kubernetes.

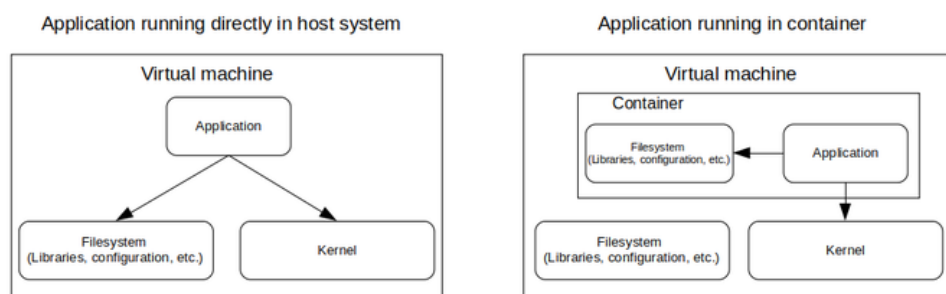
## Vantaggi e svantaggi

I vantaggi di Docker si misurano in relazione alle macchine virtuali. Infatti, **i container sono più leggeri delle macchine virtuali**, vengono avviati più velocemente e richiedono meno risorse. Anche le immagini risultano di dimensioni inferiori a quelle delle macchine virtuali. Tra gli svantaggi si annovera il fatto che i container non contengono un proprio sistema operativo e quindi i processi in esecuzione non possono essere isolati in maniera perfetta.

**Docker può utilizzare diverse interfacce per accedere alle funzionalità di virtualizzazione del kernel Linux**



## Differenza fra virtualizzazione e OS-level virtualization





## 2. Fase di Installazione e Test

### 2.1. Pre-requisiti

Questo progetto di Sistemi Distribuiti e Cloud Computing si è svolto utilizzando un sistema operativo sviluppato da Apple ovvero **MacOS Ventura 13.4** ed un **MacBook Air M1** con architettura di tipo **RISC** con utilizzo di istruzioni **ARM64**. In modo da avere un'installazione tranquilla, si è partiti installando tutti i necessari prerequisiti all'interno della macchina. I prerequisiti necessari sono stati: *homebrew, wget, xcode e rosetta2*.

#### Homebrew e xcode

In poche parole, **Homebrew** installa quello che ti serve e che Apple non ha installato.

È sostanzialmente un gestore di pacchetti che consente l'installazione di componenti aggiuntive basate principalmente per il terminale dei comandi in modo semplice e veloce. Tramite la sua compatibilità con Ruby ed altri linguaggi di programmazione è possibile anche usare il sistema per ottenere applicazioni e software desktop senza passare necessariamente per il Mac App Store.

Prima di tutto, come detto in precedenza, abbiamo bisogno anche di *xcode*.

**Xcode** è l'IDE (Integrated Development Environment) di Apple per macOS. Possiamo utilizzarlo per creare app per i vari dispositivi. Abbiamo bisogno di Xcode per Homebrew per l'installazione e la compilazione delle app. In altre parole, sul terminale digitiamo il comando al prompt di bash/zsh per installare Homebrew su Mac:

```
$ xcode-select --install
```

Confermiamo l'installazione, attendiamo il download e sarà pronto ed installato. Ora è il momento di procedere con Homebrew. Si può utilizzare il comando *wget* (non ancora installato) o scaricare un file con *curl* (alternativa a *wget* nel caso non sia ancora installato) nel seguente modo:

```
$ curl -O https://raw.githubusercontent.com/Homebrew/install/master/install.sh
```

Ora dobbiamo solamente installare con il comando ed abbiamo completato la procedura:

```
$ bash install.sh
```

## GNU/wget

GNU wget è un gestore di download libero, multiplatforma, parte del progetto **GNU**. Supporta i protocolli **HTTP**, **HTTPS** e **FTP**. Procediamo all'installazione col comando:

```
$ brew install wget
```

Installazione di wget completata, sarà necessario per poter effettuare il download dei prossimi strumenti.

## Rosetta 2

Rosetta è un **emulatore software** sviluppato dalla Apple che permette alle macchine dotate di processori x86 Intel di utilizzare il software compilato per PowerPC e alle macchine dotate di processori Apple Silicon di utilizzare il software compilato per processori Intel. Il nome è un chiaro riferimento alla Stele di Rosetta, la stele che consentì agli archeologi la decifrazione dei geroglifici egizi.

In quanto necessiteremo il download e l'utilizzo di Docker, andremo ad installare Rosetta 2, anche se a partire dalla versione 4.3 di *Docker Desktop*, non ci sono più grossi requisiti da rispettare per installare Docker su macOS con processori Apple Silicon M1 e M2.

Infatti, proprio dalla versione 4.3 di Docker Desktop, infatti, è stato rimosso il requisito fondamentale per l'installazione di **Rosetta 2**. Esistono tuttavia alcuni strumenti a riga di comando facoltativi che lo richiedono. Per installarlo manualmente dalla riga di comando, si esegue il seguente comando:

```
$ softwareupdate --install-rosetta
```

```
Install of Rosetta 2 finished successfully
simonegatto@MacBook-Air-di-Simone-4 ~ %
```

È ora di passare all'installazione dei principali software utilizzati per il progetto: **Apache Zookeeper** e **Docker Desktop**.

## 2.2. Installazione e Configurazione Apache Zookeeper

Prima di effettuare il testing di Zookeeper all'interno di Docker, è stato effettuato un test completo a livello locale e per questo si sono susseguite alcune diverse fasi, installazione, test, effettivo funzionamento ed in seguito la dockerizzazione del pacchetto Zookeeper.

**Download di Zookeeper 3.4.14:** (è stata utilizzata questa versione in modo che fosse compatibile con tutti gli strumenti utilizzati)

```
MacBook-Air-di-Simone-4:~ simonegatto$ cd desktop
MacBook-Air-di-Simone-4:desktop simonegatto$ wget https://archive.apache.org/dist/zookeeper/zookeeper-3.4.14/zookeeper-3.4.14.tar.gz
--2023-06-18 11:00:21-- https://archive.apache.org/dist/zookeeper/zookeeper-3.4.14/zookeeper-3.4.14.tar.gz
Risoluzione di archive.apache.org (archive.apache.org)... 65.108.204.189
Connessione a archive.apache.org (archive.apache.org)|65.108.204.189|:443... connesso.
Richiesta HTTP inviata, in attesa di risposta... 200 OK
Lunghezza: 37676320 (36M) [application/x-gzip]
Salvataggio in: «zookeeper-3.4.14.tar.gz»

zookeeper-3.4.14.tar.gz 9%[==>] 3,45M 625KB/s prev 56s
zzzzzookeeper-3.4.14.tar.gz 11%[==>] 4,13M 644KB/s pr
ev 53s zookeeper-3.4.14.tar.gz 11%[==>] 4,20M 621KB/s
prev 53s zookeeper-3.4.14.tar.gz 11%[==>] 4,27M 624KB/s
prev 53szookeeper-3.4.14.tar.gz 11%[==>] 4,31M 631KB/s pr
ev 53zookeeper-3.4.14.tar.gz 12%[==>] 4,36M 633KB/s prev 5
zookeeper-3.4.14.tar.gz 100%[=====] 35,93M 885KB/s in 47s

2023-06-18 11:01:09 (779 KB/s) - «zookeeper-3.4.14.tar.gz» salvato [37676320/37676320]
```

**Estrazione dall'archivio:**

```
MacBook-Air-di-Simone-4:~ simonegatto$ cd desktop
MacBook-Air-di-Simone-4:desktop simonegatto$ tar -zxvf zookeeper-3.4.14.tar.gz
x zookeeper-3.4.14/
x zookeeper-3.4.14/bin/
x zookeeper-3.4.14/bin/README.txt
x zookeeper-3.4.14/bin/zkCleanup.sh
x zookeeper-3.4.14/bin/zkCli.cmd
x zookeeper-3.4.14/bin/zkCli.sh
x zookeeper-3.4.14/bin/zkEnv.cmd
x zookeeper-3.4.14/bin/zkEnv.sh
x zookeeper-3.4.14/bin/zkServer.cmd
x zookeeper-3.4.14/bin/zkServer.sh
x zookeeper-3.4.14/bin/zkTxnLogToolkit.cmd
x zookeeper-3.4.14/bin/zkTxnLogToolkit.sh
x zookeeper-3.4.14/conf/
```

**Comandi utilizzati:**

```
$ cd desktop
```

```
$ wget https://archive.apache.org/dist/zookeeper/zookeeper-3.4.14/zookeeper-3.4.14.tar.gz
```

```
$ tar -zxvf apache-zookeeper-3.4.14.tar.gz
```

**Impostazioni necessarie:**

Configurare un server ZooKeeper in modalità **standalone** è semplice. Il server è contenuto in un singolo file JAR, quindi l'installazione consiste nel creare una configurazione.

Per avviare ZooKeeper è necessario un file di configurazione. Essa va creata nella seguente directory **conf/zoo.cfg**: (attenzione che inizialmente il file viene chiamato *zoo\_sample.cfg*, non è funzionante crearne uno nuovo in quanto crea problemi con i permessi ma è consigliabile di modificare e rinominare quello iniziale)

```
zoo.cfg ×
Users > simonegatto > Desktop > Zookeeper > zookeeper-3.4.14 > conf > zoo.cfg
1  tickTime=2000
2  dataDir=/var/log/zookeeperData
3  clientPort=2181
4  initLimit=5
5  syncLimit=2
```

**N.B.** Il valore di **dataDir** va modificato in modo da specificare una directory esistente (vuota all'inizio) in quanto inizialmente viene inserito come **dataDir=/tmp/zookeeper** e ciò non è funzionante.

#### Ecco ora il significato di ciascun campo:

**tickTime**: l'unità di tempo di base in millisecondi utilizzata da ZooKeeper. Viene utilizzato per inviare segnali e il timeout minimo della sessione sarà il doppio del tickTime;

**dataDir**: la posizione in cui archiviare gli snapshot del database in memoria e, a meno che diversamente specificato, il registro delle transazioni degli aggiornamenti al database;

**clientPort**: la porta su cui ascoltare le connessioni dei client;

**initLimit**: sono timeout che ZooKeeper utilizza per limitare il tempo massimo che i server ZooKeeper in quorum hanno per connettersi a un leader;

**syncLimit**: limita quanto un server può essere in ritardo rispetto a un leader.

Ora che è stato creato il file di configurazione, si può avviare il server ZooKeeper:

```
$ bin/zkServer.sh start
```

```
MacBook-Air-di-Simone-4:zookeeper-3.4.14 simonegatto$ sudo bin/zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /Users/simonegatto/Desktop/Zookeeper/zookeeper-3.4.14/bin/../conf/
zoo.cfg
Starting zookeeper ... STARTED
```

I passaggi che sono stati descritti precedentemente eseguono ZooKeeper in modalità standalone. Non c'è alcuna replica, quindi se il processo di ZooKeeper fallisce, il servizio verrà interrotto. Questo è sufficiente per la maggior parte delle situazioni di sviluppo.

Adesso possiamo procedere a connetterci a Zookeeper in modo da poter iniziare ad utilizzare la **CLI** (*Command Line Interface*) e per avere la possibilità di creare lo znode necessario al corretto funzionamento.

```
MacBook-Air-di-Simone-4:zookeeper-3.4.14 simonegatto$ sudo bin/zkCli.sh -server localhost:2181
Password:
Connecting to localhost:2181
2023-06-18 11:56:39,780 [myid: ] - INFO [main:Environment@100] - Client environment:zookeeper.version=3.4.14-4c25d480e66aad837d
e8b2df8da255ac140bcf, built on 03/06/2019 16:18 GMT
2023-06-18 11:56:39,782 [myid: ] - INFO [main:Environment@100] - Client environment:host.name=172.20.10.2
2023-06-18 11:56:39,782 [myid: ] - INFO [main:Environment@100] - Client environment:java.version=1.8_0_372
2023-06-18 11:56:39,783 [myid: ] - INFO [main:Environment@100] - Client environment:java.vendor=Azul Systems, Inc.
2023-06-18 11:56:39,783 [myid: ] - INFO [main:Environment@100] - Client environment:java.home=/Library/Java/JavaVirtualMachines/
zulu-8.jdk/Contents/Home/jre
2023-06-18 11:56:39,783 [myid: ] - INFO [main:Environment@100] - Client environment:java.class.path=/Users/simonegatto/Desktop/Z
ookeeper/zookeeper-3.4.14/bin/./zookeeper-server/target/classes:/Users/simonegatto/Desktop/Zookeeper/zookeeper-3.4.14/bin/./bu
il/classes:/Users/simonegatto/Desktop/Zookeeper/zookeeper-3.4.14/bin/./zookeeper-server/target/lib/*.jar:/Users/simonegatto/D
esktop/Zookeeper/zookeeper-3.4.14/bin/./build/lib/*.jar:/Users/simonegatto/Desktop/Zookeeper/zookeeper-3.4.14/bin/./lib/slf4j-
lg4j12-1.7.25.jar:/Users/simonegatto/Desktop/Zookeeper/zookeeper-3.4.14/bin/./lib/slf4j-api-1.7.25.jar:/Users/simonegatto/Desk
op/Zookeeper/zookeeper-3.4.14/bin/./lib/netty-3.10.6.Final.jar:/Users/simonegatto/Desktop/Zookeeper/zookeeper-3.4.14/bin/./lib
/loq4j-1.2.17.jar:/Users/simonegatto/Desktop/Zookeeper/zookeeper-3.4.14/bin/./lib/jline-0.9.94.jar:/Users/simonegatto/Desktop/Z
ookeeper/zookeeper-3.4.14/bin/./lib/audience-annotations-0.5.0.jar:/Users/simonegatto/Desktop/Zookeeper/zookeeper-3.4.14/bin/.
/zookeeper-3.4.14.jar:/Users/simonegatto/Desktop/Zookeeper/zookeeper-3.4.14/bin/./zookeeper-server/src/main/resources/lib/*.jar
:/Users/simonegatto/Desktop/Zookeeper/zookeeper-3.4.14/bin/./conf:
2023-06-18 11:56:39,783 [myid: ] - INFO [main:Environment@100] - Client environment:java.library.path=/Users/simonegatto/Library
/Java/Extensions:/Library/Java/Extensions:/Network/Library/Java/Extensions:/System/Library/Java/Extensions:/usr/lib/java:
2023-06-18 11:56:39,783 [myid: ] - INFO [main:Environment@100] - Client environment:java.io.tmpdir=/var/folders/zz/zyxpvq6gcxf
vn_n00000000000000/T/
2023-06-18 11:56:39,783 [myid: ] - INFO [main:Environment@100] - Client environment:java.compiler=<NA>
2023-06-18 11:56:39,783 [myid: ] - INFO [main:Environment@100] - Client environment:os.name=Mac OS X
2023-06-18 11:56:39,783 [myid: ] - INFO [main:Environment@100] - Client environment:os.arch=arch64
2023-06-18 11:56:39,783 [myid: ] - INFO [main:Environment@100] - Client environment:os.version=13.4
2023-06-18 11:56:39,783 [myid: ] - INFO [main:Environment@100] - Client environment:user.name=root
2023-06-18 11:56:39,783 [myid: ] - INFO [main:Environment@100] - Client environment:user.home=/var/root
2023-06-18 11:56:39,783 [myid: ] - INFO [main:Environment@100] - Client environment:user.dir=/Users/simonegatto/Desktop/Zookeep
er/zookeeper-3.4.14
2023-06-18 11:56:39,784 [myid: ] - INFO [main:ZooKeeper@442] - Initiating client connection, connectString=localhost:2181 sessio
nTimeout=30000 watcher=org.apache.zookeeper.ZooKeeperMain$MainWatcher@699d8a921
Welcome to ZooKeeper!
2023-06-18 11:56:39,796 [myid: ] - INFO [main-SendThread(localhost:2181):ClientCnxn$SendThread@1025] - Opening socket connection
to server localhost/127.0.0.1:2181. Will not attempt to authenticate using SASL (unknown error)
Jline support is enabled
2023-06-18 11:56:39,835 [myid: ] - INFO [main-SendThread(localhost:2181):ClientCnxn$SendThread@8799] - Socket connection establis
hed to localhost/127.0.0.1:2181, initiating session
2023-06-18 11:56:39,848 [myid: ] - INFO [main-SendThread(localhost:2181):ClientCnxn$SendThread@12999] - Session establishment com
plete on server localhost/127.0.0.1:2181, sessionId= 0x100025ba060b0000, negotiated timeout = 30000

WATCHER::

WatchedEvent state=SyncConnected type=None path=null
[zk:localhost:2181(CONNECTED) 0] █
```

Siamo adesso collegati, come possiamo vedere è tutto andato per il verso giusto e ci viene indicato che siamo connessi al server.

Ora è necessario andare a creare lo znode che sarà necessario al corretto funzionamento, come?

Utilizziamo il seguente comando:

\$ create /election election

```
[zk: localhost:2181(CONNECTED) 0] create /election election
Created /election
[zk: localhost:2181(CONNECTED) 1]
```

Come possiamo osservare dall'immagine, abbiamo creato lo znode **/election** ed in seguito viene assegnata la seguente stringa *election* al nodo.

Successivamente, possiamo verificare che i dati siano stati associati allo znode eseguendo il comando **get /directory**. (/election in questo caso)

```
[zk: localhost:2181(CONNECTED) 1] get /election
election
cZxid = 0x1f
ctime = Sun Jun 18 12:01:17 CEST 2023
mZxid = 0x1f
mtime = Sun Jun 18 12:01:17 CEST 2023
pZxid = 0x1f
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 8
numChildren = 0
```

Esistono diversi comandi che si possono utilizzare in questo caso ma nel nostro non è necessario utilizzarne altri.

Di seguito alcuni comandi presenti e che si potrebbero utilizzare digitando il comando **help**:

```
[zk: localhost:2181(CONNECTED) 2] help
ZooKeeper -server host:port cmd args
  stat path [watch]
  set path data [version]
  ls path [watch]
  delquota [-n|-b] path
  ls2 path [watch]
  setAcl path acl
  setquota -n|-b val path
  history
  redo cmdno
  printwatches on|off
  delete path [version]
  sync path
  listquota path
  rmr path
  get path [watch]
  create [-s] [-e] path data acl
  addauth scheme auth
  quit
  getAcl path
  close
  connect host:port
```

Adesso siamo pronti per poter testare i nostri client. (Coordinatore e NON-Coordinatori)

Abbiamo ancora un piccolo passaggio da effettuare, capire come connettere i vari client tra loro, come impostare il coordinatore, come effettuare l'elezione e l'eventuale disconnessione del coordinatore con successiva nuova elezione del prossimo nodo.

Per poter far questo abbiamo bisogno dell'utilizzo di JAVA e delle API messe a disposizione da Zookeeper.

## 2.3. Java & Zookeeper API

Prima di iniziare facciamo una piccola analisi, sulla logica di elezione del coordinatore. Notiamo subito e bene che è importante monitorare i **guasti del coordinatore**, in modo che un nuovo cliente possa emergere come nuovo leader nel caso in cui il leader attuale fallisca. Una soluzione banale (e quella applicata in questo caso d'uso) è far sì che tutti i processi dell'applicazione monitorino il nodo più piccolo corrente e verifichino se sono diventati il nuovo leader quando il

nodo più piccolo scompare (**noteremo bene che il nodo più piccolo scomparirà se il leader fallisce perché il nodo è effimero**).

**Ciò provoca un effetto di branco:** in caso di guasto del coordinatore attuale, tutti gli altri processi ricevono una notifica ed eseguono *getChildren* su *"/election"* per ottenere l'elenco corrente dei figli di *"/election"*. Il figlio più piccolo diventerà il leader e così via.

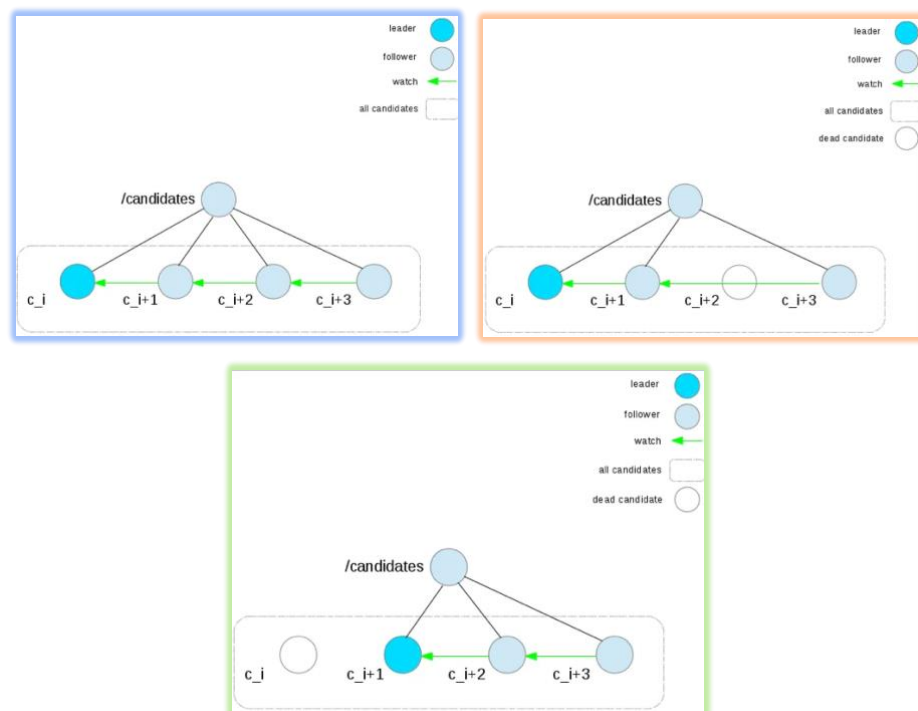
**Sessione:** La connessione tra il client e un nodo nell'ensemble/cluster;

**Watcher:** Trigger monouso sui valori o sui figli del zNode;

Asincroni e si garantisce la visibilità dal client prima del cambio dei dati.

### ESEMPIO:

- Crea un zNode chiamato election;
- Tutti i candidati creano figli sequenziali effimeri (interi crescenti) sotto election;
- Il figlio di election con il numero di sequenza più piccolo è il leader;
- Si osserva il zNode con il numero di sequenza successivo più piccolo per ricevere buone notizie.

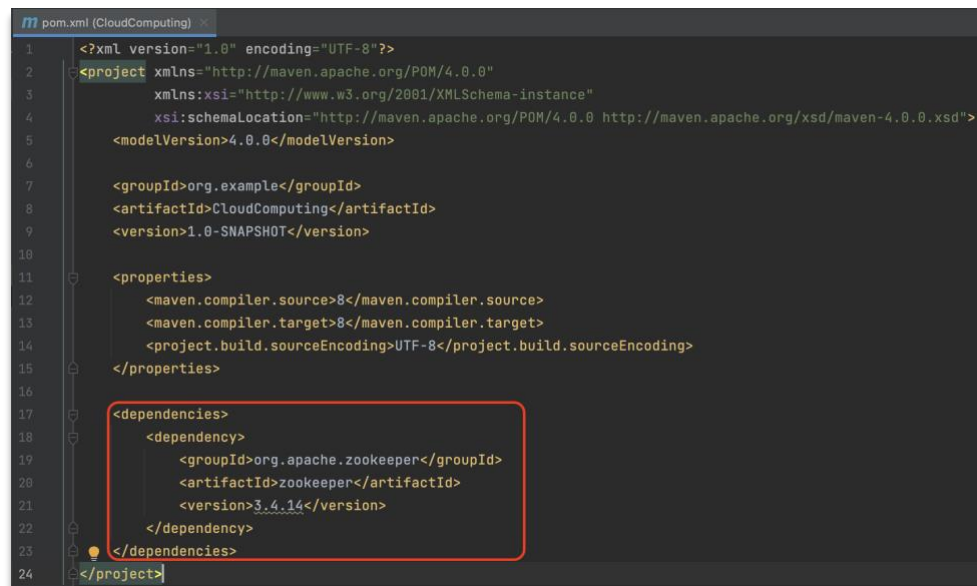


Ora iniziamo ad analizzare effettivamente il codice.

Abbiamo bisogno di creare un **Maven Project** in modo da poter impostare il **POM.XML** che sarà di fondamentale importanza per le dipendenze da dover dichiarare. Per questo progetto è stato utilizzato come **IDE IntelliJ IDEA**.



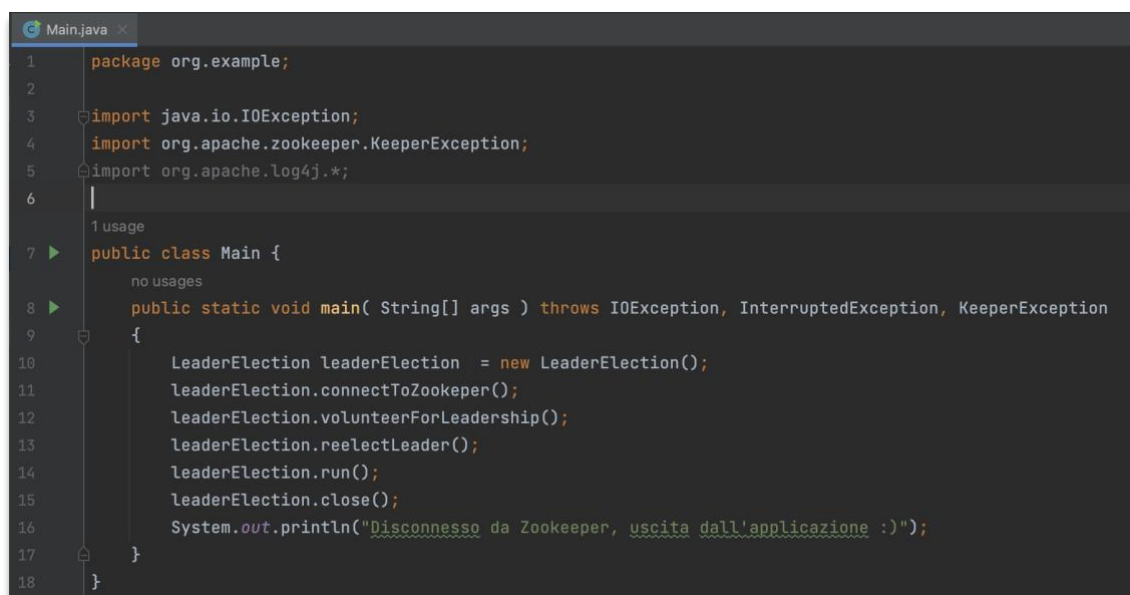
Di seguito vediamo la dichiarazione delle dipendenze con il **package**, il nome dell'**artifact** e la **versione** in utilizzo. Attenzione che è necessario inserirlo il tutto all'interno dei vari tag necessari per dichiarare le dipendenze.



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6
7     <groupId>org.example</groupId>
8     <artifactId>CloudComputing</artifactId>
9     <version>1.0-SNAPSHOT</version>
10
11     <properties>
12         <maven.compiler.source>8</maven.compiler.source>
13         <maven.compiler.target>8</maven.compiler.target>
14         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15     </properties>
16
17     <dependencies>
18         <dependency>
19             <groupId>org.apache.zookeeper</groupId>
20             <artifactId>zookeeper</artifactId>
21             <version>3.4.14</version>
22         </dependency>
23     </dependencies>
24 </project>
```

Possiamo passare ora alle classi e ai metodi creati utilizzando le API, come detto in precedenza, che andranno a gestire il funzionamento dei client.

Andiamo ora a dare un'occhiata alla classe **Main** che conterrà il metodo main necessario all'esecuzione del nostro progetto.



```
1 package org.example;
2
3 import java.io.IOException;
4 import org.apache.zookeeperKeeperException;
5 import org.apache.log4j.*;
6
7 public class Main {
8     public static void main( String[] args ) throws IOException, InterruptedException, KeeperException
9     {
10         LeaderElection leaderElection = new LeaderElection();
11         leaderElection.connectToZookeeper();
12         leaderElection.volunteerForLeadership();
13         leaderElection.reelectLeader();
14         leaderElection.run();
15         leaderElection.close();
16         System.out.println("Disconnesso da Zookeeper, uscita dall'applicazione :)");
17     }
18 }
```

Viene utilizzato il costruttore della classe principale **LeaderElection** che andrà ad utilizzare tutti i suoi metodi necessari al corretto funzionamento con il **run** ed il **close** in parte principale e dopo tutti i metodi che servono per la **connessione**, **scelta del volontario per essere il coordinatore** (*volunteerForLeadership()*) e



la **fase di elezione del nuovo coordinatore** (*reelectLeader()*) in caso di disconnessione di quello attualmente in carica.

Sono stati necessari importare alcune librerie necessarie per le varie eccezioni da gestire **IOException** (eccezioni di input e output) e quelle per il **KeeperException** direttamente necessarie per eccezioni create dall'utilizzo delle API di Zookeeper.

Vediamo adesso la classe principale **LeaderElection** che contiene tutte le API e metodi necessari.

```
1 package org.example;
2
3 import java.io.IOException;
4 import java.util.Collections;
5 import java.util.List;
6
7 import org.apache.zookeeper.CreateMode;
8 import org.apache.zookeeper.KeeperException;
9 import org.apache.zookeeper.WatchedEvent;
10 import org.apache.zookeeper.Watcher;
11 import org.apache.zookeeper.ZooDefs;
12 import org.apache.zookeeper.ZooKeeper;
13 import org.apache.zookeeper.data.Stat;
```

Nell'immagine in precedenza possiamo notare tutte le librerie importate affinché il programma possa funzionare adeguatamente.

Iniziamo ora a vedere le varie costanti, variabili e tutti i metodi necessari.

**volunteerForLeadership());**

```
14
15 public class LeaderElection implements Watcher {
16     private static final String ZOOKEEPER_ADDRESS = "localhost:2181"; //localhost o zoo1
17     private ZooKeeper zooKeeper;
18     private static final int SESSION_TIMEOUT = 3000;
19     private static final String ELECTION_NAMESPACE = "/election";
20     private String currentZnodeName;
21
22     public void volunteerForLeadership() throws KeeperException, InterruptedException {
23         String zNodePrefix = ELECTION_NAMESPACE + "/" + "c_";
24         String zNodeFullPath = zooKeeper.create(zNodePrefix, new byte[] {}, ZooDefs.Ids.OPEN_ACL_UNSAFE, CreateMode.EPHEMERAL_SEQUENTIAL);
25         System.out.println("ZNODE_NAME: " + zNodeFullPath);
26         this.currentZnodeName = zNodeFullPath.replace(target: ELECTION_NAMESPACE + "/", replacement: "");
27     }
28 }
```

**ZOOKEEPER\_ADDRESS:** Indirizzo a cui i client si dovranno connettere; (localhost in locale mentre zoo1 verrà utilizzato per docker)

**SESSION\_TIMEOUT:** tempo di attesa prima del timeout;

**ELECTION\_NAMESPACE:** viene indicato quale sarà il namespace dello znode;

**zooKeeper:** viene creata un'istanza di zookeeper;

**currentZnodeName:** nome dello znode corrente;

**zNodePrefix:** viene aggiunto un prefisso all'election\_namespace visto in precedenza (es. c\_000001 come vedremo in seguito)

**zNodeFullPath:** indirizzo completo per lo znode;

**reelectLeader();**

```
29 public void reelectLeader() throws KeeperException, InterruptedException {
30     Stat predecessorStat = null;
31     String predecessorZnodeName = "";
32     while (predecessorStat == null) {
33         List<String> children = zooKeeper.getChildren(ELECTION_NAMESPACE, watcher, false);
34         Collections.sort(children);
35         String smallestChild = children.get(0);
36         if (smallestChild.equals(currentZnodeName)) {
37             Thread.sleep(2000);
38             System.out.println("Sono il coordinatore :)");
39         } else {
40             Thread.sleep(2000);
41             System.out.println("Non sono il coordinatore :)");
42             int a = (int) Math.floor(Math.random() * (100 - 1 + 1) + 1);
43             int b = (int) Math.floor(Math.random() * (100 - 1 + 1) + 1);
44             int somma = a + b;
45             System.out.println("Il coordinatore mi ha detto di fare " + a + " + " + b + " = " + somma);
46             System.out.println("Adesso rimango in attesa");
47             int predecessorIndex = Collections.binarySearch(children, currentZnodeName) - 1;
48             predecessorZnodeName = children.get(predecessorIndex);
49             predecessorStat = zooKeeper.exists(path: ELECTION_NAMESPACE + "/" + predecessorZnodeName, watcher: this);
50         }
51     }
52     System.out.println("Osservo e attendo: " + predecessorZnodeName);
53     System.out.println();
54 }
```

**predecessorZnodeName:** Nome dello znode precedente;

**List<String> children:** Lista di tutti i figli dello znode che vengono inseriti volta per volta e poi ordinati;

**smallestChildren:** ID del figlio più piccolo; (in posizione 0 della lista)

**predecessorIndex:** Indice del figlio precedente;

**watchTargetZNode(), connectToZookeeper(), run(), close();**

```
56 public void watchTargetZNode() throws KeeperException, InterruptedException {
57     Stat stat = zooKeeper.exists(ELECTION_NAMESPACE, watcher: this);
58     if (stat == null) {
59         return;
60     }
61     byte[] data = zooKeeper.getData(ELECTION_NAMESPACE, watcher: this, stat);
62     List<String> children = zooKeeper.getChildren(ELECTION_NAMESPACE, watcher: this);
63     System.out.println("Data: " + new String(data) + " dei figli: " + children);
64 }
65
66 public void connectToZookeeper() throws IOException{
67     this.zooKeeper = new ZooKeeper(ZOOKEEPER_ADDRESS, SESSION_TIMEOUT, watcher: this);
68 }
69
70 public void run() throws InterruptedException {
71     synchronized (zooKeeper) {
72         zooKeeper.wait();
73     }
74 }
75
76 public void close() throws InterruptedException {
77     zooKeeper.close();
78 }
```

Il metodo **watchTargetZNode** è necessario affinché si verifichi se il namespace dello znode esiste ed in caso lo va ad aggiungere all'interno di un vettore di byte. In seguito, abbiamo i metodi di run in cui si vanno a sincronizzare i vari client e quindi in caso rimanere in attesa mentre il metodo close va a chiudere il client.

**process (WatchedEvent event);**

```
80 @ public void process(WatchedEvent event) {
81     switch(event.getType()) {
82         case None:
83             if (event.getState() == Event.KeeperState.SyncConnected) {
84                 System.out.println("Connessione riuscita a Zookeeper!");
85             } else {
86                 synchronized (zooKeeper) {
87                     System.out.println("Disconnesso da Zookeeper!");
88                     zooKeeper.notifyAll();
89                 }
90             } break;
91         case NodeDeleted:
92             try {
93                 reelectLeader();
94             } catch (KeeperException e) {
95             } catch (InterruptedException e) {
96             }
97             System.out.println(ELECTION_NAMESPACE + " è stato eliminato!!");
98         case NodeCreated:
99             System.out.println(ELECTION_NAMESPACE + " è stato creato!!");
100        case NodeDataChanged:
101            System.out.println(ELECTION_NAMESPACE + " è stato cambiato!!");
102        case NodeChildrenChanged:
103            System.out.println(ELECTION_NAMESPACE + " figli cambiati!!");
104        default:
105            break;
106    }
107
108    try {
109        watchTargetZNode();
110    } catch (KeeperException e) {e.printStackTrace();}
111    catch (InterruptedException e) {e.printStackTrace();}
112
113 }
```

Il metodo **process** gestisce la connessione e la sincronizzazione del client ed in caso di un nodo eliminato (crash, timeout o effettiva cancellazione) fa partire l'elezione del nuovo coordinatore.

Non si può ancora procedere al testing dell'applicazione finché non viene creato il **JAR** del progetto poiché esso necessita di essere avviato diverse volte per poter effettuare una prova adeguata con vari client collegati che dovranno contendersi il posto da coordinatore.

### Come si crea il JAR?

- Direttamente da **IntelliJ**
  - Dal menu principale, seleziona **File | Struttura del progetto** e clicca su **Artifacts**;
  - Clicca sul pulsante **Aggiungi**, seleziona **JAR** e poi seleziona "**Da moduli con dipendenze**". A destra del campo **Classe principale**, clicca sul pulsante **Sfoglia e seleziona la classe principale nella finestra di dialogo** che si apre (ad esempio, HelloWorld (com.example.helloworld)).

- IntelliJ IDEA creerà la configurazione dell'artefatto e mostrerà le impostazioni nella parte destra della finestra di dialogo **Struttura del progetto**. Applica le modifiche e chiudi la finestra di dialogo.
  - Dal menu principale, seleziona **Build | Build Artifacts**.
  - Punta al file **.jar** creato (*CloudComputing.jar*) e seleziona **Build**.
  - Se ora guardiamo nella cartella **out/artifacts**, troveremo il file **.jar** lì.
- Da shell inserendo il comando:

```
$ jar cf file-jar file-input
```

**Le opzioni e gli argomenti utilizzati in questo comando sono:**

**c** indica che desideri creare un file JAR;

**f** indica che desideri che l'output venga scritto su un file anziché sulla standard output (stdout);

**file-jar** è il nome che vuoi assegnare al file JAR risultante.

Una volta creato il JAR finalmente possiamo passare all'effettivo funzionamento!

Come si avvia?

```
$ java -jar path/nomefile.jar
```

## 2.4. Prima fase di test

Facciamo partire il nostro JAR inserendo il comando visto in precedenza, lo facciamo eseguire per 4 volte in 4 terminali diversi per vedere come si comportano i client!

## Prima schermata

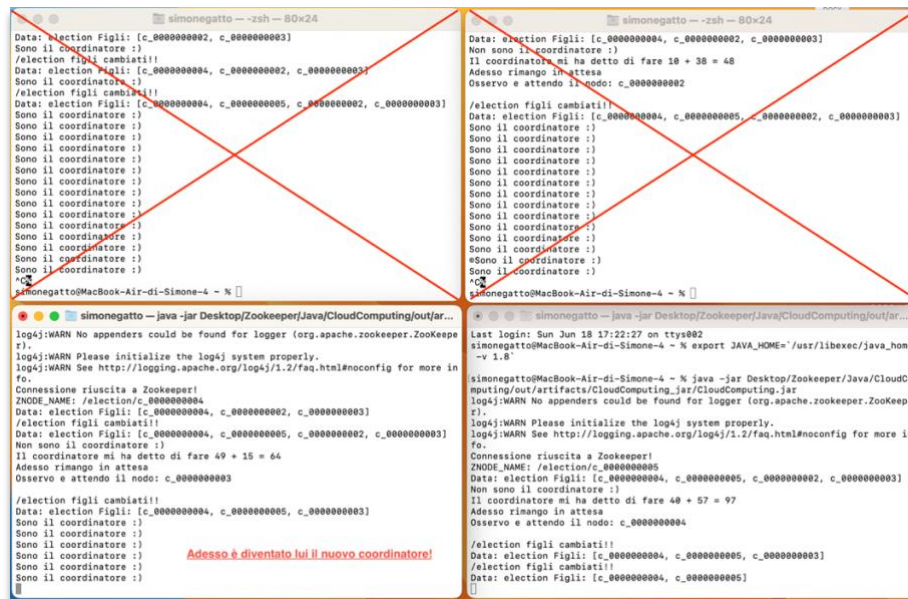
[illegible]

## Seconda schermata

[illegible]



### Terza schermata



Come possiamo osservare, in seguito alla disconnessione del primo coordinatore, il secondo viene eletto e così via perché è stato impostato l'ordine di elezione.

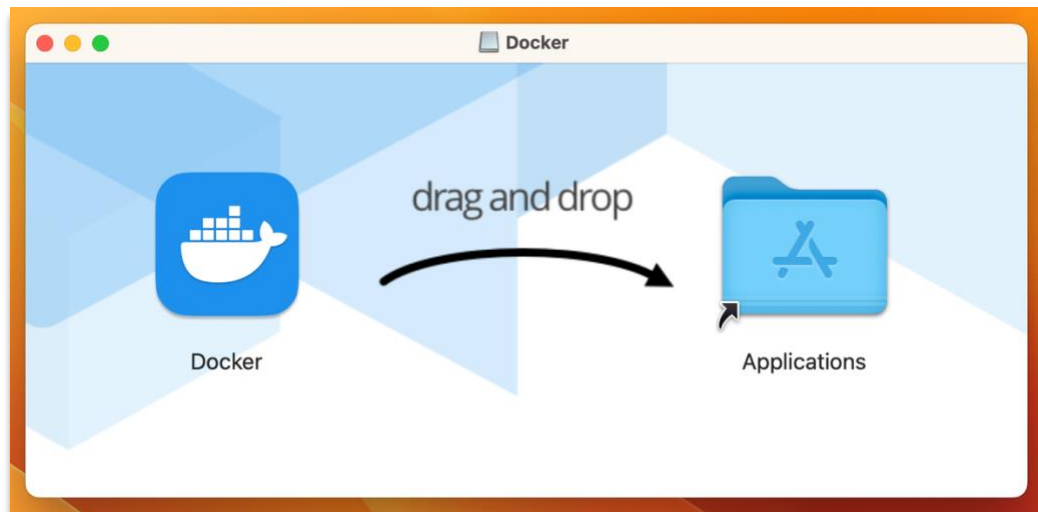
Mentre il coordinatore comunica di esserlo diventato, i figli eseguono un programma e rimangono in attesa dell'elezione per poterlo diventare!

Ogni volta che si collega un nuovo client, viene aggiornata la lista dei figli presenti, infatti, si viene avvisati con un messaggio *"election figli cambiati"*.

## 2.5. Installazione Docker

Passiamo adesso all'installazione di **Docker Desktop** che contiene direttamente **Docker Engine**, **Docker CLI** e **Docker Compose**.

Dopo aver scaricato il software direttamente dal sito ufficiale, che altro non è che un file chiamato **Docker.dmg**, sarà sufficiente cliccarci sopra e apparirà la seguente schermata:



Insieme all'applicazione desktop, viene installata anche la **CLI** di Docker, utilizzabile quindi da riga di comando, attraverso il Terminale. La CLI di Docker sarà particolarmente utile quando andremo a utilizzare docker-compose.

**N.B.** Sarà importante effettuare la registrazione ed il login a Docker Hub per poter pullare le immagini dei container utili.

Ora passiamo alla parte importante di docker, il *pulling* delle immagini dal docker hub per poter utilizzare i nostri software all'interno dei container di docker.

Il comando **docker pull** è un comando Docker per scaricare un'immagine Docker o un repository localmente sull'host da un registro pubblico o privato. Quando eseguiamo un contenitore e l'immagine Docker specificata non è presente localmente, viene prima scaricata. La maggior parte delle volte le immagini vengono scaricate da un registro pubblico chiamato **hub.docker.com** e se creiamo le nostre immagini Docker personalizzate andiamo ad utilizzare l'immagine Docker ufficiale come immagine di base.

Quando eseguiamo il comando pull dalla riga di comando, prima controlla localmente o sull'host la presenza delle immagini e se l'immagine non esiste localmente, il demone Docker si connette al registro pubblico hub.docker.com se

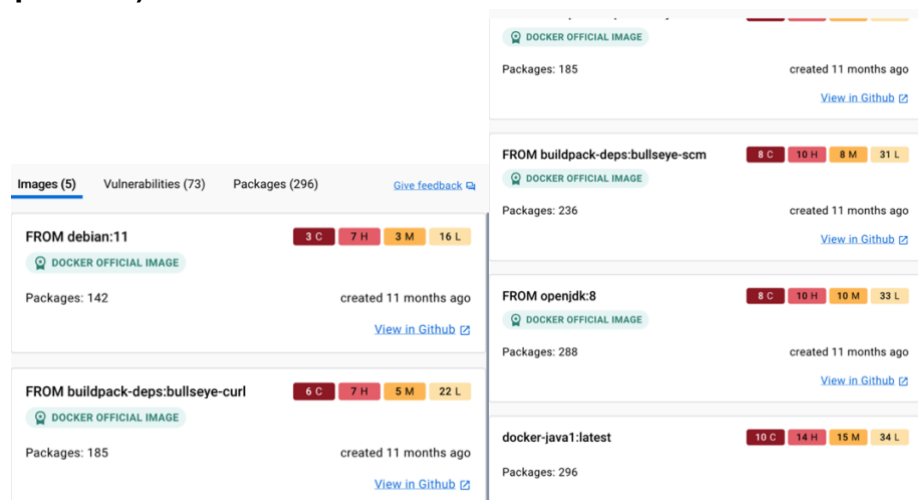
non viene specificato un registro privato nel file **daemon.json** e scarica l'immagine Docker indicata nel comando. Se invece l'immagine viene trovata localmente, vengono controllati gli aggiornamenti e viene scaricata una versione più recente dell'immagine. In realtà, viene effettuato un confronto del digest dell'immagine dietro le quinte. Inoltre, se non specifichiamo una versione, viene scaricata l'immagine con il tag *latest* per impostazione predefinita.

**In questo caso si sono utilizzate diverse immagini che sono stati utilizzare per creare dei container:**

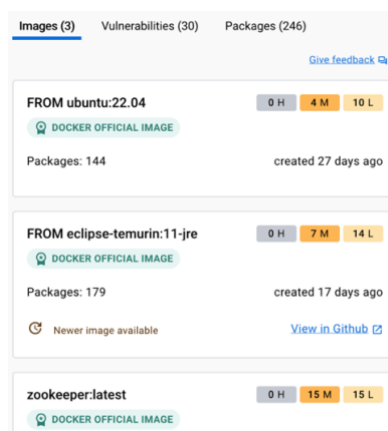
- Zookeeper
- OpenJDK (per l'utilizzo della JDK di Java)

In seguito a queste immagini ne sono state installate altre per via delle dipendenze.

## Java (OpenJDK)



## Zookeeper



Ma come siamo arrivati a ciò visto che è necessario poter far partire diversi container contemporaneamente visto che c'è la necessità di far avviare vari client contemporaneamente? La risposta è semplice, si è utilizzato **docker-compose**!



Docker Compose è un tool ufficiale che consente, in modo piuttosto pratico, di gestire i propri container salvandone la configurazione di istanziamento in un unico file di configurazione in formato **YAML**.

### **A cosa serve?**

Docker Compose serve, per l'appunto, a gestire in modo immediato i propri container. I suoi comandi infatti consentono di:

- Avviare, fermare, e riavviare le applicazioni istanziate;
- Vedere lo stato dei servizi in esecuzione;
- Consultare lo stream dei log dei servizi in esecuzione;
- Seguire comandi all'interno dei container
- Etc.

In pratica, senza l'adozione di Docker Compose ogni qual volta è necessario mettere in esecuzione un container, è necessario conoscerne il comando ed eseguirlo.

Usando Docker Compose affiancato a Docker, ogni qual volta si ha necessità di aggiungere un container al proprio stack è sufficiente aggiungere una porzione di codice di configurazione ad esso relativa al proprio file **docker-compose.yaml**, il quale consentirà a Docker Compose di "conoscere" le caratteristiche del nuovo container (oltre a quelli già presenti) e quindi consentirne una rapida e pratica gestione.

### **Perché si usa?**

Immaginiamo banalmente di dover amministrare un certo numero di container: anche solo pensando all'applicazione che stiamo sviluppando come indicato sopra.

Va da sé che per ognuno sarebbe necessario segnarsi da parte il corretto comando di esecuzione (in base ai tanti parametri diversi di ognuno) per eseguirlo nuovamente ogni qual volta sia necessario cancellare e ricreare il container (per esempio, in caso di aggiornamento dell'immagine per sopravvenuto aggiornamento dell'applicazione).

Con Docker Compose, invece, una volta dichiarate nel suo file di configurazione le configurazioni dei vari container, di esse ci si può dimenticare: basterà infatti utilizzare i suoi comandi rapidi.

Qui di seguito verrà mostrato il docker-compose utilizzato nel nostro caso ed in seguito verrà mostrato come viene avviato e come funziona.

```

services:
  zoo1:
    image: zookeeper
    container_name: zoo1
    restart: always
    hostname: zoo1
    ports:
      - 2181:2181
    environment:
      ZOO_MY_ID: 1
      ZOO_SERVERS: server.1=zoo1:2888:3888;2181
    command: sh -c "zkServer.sh start && echo 'create /election election' | zkCli.sh -server localhost:2181; sleep infinity"
  java1:
    container_name: jav1
    build:
      context: .
      dockerfile: java/Dockerfile
    volumes:
      - ./java:/app
    command: sh -c "sleep 1 && java -jar CloudComputing.jar & sleep 25 && echo CRASH CONTAINER 1"
    depends_on:
      - zoo1
  java2:
    container_name: jav2
    build:
      context: .
      dockerfile: java/Dockerfile
    volumes:
      - ./java:/app
    command: sh -c "sleep 10 && java -jar CloudComputing.jar & sleep 40 && echo CRASH CONTAINER 2"
    depends_on:
      - zoo1
  java3:
    container_name: jav3
    build:
      context: .
      dockerfile: java/Dockerfile
    volumes:
      - ./java:/app
    command: sh -c "sleep 15 && java -jar CloudComputing.jar & sleep 60 && echo CRASH CONTAINER 3"
    depends_on:
      - zoo1

```

## Services:

**zoo1:** container;

**image:** immagine installata nel container;

**container\_name:** nome del container;

**restart:** riavvio del container;

**hostname:** nome dell'host;

**ports:** porte disponibili;

**environment:**

**ZOO\_MY\_ID:** Numero ID corrispondente;

**ZOO\_SERVERS:** indica il server con le varie porte necessarie per le varie funzionalità;

**Command:** comandi da eseguire quando avviato.

## Java1:

**Build:**

**Dockerfile:** posizione del dockerfile rispetto al docker-compose.yml;

**Volumes:** path dove è presente;

**Command:** comandi da eseguire quando avviato;

**Depends on:** vengono indicate le varie dipendenze.

All'interno di **command** troviamo i seguenti comandi, come funzionano?

```
sh -c "zkServer.sh start && echo 'create /election election' | zkCli.sh -server localhost:2181; sleep infinity"
```

**zkServer.sh start:** comando per avvio del server visto in precedenza;  
**echo 'create /election election':** creazione dello znode come visto in precedenza;  
**zkCli.sh -server <ip>:** collegamento della CLI al Server tramite IP;  
**sleep infinity:** il container continua a girare in background senza chiudersi;

```
sh -c "sleep 1 && java -jar CloudComputing.jar & sleep 25 && echo crasho container 1"
```

**sleep:** dormiente fino a tot. secondi;  
**java -jar path/file.jar:** avvio del file JAR per i client;  
**echo crasho container:** mostriamo il crash del container;

Altro importante elemento che è stato utilizzato è il **Dockerfile**!

### Cos'è il Dockerfile?

Fondamentalmente, un **Dockerfile** è un normale file di testo. Il Dockerfile contiene una serie di istruzioni, ciascuna su una riga separata. Le istruzioni vengono eseguite una dopo l'altra per creare un'immagine Docker. Durante l'esecuzione, vengono aggiunti altri livelli all'immagine un po' alla volta.

Un'immagine Docker viene creata eseguendo le istruzioni contenute in un Dockerfile. Questo passaggio è chiamato processo di costruzione e viene avviato eseguendo il comando **"docker build"**. Il **"build context"** è un concetto centrale. Questo definisce a quali file e directory ha accesso il processo di costruzione. Qui, una directory locale serve come sorgente. Quando viene eseguito il comando **"docker build"** il contenuto della directory di origine viene passato al demone Docker. Le istruzioni nel Dockerfile ottengono quindi l'accesso ai file e alle directory nel contesto di costruzione.

```
Users > simonegatto > Desktop > Zookeeper > docker > java > Dockerfile
1 FROM openjdk:8
2 WORKDIR /app
3 COPY . .
4 CMD ["java", "-jar", "CloudComputing.jar"]
```

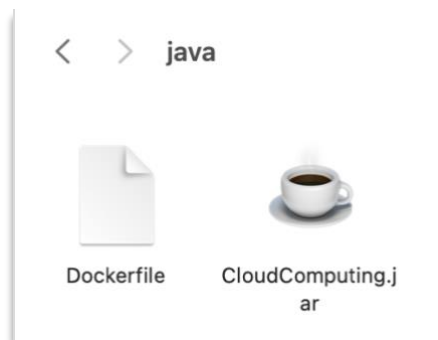
**FROM:** Imposta l'immagine di base, deve apparire come prima istruzione e permette solo un'entrata per ogni fase di costruzione;

**WORKDIR:** Cambia la directory corrente;

**COPY:** Copia file e directory nell'immagine e crea un nuovo livello;

**CMD:** Imposta l'argomento di default per l'avvio del container e si ha un solo record per ogni fase di costruzione;

Insieme al dockerfile è presente nella stessa directory, come indicato nel CMD, il file .jar che verrà eseguito.



### 3. Esecuzione finale

Siamo arrivati all'esecuzione finale in cui si andranno a comporre i vari docker container con le varie immagini necessarie per la loro costruzione. Tutto questo verrà effettuato, come detto in precedenza, il docker-compose.yaml, andando ad utilizzare la Docker CLI utilizzando alcuni comandi importanti al suo corretto funzionamento.

I principali comandi che andremo ad utilizzare sono due:

```
$ docker-compose build
```

Costruisce o ricostruisce i servizi (builda solamente le immagini senza far partire i container). I servizi vengono costruiti una volta e quindi etichettati, di default come *"project\_service"*.

Se il file **Compose** specifica un nome di immagine, l'immagine viene etichettata con tale nome, sostituendo eventuali variabili in precedenza.

Se **Dockerfile** di un servizio o i contenuti della sua directory di build vengono modificati, è necessario eseguire nuovamente *"docker compose build"* per ricostruirlo.

```
simonegatto@MacBook-Air-di-Simone-4 docker % docker-compose build
[+] Building 1.3s (16/18)
=> [java3 internal] load build definition from Dockerfile
=> => transferring dockerfile: 535B
=> [java3 internal] load .dockerignore
=> => transferring context: 2B
=> [java3 internal] load metadata for docker.io/library/openjdk:8
=> [java2 internal] load .dockerignore
=> => transferring context: 2B
=> [java2 internal] load build definition from Dockerfile
=> => transferring dockerfile: 535B
=> [java1 internal] load .dockerignore
=> => transferring context: 2B
=> [java1 internal] load build definition from Dockerfile
=> => transferring dockerfile: 535B
=> [java3 1/3] FROM docker.io/library/openjdk:8@sha256:86e863cc57215cfb1
=> [java3 internal] load build context
=> => transferring context: 1.05kB
=> [java2 internal] load build context
=> => transferring context: 1.05kB
=> [java1 internal] load build context
=> => transferring context: 1.05kB
=> CACHED [java3 2/3] WORKDIR /app
=> CACHED [java3 3/3] COPY . .
=> [java1] exporting to image
=> => exporting layers
=> => writing image sha256:faffec11fbf3df291131baf2e5bfe0dc66237301e6a1e
=> => naming to docker.io/library/docker-java1
=> [java3] exporting to image
=> => exporting layers
=> => writing image sha256:726cd09d86c783ee86ea8aeb187e846601a795c54907f
=> => naming to docker.io/library/docker-java3
=> [java2] exporting to image
=> => exporting layers
=> => writing image sha256:f2d184ca28c667acadcbda84c2fbf106edf8b48ff602d
=> => naming to docker.io/library/docker-java2
```

```
$ docker-compose up
```

Costruisce, (ri)crea, avvia e si collega ai contenitori di un servizio (costruisce le immagini se non esistono e avvia i contenitori.)

A meno che non siano già in esecuzione, questo comando avvia anche tutti i servizi collegati.

Il comando aggrega l'output di ogni container. È possibile selezionare facoltativamente un sottoinsieme di servizi a cui collegarsi utilizzando l'opzione `--attach`, o escludere alcuni servizi utilizzando l'opzione `--no-attach` per evitare che l'output sia sovraccaricato da alcuni servizi verbosi.

Quando il comando termina, tutti i container vengono arrestati. Eseguire `docker compose up --detach` avvia i container in background e li lascia in esecuzione.

Se esistono già dei container per un servizio e la configurazione o l'immagine del servizio sono state modificate dopo la creazione dei container, `docker compose up` rileva le modifiche arrestando e ricreando i container (mantenendo i volumi montati). Per evitare che Compose rilevi le modifiche, utilizzare l'opzione `--no-recreate`.

```
simonegatto@MacBook-Air-di-Simone-4 docker % docker-compose up
[+] Running 9/9
 ✓ zoo1 8 layers [#####] 0B/0B Pulled
 ✓ a1df1d4a17c6 Pull complete
 ✓ f1a707fe2fc3 Pull complete
 ✓ 86e5488ab474 Pull complete
 ✓ bf2dbf45ca92 Pull complete
 ✓ a224ecdb8428 Pull complete
 ✓ 8b64c32f7e5b Pull complete
 ✓ 5b147ea25b59 Pull complete
 ✓ 1f15f076b9c6 Pull complete
[+] Building 0.0s (0/0)
[+] Running 4/1
 ✓ Container zoo1 Created
 ✓ Container jav3 Created
 ✓ Container jav1 Created
 ✓ Container jav2 Created
Attaching to jav1, jav2, jav3, zoo1
```

Se aggiungi l'opzione `--build`, verrà forzata la costruzione delle immagini anche quando non necessario.

Di seguito, possiamo osservare il completo funzionamento di Apache Zookeeper, l'avvio del JAR e le funzionalità di docker tutte insieme. Vengono avviati 3 processi dal JAR contemporaneamente per osservare come viene gestita l'elezione e la ri-elezione. Il tutto viene amministrato, come visto sopra, all'interno del docker-

compose in cui troviamo i vari comandi che fanno crashare alcuni client per poter osservare il funzionamento dell'elezione del coordinatore.

[illegible]

```

simonegatto@MacBook-Air-di-Simone-4: ~$ docker-compose up
[+] Building 0.0s (0/0)
[+] Running 4/0
   ✓ Container zoo1   Running    0.0s
   ✓ Container jav1   Created    0.0s
   ✓ Container jav2   Created    0.0s
   ✓ Container jav3   Created    0.0s
Attaching to jav1, jav2, jav3, zoo1
jav1 | log4j:WARN No appenders could be found for logger (org.apache.zookeeper.ZooKeeper).
jav1 | log4j:WARN Please initialize the log4j system properly.
jav1 | log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
jav1 | Connessione riuscita a Zookeeper!
jav1 | ZNODE_NAME: /election/c_0000000012
jav1 | Data: election dei figli: [c_0000000012]
jav1 | Sono il coordinatore :)
jav1 | Sono il coordinatore :)
jav1 | Sono il coordinatore :)
jav1 | Sono il coordinatore :)
jav2 | log4j:WARN No appenders could be found for logger (org.apache.zookeeper.ZooKeeper).
jav2 | log4j:WARN Please initialize the log4j system properly.
jav2 | log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
jav2 | Connessione riuscita a Zookeeper!
jav2 | /election figli cambiati!!
jav2 | ZNODE_NAME: /election/c_0000000013
jav1 | Data: election dei figli: [c_0000000013, c_0000000012]
jav2 | Data: election dei figli: [c_0000000013, c_0000000012]
jav1 | Sono il coordinatore :)
jav2 | Non sono il coordinatore :)
jav2 | Il coordinatore mi ha detto di fare 75 + 4 = 79
jav2 | Adesso rimango in attesa
jav2 | Osservo e attendo: c_0000000012
jav2 |
jav1 | Sono il coordinatore :)
jav3 | log4j:WARN No appenders could be found for logger (org.apache.zookeeper.ZooKeeper).
jav3 | log4j:WARN Please initialize the log4j system properly.
jav3 | log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
jav3 | Connessione riuscita a Zookeeper!
jav3 | /election figli cambiati!!
jav3 | ZNODE_NAME: /election/c_0000000014
jav3 | /election figli cambiati!!
jav2 | Data: election dei figli: [c_0000000013, c_0000000014, c_0000000012]
jav1 | Data: election dei figli: [c_0000000013, c_0000000014, c_0000000012]
jav3 | Data: election dei figli: [c_0000000013, c_0000000014, c_0000000012]
jav1 | Sono il coordinatore :)
jav3 | Non sono il coordinatore :)
jav3 | Il coordinatore mi ha detto di fare 2 + 81 = 83
jav3 | Adesso rimango in attesa
jav3 | Osservo e attendo: c_0000000013
jav3 |
jav1 | Sono il coordinatore :)
jav1 | Sono il coordinatore :)
jav1 | Sono il coordinatore :)
jav1 | Sono il coordinatore :)
jav1 | CRASH CONTAINER 1
jav1 | exited with code 0
jav3 | /election figli cambiati!!
jav3 | Data: election dei figli: [c_0000000013, c_0000000014]
jav2 | Sono il coordinatore :)

```

## 4. Conclusioni

L'elezione del leader è la semplice idea di conferire a qualcosa (un processo, un host, un thread, un oggetto o una persona) in un sistema distribuito alcuni poteri speciali. Questi poteri speciali potrebbero includere la capacità di assegnare lavoro, la capacità di modificare un pezzo di dati o persino la responsabilità di gestire tutte le richieste nel sistema.

L'elezione del leader è uno strumento potente per migliorare l'efficienza, ridurre la coordinazione, semplificare le architetture e ridurre le operazioni.

Il connubio tra le due (Zookeeper e Docker) è veramente molto utile in quanto anche Docker è una piattaforma software che permette di creare, testare e distribuire applicazioni con la massima rapidità. Docker raccoglie il software in unità standardizzate chiamate container che offrono tutto il necessario per la loro corretta esecuzione, incluse librerie, strumenti di sistema, codice e runtime. Con Docker, è possibile distribuire e ricalibrare le risorse per un'applicazione in qualsiasi ambiente, tenendo sempre sotto controllo il codice eseguito.