

# Progetto B1: Multicast totalmente e causalmente ordinato in Go

---

MARCO CALAVARO

MATRICOLA: 0295233

# Indice

---

- Introduzione
- Descrizione dell'architettura
- Descrizione dell'implementazione
- Algoritmi
- Testing

# Introduzione

---

Il progetto prevedeva di realizzare un servizio per registrare dei processi a un gruppo di comunicazione, e di realizzare i seguenti algoritmi di multicast:

1. multicast totalmente ordinato implementato in modo centralizzato tramite un sequencer
2. multicast totalmente ordinato implementato in modo decentralizzato tramite l'uso di clock logici scalari
3. multicast causalmente ordinato implementato in modo decentralizzato tramite l'uso di clock logici vettoriali

Il progetto è stato realizzato tramite il linguaggio di programmazione Go

Per il deploy dell'architettura si è sfruttato docker

Per l'orchestrazione dei container si è usato docker-compose

# Descrizione dell'architettura

## File di configurazione

---

Il progetto è stato sviluppato seguendo l'idea di separare il codice dell'applicazione dalla tecnologia utilizzata per il deploy.

Inizialmente si è quindi scelto di configurare un file go per la gestione dei parametri.

File: `/utility/static_config.go`

Per riportare qui pochi parametri necessari a Docker compose per istanziare l'architettura si utilizza un secondo file

File: `.env`

La congruenza tra i due file è forzata ad essere mantenuta dall'utente.

È stata comunque valutata la soluzione con il passaggio dei parametri direttamente tramite env, ma poiché in `static_config.go` vi sono parametri non gestiti da Docker si è mantenuta tale soluzione.

# Descrizione dell'architettura

## Nodi sviluppati

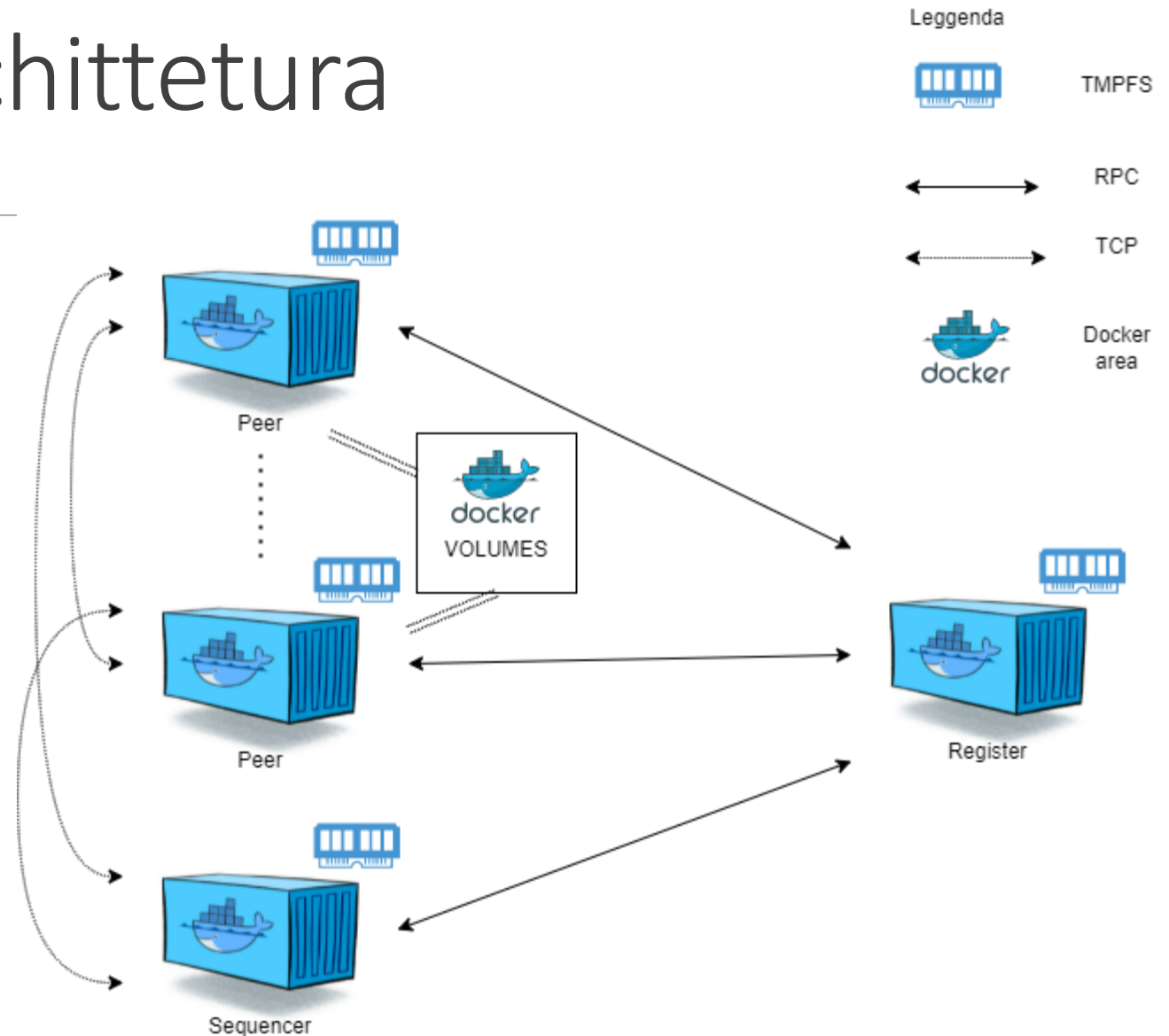
Sono stati sviluppati i 3 seguenti tipi di nodi:

- Register
- Sequencer
- Peer

Le tecnologie per la persistenza usate sono:

- TMPFS
- Volumes

La figura di lato rappresenta l'architettura realizzata e i vari collegamenti tra i vari elementi.



# Descrizione dell'implementazione

## Comunicazione

---

Per poter realizzare la comunicazione tra i nodi sono state usate le seguenti tecnologie

- Remote Procedure Call (rpc) implementato tramite il package net/rpc
- Protocollo applicativo che si appoggia direttamente su comunicazione TCP (package net)

La struttura per l'invio dei messaggi di multicast è la seguente

La ricezione dei messaggi avviene su un'unica porta, si applica l'opportuna «risposta» basandosi sul campo Type

I campi della struttura sono utilizzati in modo differente in base al tipo del messaggio

Vi sono 4 tipi:

- SeqMsg
- ScalarClockMsg
- VectorClockMsg
- ScalarAck

---

---

```
type Message struct {  
    Type      MsgType  
    SendID    int  
    Date      string  
    Text      string  
    SeqNum    []uint64  
}
```

---

---

# Algoritmi

## Algoritmo 1 Sequencer

---

Il messaggio da inviare viene mandato “raw” al sequencer che lo processa internamente , creando la struct *Message* apposita e inviandola a tutti i peer.

```
type Message struct {  
    Type           MsgType           SeqMsg  
    SendID         int                0  
    Date           string             Actual Date  
    Text           string             Msg  
    SeqNum         []uint64           SeqNum  
}
```

La ricezione è gestita da:

- Buffer di ricezione
- Demone (goroutine seqMsg\_reordering)

# Algoritmi

## Algoritmo 2 Scalar Clock

---

Il peer che vuole inviare un messaggio crea l'apposita struct *Message*

```
type Message struct {  
    Type           MsgType           ScalarClockMsg  
    SendID         int               SenderID  
    Date           string            Actual Date  
    Text           string            Msg  
    SeqNum         []uint64         ScalarClock Value  
}
```

La ricezione viene gestita da:

- Queue, implementata tramite package List e alcune funzioni di supporto
- Goroutine per gestione del messaggio
- Goroutine per gestione dell'ack



# Algoritmi

## Algoritmo 3 Vector Clock

---

Il peer che vuole inviare un messaggio crea l'apposita struct *Message*

```
type Message struct {  
    Type           MsgType           VectorClockMsg  
    SendID         int               SenderID  
    Date           string            Actual Date  
    Text           string            Msg  
    SeqNum         []uint64          VectorClock Value  
}
```

La ricezione viene gestita da:

- Buffer di ricezione
- Demone (goroutine vectMsg\_reordering)

# Testing

---

L'architettura di test è montata direttamente sul peer.

Per poter lanciare i test la variabile di configurazione statica `Lanch_Test` deve essere settata a `true`.

Sono stati prodotti 6 test, 2 per tipologia di algoritmo con la seguente distinzione:

1. Il primo test prevede l'invio di messaggi da parte di un solo peer
2. Il secondo test prevede che tutti i peer inviino messaggi

I test sviluppati risultano avere la stessa logica, ma i valori attesi dipendono dall'algoritmo che si sta testando

Di seguito i vari valori attesi

# Testing

---

## Algoritmo 1:

- Test sender singolo: un solo peer effettua l'invio di un certo numero di messaggi, ci si aspetta che i messaggi vengano consegnati a livello applicativo nello stesso ordine
- Test multi sender: ciascun peer invia dei messaggi, ci si aspetta che i messaggi vengano consegnati a livello applicativo nello stesso ordine

## Algoritmo 2:

- Test sender singolo: un solo peer effettua l'invio di un certo numero di messaggi, ci si aspetta che nessun peer riesca a consegnare i messaggi a livello applicativo
- Test multi sender: ciascun peer invia dei messaggi, ci si aspetta che non vengano consegnati tutti i messaggi ma solo una parte (dovuto al funzionamento dell'algoritmo)

## Algoritmo 3:

- Test sender singolo: un solo peer effettua l'invio di un certo numero di messaggi, ci si aspetta che i messaggi vengano consegnati a livello applicativo nello stesso ordine
- Test multi sender: ciascun peer invia dei messaggi, ci si aspetta che i messaggi vengano consegnati a livello applicativo nello stesso ordine
- Test aggiuntivo: è stato riprodotto il test che simula lo schema presentatoci a lezione

# Piattaforma software

---

Di seguito è riportata la piattaforma software utilizzata per lo sviluppo del codice.

- Sistema operativo Ubuntu 20.04 LTS
- Linguaggio programmazione Go
- Docker per la creazione di container
- Docker-compose per orchestrazione
- Github per versioning del progetto

I punti rossi rappresentano le piattaforme necessarie al funzionamento dell'applicazione.

È stato preso spunto dalla libreria riportata al seguente [link](#) per lo sviluppo del menù.