

4º ATIVIDADE de CES-27 / 2019

CTA - ITA - IEC

Prof Hirata e Prof Juliana

Objetivo: Trabalhar com o protocolo Raft para consenso distribuído. **Vamos focar no algoritmo de eleição do Raft.** Não vamos trabalhar com *log replication*.

Entregar (através do GoogleClassroom): Códigos finais (arquivos .go) e relatório. O relatório deve indicar detalhes particulares/críticos do código, além apresentar as tarefas realizadas e comentar resultados. Deve ser justificada qualquer ajuste no código ou no seguimento das instruções.

Curiosidade:

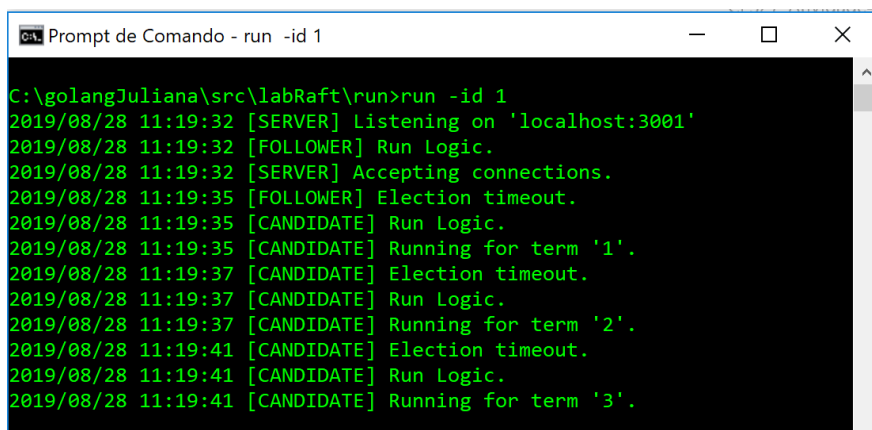
Todas chamadas entre instâncias (*servers*) são feitas via RPC (*remote procedure call*).

Atenção:

- Este lab considera que teremos 5 instâncias (*servers*) com ids 1, 2 ... 5.
- Cada instância já tem sua porta definida (vide arquivo `labRaft/run/run.go`).
Ex: porta 3001 para instância 1, e assim por diante.
- Para encerrar uma instância usar `Ctrl+C` do teclado. Isso pode ser usado para simular que a instância falhou.

Primeiramente vamos compreender como usar o código fornecido.

- Baixar `labRaft.zip`.
- Descompactar essa pasta no seu `$GOPATH/src`
 - `$GOPATH` é o diretório onde você armazena seus códigos Go.
 - Dentro de `$GOPATH/src` deve então ficar a pasta `labRaft` com três pastas (`raft`, `run` e `util`)
- Rodar apenas uma instância para analisar o comportamento:
 - Entrar em `$GOPATH/src/labRaft/run`
 - Executar: `run -id 1`
 - A instância fica tentando se eleger, mas está sozinha.



```
C:\golangJuliana\src\labRaft\run>run -id 1
2019/08/28 11:19:32 [SERVER] Listening on 'localhost:3001'
2019/08/28 11:19:32 [FOLLOWER] Run Logic.
2019/08/28 11:19:32 [SERVER] Accepting connections.
2019/08/28 11:19:35 [FOLLOWER] Election timeout.
2019/08/28 11:19:35 [CANDIDATE] Run Logic.
2019/08/28 11:19:35 [CANDIDATE] Running for term '1'.
2019/08/28 11:19:37 [CANDIDATE] Election timeout.
2019/08/28 11:19:37 [CANDIDATE] Run Logic.
2019/08/28 11:19:37 [CANDIDATE] Running for term '2'.
2019/08/28 11:19:41 [CANDIDATE] Election timeout.
2019/08/28 11:19:41 [CANDIDATE] Run Logic.
2019/08/28 11:19:41 [CANDIDATE] Running for term '3'.
```

- Rodar o lab (é assim que vamos fazer daqui para frente!)
 - Entrar em \$GOPATH/src/labRaft/run
 - Abrir 5 terminais e subir uma instância em cada terminal


```
run -id 1
...
run -id 5
```
 - O código atual está incompleto, por isso a eleição nunca termina. Sempre tem alguém como candidato (devido ao *timeout*), mas a eleição não encerra.

The image displays five terminal windows, each running the Raft algorithm with a different ID. The logs show the following sequence of events:

- Terminal -id 1:** Starts listening on localhost:3001. It receives a vote denial from localhost:3003 for term '0'. It then runs logic for term '1' and receives multiple vote denials from localhost:3003, 3004, 3005, 3002, and 3001.
- Terminal -id 3:** Starts as a candidate. It receives vote denials from localhost:3001, 3005, and 3002 for term '2'. It then runs logic for term '2' and receives a vote denial from localhost:3001.
- Terminal -id 2:** Starts listening on localhost:3002. It receives a vote denial from localhost:3003 for term '0'. It then runs logic for term '1' and receives multiple vote denials from localhost:3005, 3003, 3004, 3001, and 3003.
- Terminal -id 4:** Starts as a follower. It receives a vote denial from localhost:3003 for term '0'. It then receives a vote denial from localhost:3001 for term '0' and an election timeout.
- Terminal -id 5:** Starts as a follower. It receives a vote denial from localhost:3003 for term '0'. It then receives a vote denial from localhost:3001 for term '0' and a vote denial from localhost:3002 for term '0'.

Tarefa 1. Complete o código fornecido para termos o algoritmo de eleição funcionando. A figura abaixo é original do artigo. Colocamos em vermelho as partes abordadas nessa implementação.

State	
Persistent state on all servers: (Updated on stable storage before responding to RPCs)	
currentTerm	latest term server has seen (initialized to 0 on first boot, increases monotonically)
votedFor	candidateId that received vote in current term (or null if none)
log[]	log entries; each entry contains command for state machine, and term when entry was received by leader (first index is 1)
Volatile state on all servers:	
commitIndex	index of highest log entry known to be committed (initialized to 0, increases monotonically)
lastApplied	index of highest log entry applied to state machine (initialized to 0, increases monotonically)
Volatile state on leaders: (Reinitialized after election)	
nextIndex[]	for each server, index of the next log entry to send to that server (initialized to leader last log index + 1)
matchIndex[]	for each server, index of highest log entry known to be replicated on server (initialized to 0, increases monotonically)

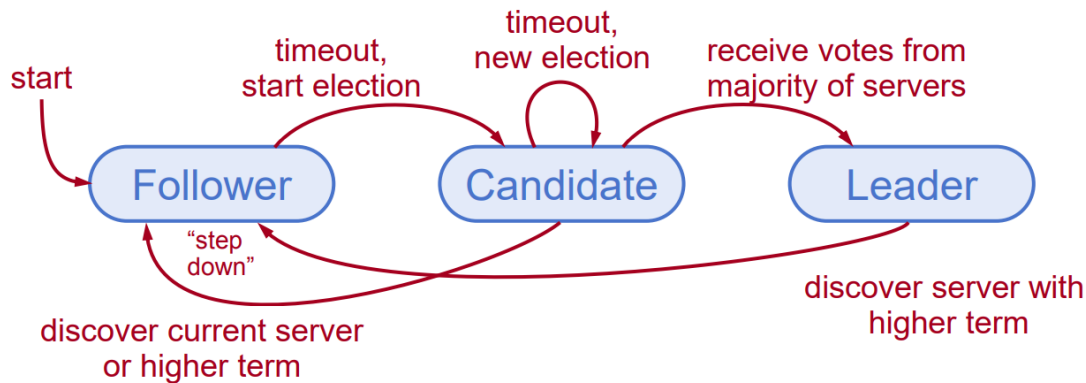
AppendEntries RPC	
Invoked by leader to replicate log entries (§5.3); also used as heartbeat (§5.2).	
Arguments:	
term	leader's term
leaderId	so follower can redirect clients
prevLogIndex	index of log entry immediately preceding new ones
prevLogTerm	term of prevLogIndex entry
entries[]	log entries to store (empty for heartbeat; may send more than one for efficiency)
leaderCommit	leader's commitIndex
Results:	
term	currentTerm, for leader to update itself
success	true if follower contained entry matching prevLogIndex and prevLogTerm
Receiver implementation:	
1. Reply false if term < currentTerm (§5.1)	
2. Reply false if log doesn't contain an entry at prevLogIndex whose term matches prevLogTerm (§5.3)	
3. If an existing entry conflicts with a new one (same index but different terms), delete the existing entry and all that follow it (§5.3)	
4. Append any new entries not already in the log	
5. If leaderCommit > commitIndex, set commitIndex = min(leaderCommit, index of last new entry)	

RequestVote RPC	
Invoked by candidates to gather votes (§5.2).	
Arguments:	
term	candidate's term
candidateId	candidate requesting vote
lastLogIndex	index of candidate's last log entry (§5.4)
lastLogTerm	term of candidate's last log entry (§5.4)
Results:	
term	currentTerm, for candidate to update itself
voteGranted	true means candidate received vote
Receiver implementation:	
1. Reply false if term < currentTerm (§5.1)	
2. If votedFor is null or candidateId, and candidate's log is at least as up-to-date as receiver's log , grant vote (§5.2, §5.4)	

Rules for Servers	
All Servers:	
• If commitIndex > lastApplied : increment lastApplied, apply log[lastApplied] to state machine (§5.3)	
• If RPC request or response contains term T > currentTerm: set currentTerm = T, convert to follower (§5.1) Step down	
Followers (§5.2):	
• Respond to RPCs from candidates and leaders	
• If election timeout elapses without receiving AppendEntries RPC from current leader or granting vote to candidate: convert to candidate	
Candidates (§5.2):	
• On conversion to candidate, start election:	
• Increment currentTerm	
• Vote for self	
• Reset election timer	
• Send RequestVote RPCs to all other servers	
• If votes received from majority of servers: become leader	
• If AppendEntries RPC received from new leader: convert to follower	
• If election timeout elapses: start new election	
Leaders:	
• Upon election: send initial empty AppendEntries RPCs (heartbeat) to each server; repeat during idle periods to prevent election timeouts (§5.2)	
• If command received from client: append entry to local log, respond after entry applied to state machine (§5.3)	
• If last log index ≥ nextIndex for a follower: send AppendEntries RPC with log entries starting at nextIndex	
• If successful: update nextIndex and matchIndex for follower (§5.3)	
• If AppendEntries fails because of log inconsistency: decrement nextIndex and retry (§5.3)	
• If there exists an N such that N > commitIndex, a majority of matchIndex[i] ≥ N, and log[N].term == currentTerm: set commitIndex = N (§5.3, §5.4).	

Informações:

1. De forma geral, este é o comportamento desejado:



Como vamos focar no algoritmo de eleição do Raft (sem *log replication*), temos que adaptar um pouco o comportamento durante a eleição. Abaixo seguem as dicas.

- 1) Ao receber um `AppendEntry` (*heartbeat*):

Se follower:

```
// Quem mandou acha que é líder, mas não é. Então 'rejeito' essa msg.
Se (term < currentTerm), retornar false

// Quem mandou é o líder de um novo term, então me atualizo.
Se (term > currentTerm), fazer currentTerm = term e votedFor = 0

// Como existe um líder do meu term ou de um novo term (já atualizado acima), então 'limpo' meu timeout
reset electionTimeout
retornar true
```

Se candidate:

```
// Quem mandou acha que é líder, mas não é. Então 'rejeito' essa msg.
Se (term < currentTerm), retornar false

// Quem mandou é o líder de um novo term, então me atualizo.
Se (term > currentTerm), fazer currentTerm = term e votedFor = 0

// Como existe um líder do meu term ou de um novo term (já atualizado acima), então volto a ser follower
Step down!
```

Se leader: idem candidate

2) Ao receber um RequestVote:

Se follower:

```
// Quem mandou quer ser leader num term velho. Então nego o voto.
Se (term < currentTerm), deny vote

// Quem mandou quer ser o líder num novo term, então me atualizo.
Se (term > currentTerm), fazer currentTerm = term e votedFor = 0

// Quem mandou quer ser o líder no meu term ou no novo (já atualizado
acima). Dou meu voto, se eu não votei ou se já votei nesse mesmo.
Se (term == currentTerm) E (votedFor é 0 ou candidateId) {
    votedFor = candidateId
    reset electionTimeout
    grant vote
}
Else
    Deny vote
```

Se candidate:

```
//Obs: No início do comportamento do candidate já tem o voto para si
mesmo.

// Quem mandou quer ser leader num term velho. Então nego o voto.
Se (term < currentTerm), deny vote

// Quem mandou quer ser o líder num novo term, então me atualizo e
volto a ser follower.
Se (term > currentTerm) {
    fazer currentTerm = term e votedFor = 0
    Step down!
}

// Quem mandou quer ser o líder no meu term. Dou meu voto, se eu não
votei ou se já votei nesse mesmo. Daí eu adio a minha própria eleição
('resetando' o timeout).
Se (term == currentTerm) E (votedFor é 0 ou candidateId) {
    votedFor = candidateId
    reset electionTimeout
    grant vote
}
Else
    Deny vote
```

Se leader: idem candidate

c. Comportamento do *leader* em `func (raft *Raft) leaderSelect()`

Obs: O canal `replyChan` aqui recebe respostas dos `AppendEntry` enviados por mim (líder). Basicamente faço o seguinte: Se receber falso, é porque existe outro líder mais atual, então faço `Step down`!

```
func (raft *Raft) leaderSelect() {
    (...)
    for {
        select {
            (...)
            case aet := <-replyChan:
                // Append Entry Responses
                //////////////////////////////////////////////////
                // MODIFY HERE //
                (...)
                // END OF MODIFY //
                //////////////////////////////////////////////////

            case rv := <-raft.requestVoteChan:
                // Request Vote Operation
                //////////////////////////////////////////////////
                // MODIFY HERE //
                (...)
                // END OF MODIFY //
                //////////////////////////////////////////////////

            case ae := <-raft.appendEntryChan:
                // Append Entry Operation
                //////////////////////////////////////////////////
                // MODIFY HERE //
                (...)
                // END OF MODIFY //
                //////////////////////////////////////////////////
        }
    }
}
```

3. Lembre-se de colocar mensagens para indicar todas as ações do server. Ex:

```
log.Printf("[FOLLOWER] Update old term '%v' to new term '%v' from '%v'.\n", raft.currentTerm, rv.Term, raft.peers[rv.CandidateID])

log.Printf("[CANDIDATE] Stepping down.\n")
```

4. Ao alterar o `currentTerm` em uma instância, você deve sempre alterar o `votedFor`, normalmente deixando-o em 0 (ou seja, não votou naquele *term*). Obs: Isso já foi até indicado no pseudo-código.

5. Tenha certeza de *resetar* o `electionTimer` quando:

- Ao receber um `AppendEntry` do líder atual, ou seja, quando este possuir um `term` válido;
- Ao iniciar uma eleição;
- Ao votar em um candidato.

Obs: Isso já foi até indicado no pseudo-código.

6. Numa operação de *Step Down* (de *leader* ou *candidate*), você deve fazer a alteração no estado imediatamente (usando `raft.currentState.Set(follower)`) e continuar a partir desse novo estado (usando `return`). Antes disso, para não perder a chamada atual, basta recolocar o objeto da chamada no canal original (ex: `raft.appendEntryChan <- x`). Veja o exemplo completo abaixo. Não precisa *resetar* o `electionTimeout` antes do *step down*, pois o código atual já faz isso ao entrar no modo *follower*.

```
// Step down
log.Printf("[LEADER] Stepping down.\n")
raft.currentState.Set(follower)
raft.appendEntryChan <- x //Usar o canal e objeto corretos
return
```

Exemplo de funcionamento com todas instâncias funcionando:

- Server 1 vira *candidate*
- Server 2 também vira *candidate*
- Server 1 nega voto de 2 (pois 1 já é *candidate* nesse *term*)
- Server 1 tem voto rejeitado por 2 (que também é *candidate*)
- Server 1 tem voto rejeitado por 3 (pois ele já votou no 2)
- Server 1 recebe votos de 5 e 4, por isso vira *leader*
- Server 2 não consegue maioria dos votos
- Server 2 faz *Step Down*, pois percebe que 1 é o *leader* (ao receber seu *AppendEntry* com mesmo *term*)
- Server 2, 3, 4 e 5 continuam como *follower* indefinidamente, pois 1 é *leader*

```
C:\golangJuliana\src\labRaft\run>run -id 1
2019/08/29 14:48:28 [SERVER] Listening on 'localhost:3001'
2019/08/29 14:48:28 [FOLLOWER] Run Logic.
2019/08/29 14:48:28 [SERVER] Accepting connections.
2019/08/29 14:48:31 [FOLLOWER] Election timeout.
2019/08/29 14:48:31 [CANDIDATE] Run Logic.
2019/08/29 14:48:31 [CANDIDATE] Running for term '1'.
2019/08/29 14:48:31 [CANDIDATE] Vote denied to 'localhost:3002' for term '1'.
2019/08/29 14:48:31 [CANDIDATE] Vote denied by 'localhost:3002'.
2019/08/29 14:48:31 [CANDIDATE] Vote denied by 'localhost:3003'.
2019/08/29 14:48:31 [CANDIDATE] Vote granted by 'localhost:3005'.
2019/08/29 14:48:31 [CANDIDATE] VoteCount: 2
2019/08/29 14:48:31 [CANDIDATE] Vote granted by 'localhost:3004'.
2019/08/29 14:48:31 [CANDIDATE] VoteCount: 3
2019/08/29 14:48:31 [CANDIDATE] Run Logic.
2019/08/29 14:48:31 [LEADER] Run Logic.

C:\golangJuliana\src\labRaft\run>run -id 2
2019/08/29 14:48:28 [SERVER] Listening on 'localhost:3002'
2019/08/29 14:48:28 [SERVER] Accepting connections.
2019/08/29 14:48:28 [FOLLOWER] Run Logic.
2019/08/29 14:48:31 [FOLLOWER] Election timeout.
2019/08/29 14:48:31 [CANDIDATE] Run Logic.
2019/08/29 14:48:31 [CANDIDATE] Running for term '1'.
2019/08/29 14:48:31 [CANDIDATE] Vote denied to 'localhost:3001' for term '1'.
2019/08/29 14:48:31 [CANDIDATE] Vote denied by 'localhost:3001'.
2019/08/29 14:48:31 [CANDIDATE] Vote granted by 'localhost:3003'.
2019/08/29 14:48:31 [CANDIDATE] VoteCount: 2
2019/08/29 14:48:31 [CANDIDATE] Vote denied by 'localhost:3005'.
2019/08/29 14:48:31 [CANDIDATE] Vote denied by 'localhost:3004'.
2019/08/29 14:48:31 [CANDIDATE] Stepping down.
2019/08/29 14:48:31 [FOLLOWER] Run Logic.
2019/08/29 14:48:31 [FOLLOWER] Accept AppendEntry from 'localhost:3001'
2019/08/29 14:48:31 [FOLLOWER] Accept AppendEntry from 'localhost:3001'

C:\golangJuliana\src\labRaft\run>run -id 3
2019/08/29 14:48:28 [SERVER] Listening on 'localhost:3003'
2019/08/29 14:48:28 [FOLLOWER] Run Logic.
2019/08/29 14:48:28 [SERVER] Accepting connections.
2019/08/29 14:48:31 [FOLLOWER] Update old term '0' to new term '1' from 'localhost:3002'.
2019/08/29 14:48:31 [FOLLOWER] Vote granted to 'localhost:3002' for term '1'.
2019/08/29 14:48:31 [FOLLOWER] Vote denied to 'localhost:3001' for term '1'.
2019/08/29 14:48:31 [FOLLOWER] Accept AppendEntry from 'localhost:3001'
2019/08/29 14:48:31 [FOLLOWER] Accept AppendEntry from 'localhost:3001'

C:\golangJuliana\src\labRaft\run>run -id 4
2019/08/29 14:48:29 [SERVER] Listening on 'localhost:3004'
2019/08/29 14:48:29 [FOLLOWER] Run Logic.
2019/08/29 14:48:29 [SERVER] Accepting connections.
2019/08/29 14:48:31 [FOLLOWER] Update old term '0' to new term '1' from 'localhost:3001'.
2019/08/29 14:48:31 [FOLLOWER] Vote granted to 'localhost:3001' for term '1'.
2019/08/29 14:48:31 [FOLLOWER] Vote denied to 'localhost:3002' for term '1'.
2019/08/29 14:48:31 [FOLLOWER] Accept AppendEntry from 'localhost:3001'

C:\golangJuliana\src\labRaft\run>run -id 5
2019/08/29 14:48:29 [SERVER] Listening on 'localhost:3005'
2019/08/29 14:48:29 [FOLLOWER] Run Logic.
2019/08/29 14:48:29 [SERVER] Accepting connections.
2019/08/29 14:48:31 [FOLLOWER] Update old term '0' to new term '1' from 'localhost:3001'.
2019/08/29 14:48:31 [FOLLOWER] Vote granted to 'localhost:3001' for term '1'.
2019/08/29 14:48:31 [FOLLOWER] Vote denied to 'localhost:3002' for term '1'.
2019/08/29 14:48:31 [FOLLOWER] Accept AppendEntry from 'localhost:3001'
```


Exemplo similar ao anterior, considere que 1 é *leader* do *term* 1, mas 1 falha:

- Server 2 e 5 viram *candidate* no *term* 2
- Server 2 vira *leader* (com votos de 3, 4 e o seu próprio)
- Server 5 faz *Step Down* (aceitando 2 como *leader*)
- Server 3, 4 e 5 continuam como *follower* indefinidamente, pois 2 é *leader*

```

C:\golangJuliana\src\labRaft\run>
2019/08/29 15:20:53 [LEADER] Run Logic.

C:\golangJuliana\src\labRaft\run>

2019/08/29 15:20:57 [FOLLOWER] Accept AppendEntry from 'localhost:3001'
2019/08/29 15:20:58 [FOLLOWER] Update old term '1' to new term '2' from 'localhost:3002'
2019/08/29 15:20:58 [FOLLOWER] Vote granted to 'localhost:3002' for term '2'
2019/08/29 15:20:58 [FOLLOWER] Vote denied to 'localhost:3005' for term '2'
2019/08/29 15:20:59 [FOLLOWER] Accept AppendEntry from 'localhost:3002'
2019/08/29 15:20:59 [FOLLOWER] Accept AppendEntry from 'localhost:3002'

2019/08/29 15:20:57 [FOLLOWER] Accept AppendEntry from 'localhost:3001'
2019/08/29 15:20:57 [FOLLOWER] Accept AppendEntry from 'localhost:3001'
2019/08/29 15:20:57 [FOLLOWER] Accept AppendEntry from 'localhost:3001'
2019/08/29 15:20:58 [FOLLOWER] Election timeout.
2019/08/29 15:20:58 [CANDIDATE] Run Logic.
2019/08/29 15:20:58 [CANDIDATE] Running for term '2'.
2019/08/29 15:20:58 [CANDIDATE] Vote denied by 'localhost:3005'.
2019/08/29 15:20:58 [CANDIDATE] Vote granted by 'localhost:3003'.
2019/08/29 15:20:58 [CANDIDATE] VoteCount: 2
2019/08/29 15:20:58 [CANDIDATE] Vote granted by 'localhost:3004'.
2019/08/29 15:20:58 [CANDIDATE] VoteCount: 3
2019/08/29 15:20:58 [LEADER] Run Logic.
2019/08/29 15:20:58 [LEADER] Vote denied to 'localhost:3005' for term '2'.

C:\golangJuliana\src\labRaft\run>

2019/08/29 15:20:57 [FOLLOWER] Accept AppendEntry from 'localhost:3001'
2019/08/29 15:20:58 [FOLLOWER] Election timeout.
2019/08/29 15:20:58 [CANDIDATE] Run Logic.
2019/08/29 15:20:58 [CANDIDATE] Running for term '2'.
2019/08/29 15:20:58 [CANDIDATE] Vote denied to 'localhost:3002' for term '2'.
2019/08/29 15:20:58 [CANDIDATE] Vote denied by 'localhost:3003'.
2019/08/29 15:20:58 [CANDIDATE] Vote denied by 'localhost:3002'.
2019/08/29 15:20:58 [CANDIDATE] Stepping down.
2019/08/29 15:20:59 [FOLLOWER] Run Logic.
2019/08/29 15:20:59 [FOLLOWER] Accept AppendEntry from 'localhost:3002'

```

- Se Server 1 retomar, ele aceita 2 como *leader*, veja:

```

C:\golangJuliana\src\labRaft\run>run -id 1
2019/08/29 15:34:02 [SERVER] Listening on 'localhost:3001'
2019/08/29 15:34:02 [FOLLOWER] Run Logic.
2019/08/29 15:34:02 [SERVER] Accepting connections.
2019/08/29 15:34:02 [FOLLOWER] Update old term '0' to new term '1' from 'localhost:3002'
2019/08/29 15:34:02 [FOLLOWER] Accept AppendEntry from 'localhost:3002'
2019/08/29 15:34:02 [FOLLOWER] Accept AppendEntry from 'localhost:3002'
2019/08/29 15:34:02 [FOLLOWER] Accept AppendEntry from 'localhost:3002'
2019/08/29 15:34:02 [FOLLOWER] Accept AppendEntry from 'localhost:3002'
2019/08/29 15:34:02 [FOLLOWER] Accept AppendEntry from 'localhost:3002'
2019/08/29 15:34:02 [FOLLOWER] Accept AppendEntry from 'localhost:3002'
2019/08/29 15:34:02 [FOLLOWER] Accept AppendEntry from 'localhost:3002'

```

Tarefa 2. Simule um caso similar ao apresentado com: falha do líder (daí outro assume a liderança) e retomada do servidor falho (para ele reconhecer a liderança do outro).

Tarefa 3. Simule um caso com mais falhas. Explique os detalhes. Em seguida, mostre o caso com 3 ou mais falhas, onde o sistema não consegue mais decidir por um novo líder.

Bom trabalho!