# SHOPPING LISTS NA CLOUD 

SDLE 2023/2024

Grupo G77
Bruna Marques - up202007191
Hugo Gomes - up202004343
João Moreira - up202005035
Lia Vieira - up202005042

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

# Descrição do problema

Este projeto tem como objetivo desenvolver uma aplicação de lista de compras com uma abordagem local-first que permita a persistência local de dados, juntamente com um componente na nuvem para partilha e backup. Os utilizadores devem poder criar listas distintas identificadas por IDs únicos e quem conhecer esse ID poder adicionar e excluir itens da lista. Para suportar a colaboração simultânea, propõe-se a implementação de *Conflict-free Replicated Data Types* (CRDTs). Além disso, para acomodar milhões de utilizadores, a arquitetura da nuvem deve ser cuidadosamente desenvolvida, inspirando-se em princípios semelhantes aos do *Amazon Dynamo*.

# SOLUÇÃO TÉCNICA

# CRDTs

- O **ORMap** consiste numa CRDT que mapeia strings para **CCounters**. Assim, é possível associar uma chave de um produto de uma Shopping List a um contador (CCounter) que pode ser incrementado ou decrementado.
- Quando um nó deseja atualizar o contador associado a um produto específico, ele atualiza o CCounter localmente.
- Por cada incremento/decremento, consulta-se a sua última contagem, altera-se e armazena-se sob um novo **dot** após excluir o seu **dot** anterior.
- Um **dot** corresponde a um par (replica-id, sequence-id) que representa o timestamp de uma operação específica feita naquela réplica.
- As propriedades CRDT do ORMap garantem que essas atualizações possam ser propagadas para outros nós sem conflitos e a junção de atualizações de nós diferentes garante que o estado final é consistente.

# Cliente

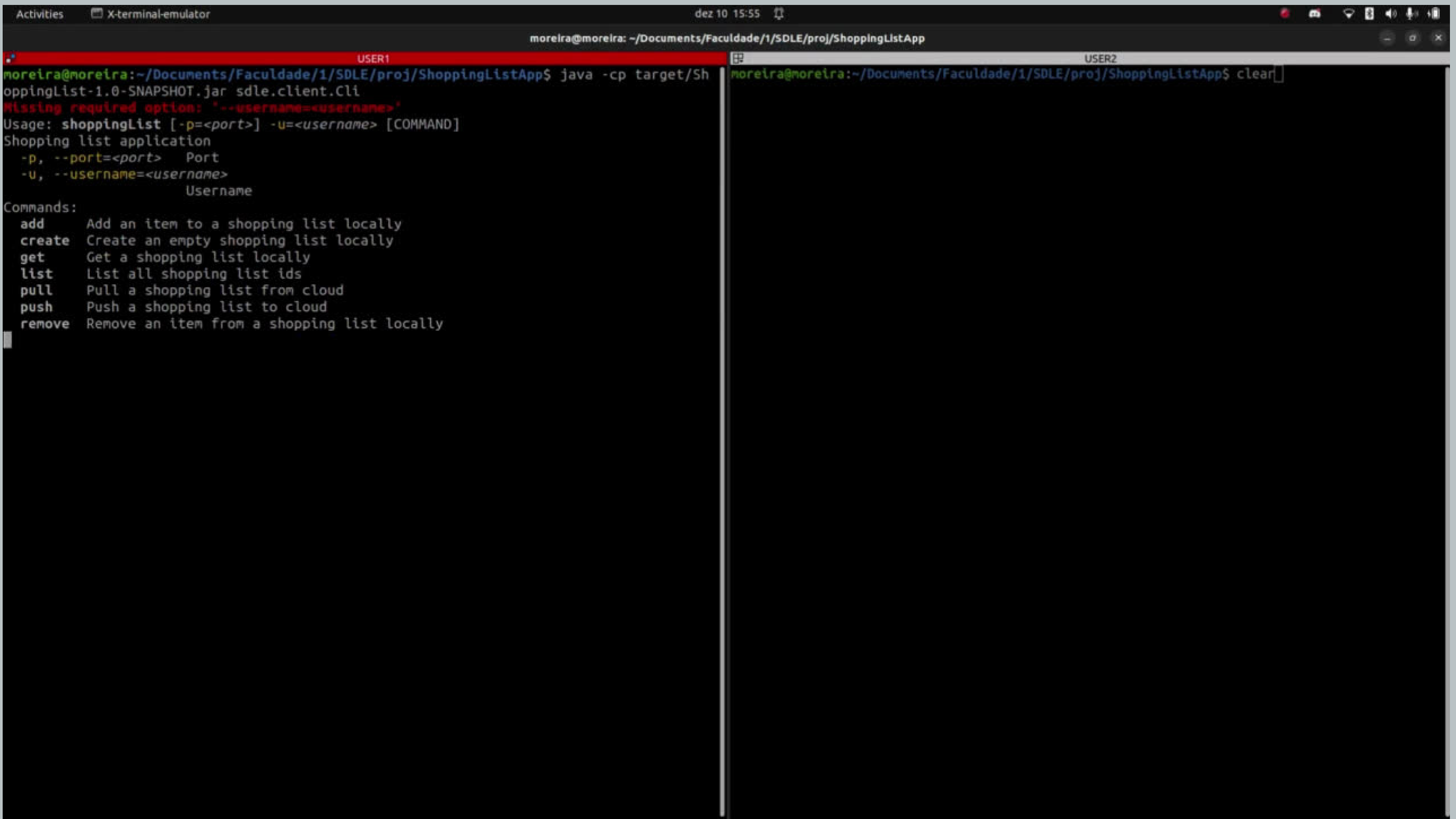
Os utilizadores podem interagir com a aplicação através de comandos específicos.
Localmente, os utilizadores podem visualizar e criar listas, adicionar e eliminar itens nessas listas.

> Get {Shopping List Id}
>
> Create {Shopping List Id}
>
> Add {Shopping List id} {Item Name} {Item Quantity}
>
> Remove {Shopping List id} {Item Name} {Item Quantity}

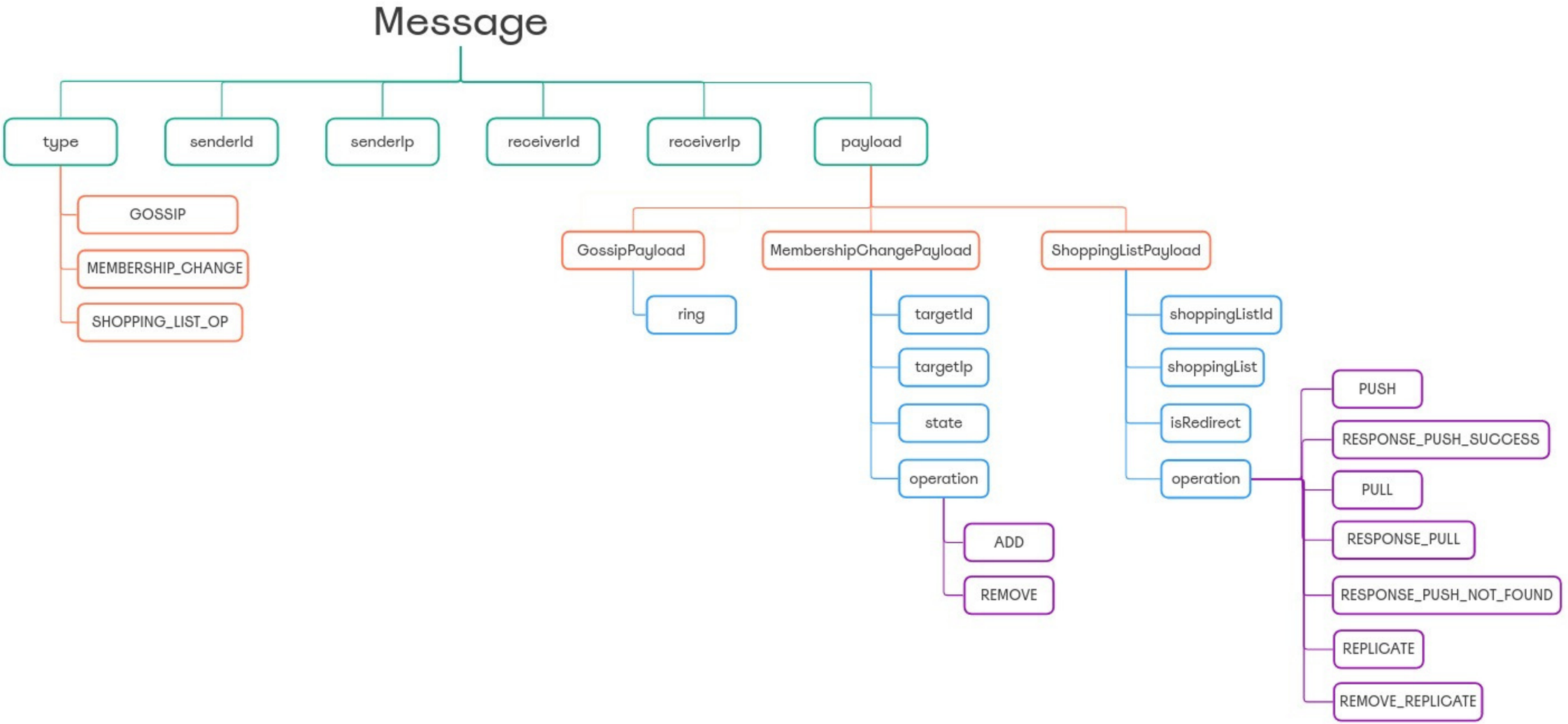
Através do comando *push*, as alterações na lista são enviadas para a nuvem, assegurando a sincronização em vários dispositivos. Com o comando *pull*, os utilizadores obtêm a versão mais recente da lista na nuvem para o seu dispositivo local.

> Pull {Shopping List Id}
>
> Push {Shopping List Id}

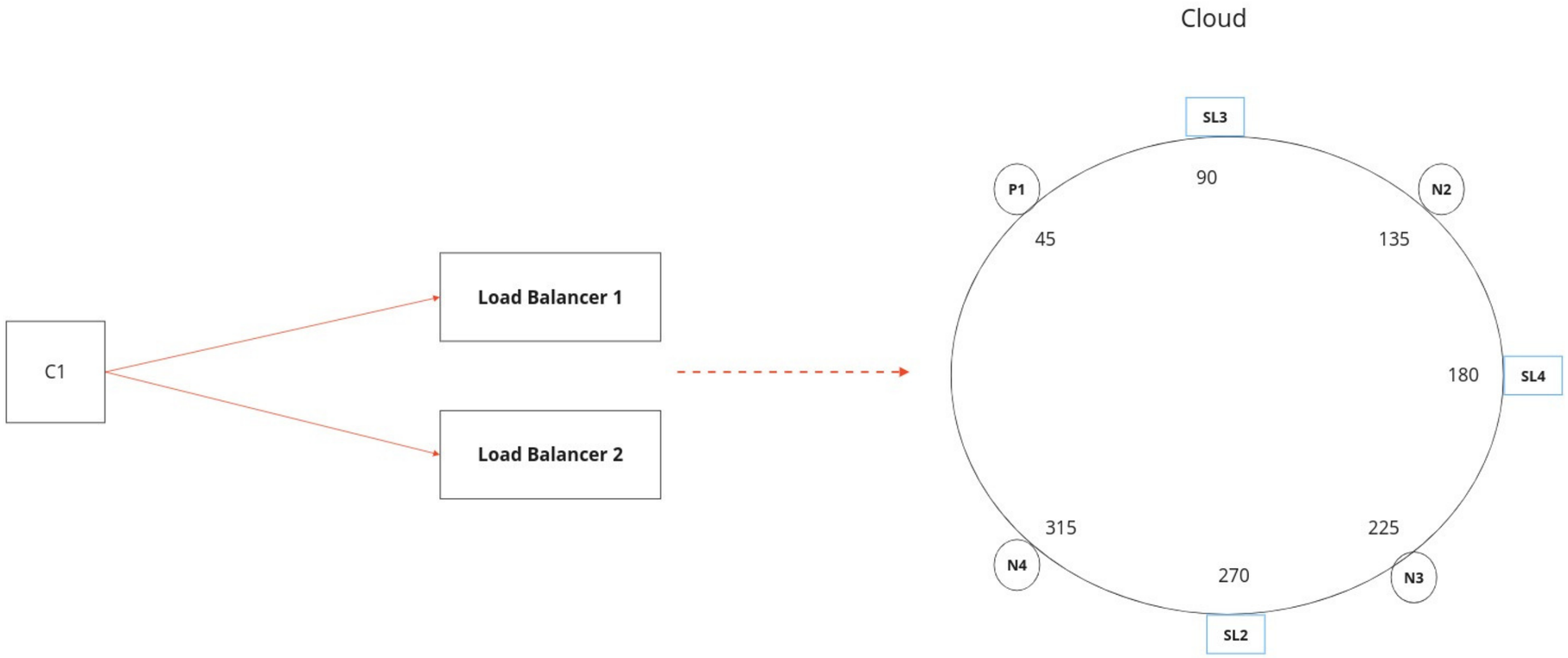
Cada cliente tem a sua DB local onde são armazenadas as suas versões das listas (guardadas em BLOB através da serialização da CRDT).
A demo seguinte mostra as operações de *create*, *get*, *add* e *remove*.


Activities
X-terminal-emulator
dez 10 15:55
moreira@moreira: ~/Documents/Faculdade/1/SDLE/proj/ShoppingListApp
USER1
USER2
moreira@moreira:~/Documents/Faculdade/1/SDLE/proj/ShoppingListApp$ java -cp target/Sh
oppingList-1.0-SNAPSHOT.jar sdle.client.Cli
Missing required option: '--username=<username>'
Usage: shoppingList [-p=<port>] -u=<username> [COMMAND]
Shopping list application
-p, --port=<port> Port
-u, --username=<username>
Username
Commands:
add Add an item to a shopping list locally
create Create an empty shopping list locally
get Get a shopping list locally
list List all shopping list ids
pull Pull a shopping list from cloud
push Push a shopping list to cloud
remove Remove an item from a shopping list locally
moreira@moreira:~/Documents/Faculdade/1/SDLE/proj/ShoppingListApp$ clear

# Mensagem


Message
type
senderId
senderIp
receiverId
receiverIp
payload
GOSSIP
MEMBERSHIP_CHANGE
SHOPPING_LIST_OP
GossipPayload
ring
MembershipChangePayload
targetId
targetIp
state
operation
ADD
REMOVE
ShoppingListPayload
shoppingListId
shoppingList
isRedirect
operation
PUSH
RESPONSE_PUSH_SUCCESS
PULL
RESPONSE_PULL
RESPONSE_PUSH_NOT_FOUND
REPLICATE
REMOVE_REPLICATE

# Arquitetura




**Load Balancer**: Foi implementado um Load Balancer simplificado que, ao receber uma mensagem por parte do cliente, redireciona-a para um nó aleatório da cloud.

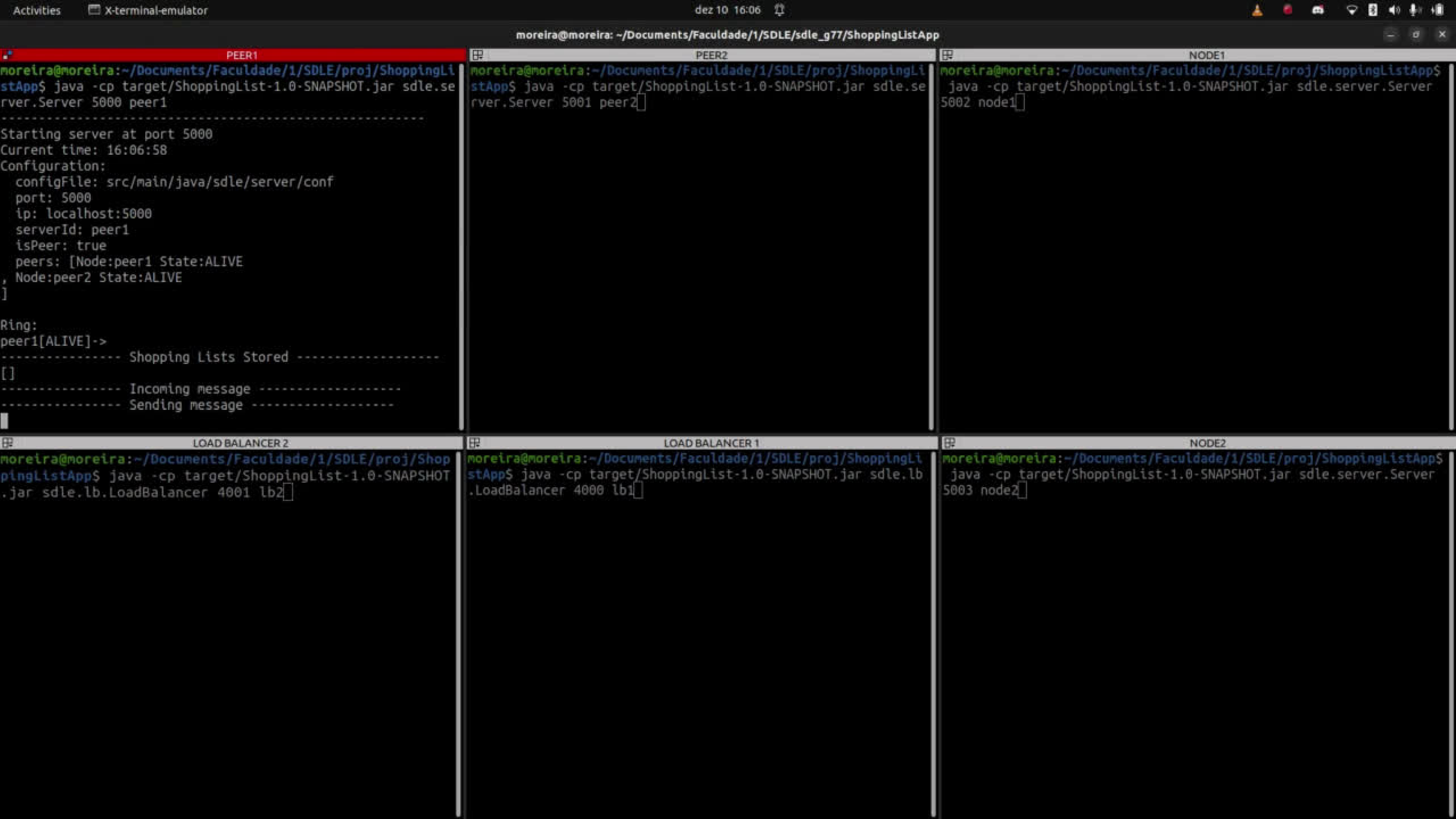
# Cloud

- A cloud foi implementada utilizando o conceito de *consistent hashing*. A cada nó é atribuido uma posição no anel de acordo com o MD5 hash do id.
- Cada nó mantém o estado atual do anel atavés de um SortedMap<Long, NodeState>.
- NodeState mantém o estado do nó(ALIVE, DOWN) e a timestamp da ultima atualização.
- Periodicamente (20 em 20 segundos) cada nó começa um protocolo de Gossip em que escolhe 2 nós aleatórios e envia o seu estado do anel. O nó recetor junta os dois estados e atualiza o seu estado. Eventualmente todos os nós irão atingir o mesmo estado.
- A adição de nós ao anel é feita através de Peers. Peers são nós que têm um comportamento igual aos outros, no entanto, são conhecidos por todos os nós através de uma configuração estática.

# Cloud Setup

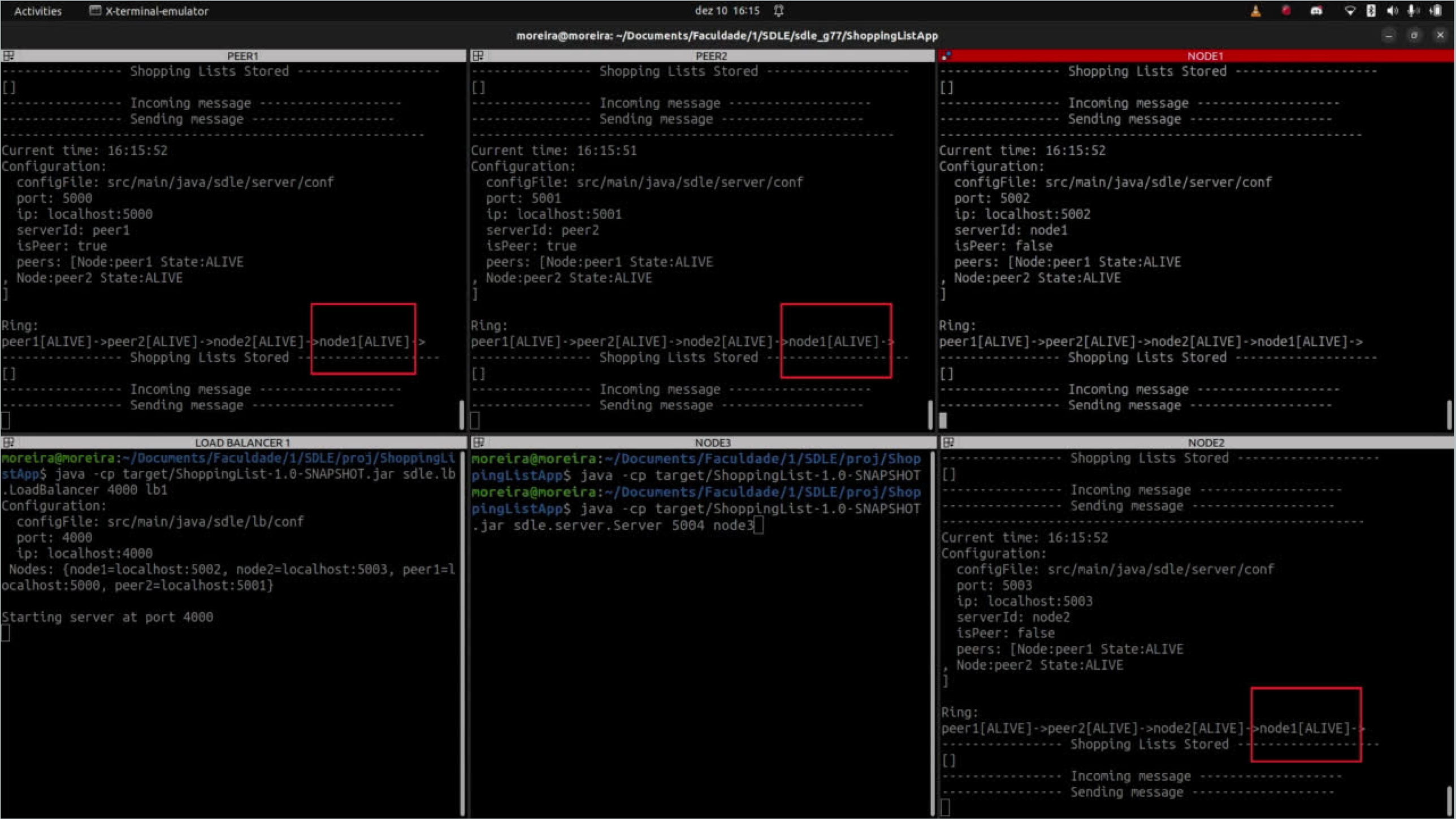
A demo seguinte mostra o setup inicial da cloud com os seguintes componentes:

- peer1 localhost:5000
- peer2 localhost:5001
- node1 localhost:5002
- node2 localhost:5003
- lb1  localhost:4000
- lb2  localhost:4001


Activities
X-terminal-emulator
dez 10 16:06
moreira@moreira: ~/Documents/Faculdade/1/SDLE/sdle_g77/ShoppingListApp
PEER1
moreira@moreira:~/Documents/Faculdade/1/SDLE/proj/ShoppingListApp$ java -cp target/ShoppingList-1.0-SNAPSHOT.jar sdle.server.Server 5000 peer1
Starting server at port 5000
Current time: 16:06:58
Configuration:
configFile: src/main/java/sdle/server/conf
port: 5000
ip: localhost:5000
serverId: peer1
isPeer: true
peers: [Node:peer1 State:ALIVE
, Node:peer2 State:ALIVE
]
Ring:
peer1[ALIVE]->
Shopping Lists Stored
[]
Incoming message
Sending message
PEER2
moreira@moreira:~/Documents/Faculdade/1/SDLE/proj/ShoppingListApp$ java -cp target/ShoppingList-1.0-SNAPSHOT.jar sdle.server.Server 5001 peer2
NODE1
moreira@moreira:~/Documents/Faculdade/1/SDLE/proj/ShoppingListApp$ java -cp target/ShoppingList-1.0-SNAPSHOT.jar sdle.server.Server 5002 node1
LOAD BALANCER 2
moreira@moreira:~/Documents/Faculdade/1/SDLE/proj/ShoppingListApp$ java -cp target/ShoppingList-1.0-SNAPSHOT.jar sdle.lb.LoadBalancer 4001 lb2
LOAD BALANCER 1
moreira@moreira:~/Documents/Faculdade/1/SDLE/proj/ShoppingListApp$ java -cp target/ShoppingList-1.0-SNAPSHOT.jar sdle.lb.LoadBalancer 4000 lb1
NODE2
moreira@moreira:~/Documents/Faculdade/1/SDLE/proj/ShoppingListApp$ java -cp target/ShoppingList-1.0-SNAPSHOT.jar sdle.server.Server 5003 node2

# Cloud Adição e Remoção de nós

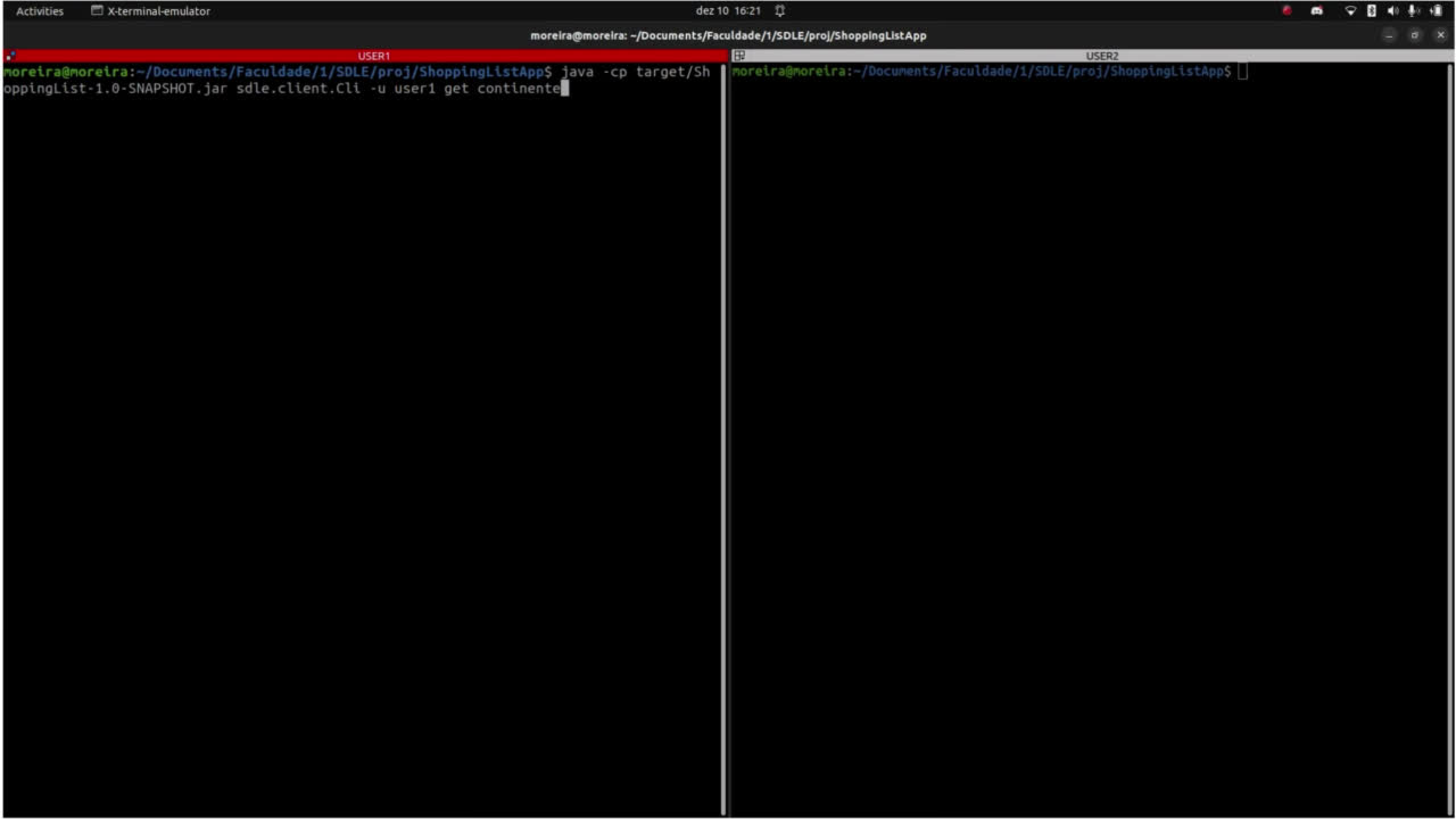
- A demo seguinte mostra a remoção do nó NODE1 e a adição do nó NODE3
- Através do protocolo de Gossip todos os nós convergem para o mesmo estado:
  - PEER1[ALIVE] -> PEER2[ALIVE]->NODE3[ALIVE]->NODE1[DOWN]->NODE2[ALIVE]


Activities
X-terminal-emulator
dez 10 16:15
moreira@moreira: ~/Documents/Faculdade/1/SDLE/sdle_g77/ShoppingListApp
PEER1
---------- Shopping Lists Stored ----------
[]
---------- Incoming message ----------
---------- Sending message ----------
Current time: 16:15:52
Configuration:
configFile: src/main/java/sdle/server/conf
port: 5000
ip: localhost:5000
serverId: peer1
isPeer: true
peers: [Node:peer1 State:ALIVE
, Node:peer2 State:ALIVE
]
Ring:
peer1[ALIVE]->peer2[ALIVE]->node2[ALIVE]->node1[ALIVE]->
---------- Shopping Lists Stored ----------
[]
---------- Incoming message ----------
---------- Sending message ----------
PEER2
---------- Shopping Lists Stored ----------
[]
---------- Incoming message ----------
---------- Sending message ----------
Current time: 16:15:51
Configuration:
configFile: src/main/java/sdle/server/conf
port: 5001
ip: localhost:5001
serverId: peer2
isPeer: true
peers: [Node:peer1 State:ALIVE
, Node:peer2 State:ALIVE
]
Ring:
peer1[ALIVE]->peer2[ALIVE]->node2[ALIVE]->node1[ALIVE]->
---------- Shopping Lists Stored ----------
[]
---------- Incoming message ----------
---------- Sending message ----------
NODE1
---------- Shopping Lists Stored ----------
[]
---------- Incoming message ----------
---------- Sending message ----------
Current time: 16:15:52
Configuration:
configFile: src/main/java/sdle/server/conf
port: 5002
ip: localhost:5002
serverId: node1
isPeer: false
peers: [Node:peer1 State:ALIVE
, Node:peer2 State:ALIVE
]
Ring:
peer1[ALIVE]->peer2[ALIVE]->node2[ALIVE]->node1[ALIVE]->
---------- Shopping Lists Stored ----------
[]
---------- Incoming message ----------
---------- Sending message ----------
LOAD BALANCER 1
moreira@moreira:~/Documents/Faculdade/1/SDLE/proj/ShoppingListApp$ java -cp target/ShoppingList-1.0-SNAPSHOT.jar sdle.lb.LoadBalancer 4000 lb1
Configuration:
configFile: src/main/java/sdle/lb/conf
port: 4000
ip: localhost:4000
Nodes: {node1=localhost:5002, node2=localhost:5003, peer1=localhost:5000, peer2=localhost:5001}
Starting server at port 4000
NODE3
moreira@moreira:~/Documents/Faculdade/1/SDLE/proj/ShoppingListApp$ java -cp target/ShoppingList-1.0-SNAPSHOT
moreira@moreira:~/Documents/Faculdade/1/SDLE/proj/ShoppingListApp$ java -cp target/ShoppingList-1.0-SNAPSHOT.jar sdle.server.Server 5004 node3
NODE2
---------- Shopping Lists Stored ----------
[]
---------- Incoming message ----------
---------- Sending message ----------
Current time: 16:15:52
Configuration:
configFile: src/main/java/sdle/server/conf
port: 5003
ip: localhost:5003
serverId: node2
isPeer: false
peers: [Node:peer1 State:ALIVE
, Node:peer2 State:ALIVE
]
Ring:
peer1[ALIVE]->peer2[ALIVE]->node2[ALIVE]->node1[ALIVE]->
---------- Shopping Lists Stored ----------
[]
---------- Incoming message ----------
---------- Sending message ----------

# Demo: Colaboração entre utilizadores

Cenário:

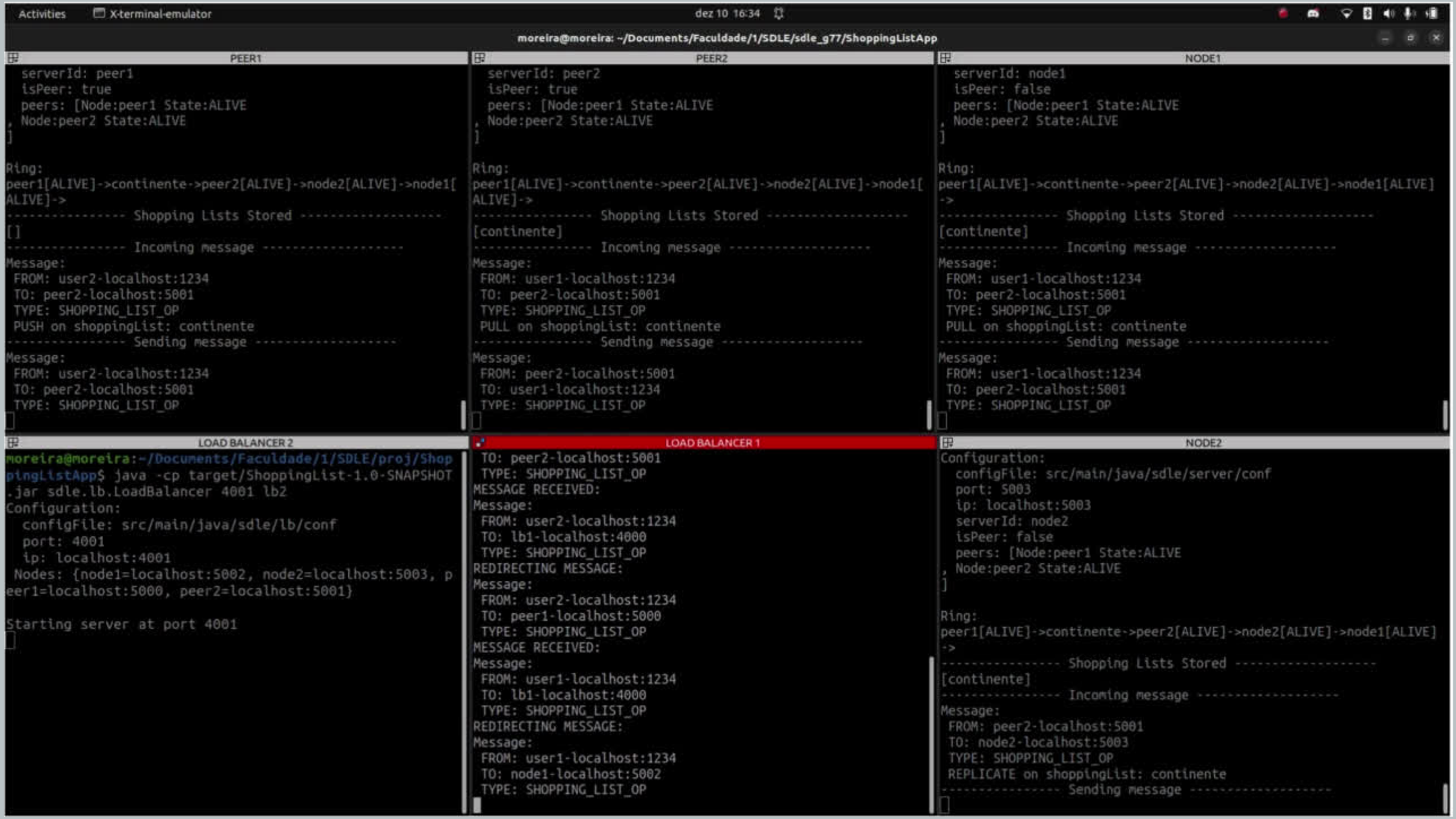
- User1 tem uma lista com id ‘continente’
- User1 dá push da lista ‘continente’ para cloud
- User2 dá pull da lista ‘continente’ da cloud
- User2 remove quantidade 2 do item ‘tomate’ da lista ‘continente’
- User2 dá push da lista ‘continente’ para cloud
- User1 dá pull da lista ‘continente’ da cloud


Activities
X-terminal-emulator
dez 10 16:21
moreira@moreira: ~/Documents/Faculdade/1/SDLE/proj/ShoppingListApp
USER1
moreira@moreira:~/Documents/Faculdade/1/SDLE/proj/ShoppingListApp$ java -cp target/ShoppingList-1.0-SNAPSHOT.jar sdle.client.Cli -u user1 get continente
USER2
moreira@moreira:~/Documents/Faculdade/1/SDLE/proj/ShoppingListApp$

# Demo: Falha Load Balancer 1

Cenário:

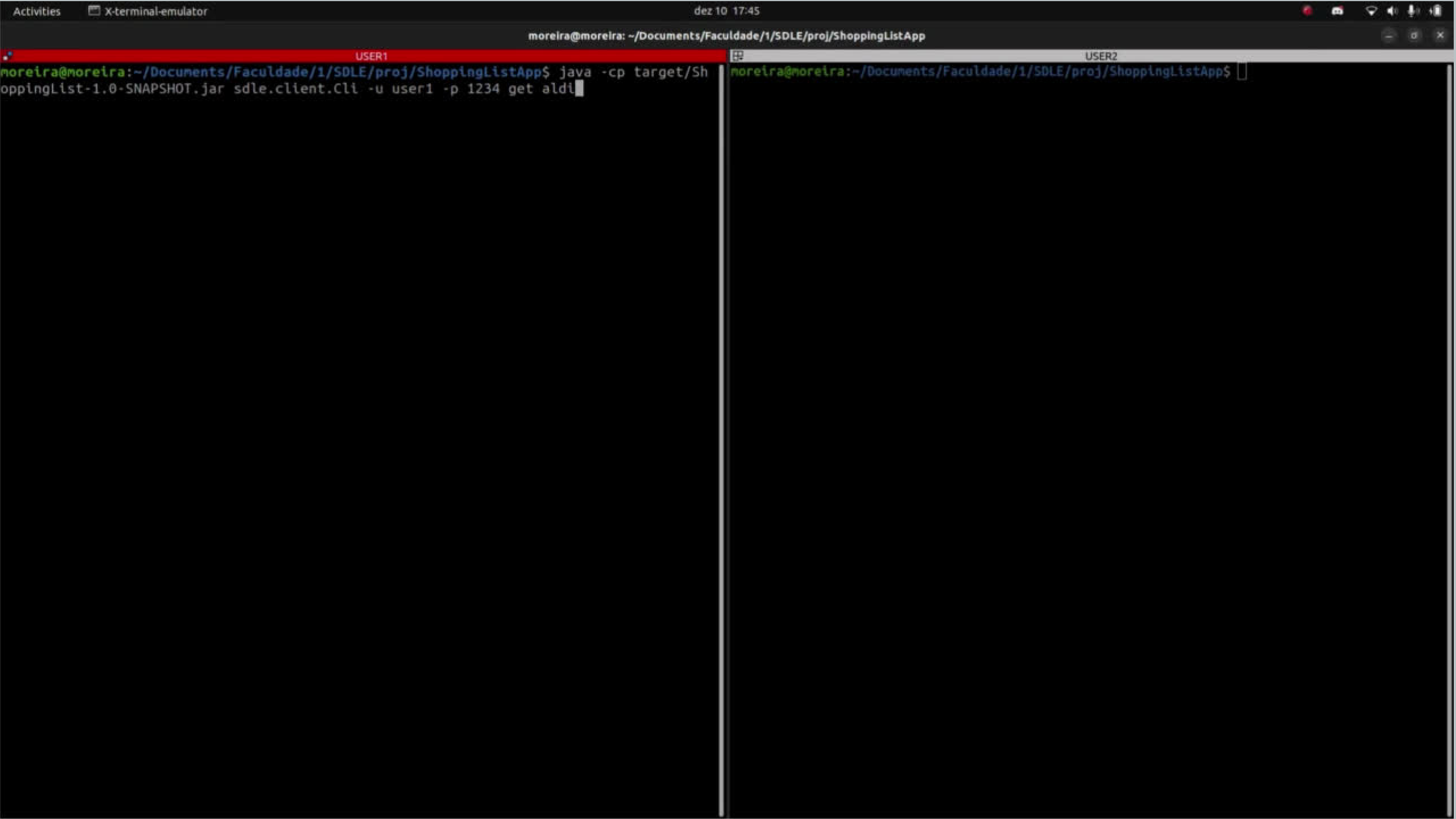
- Load Balancer 1 falha
- User1 adiciona 2 unidades ao item ‘laranja’ da lista ’continente’
- User1 dá push da lista ‘continente’ - Envio é feito pelo Load Balancer 2
- User2 dá pull da lista ‘continente’


Activities
X-terminal-emulator
dez 10 16:34
moreira@moreira: ~/Documents/Faculdade/1/SDLE/sdle_g77/ShoppingListApp
PEER1
serverId: peer1
isPeer: true
peers: [Node:peer1 State:ALIVE
, Node:peer2 State:ALIVE
]
Ring:
peer1[ALIVE]->continente->peer2[ALIVE]->node2[ALIVE]->node1[
ALIVE]->
---------- Shopping Lists Stored ----------
[]
---------- Incoming message ----------
Message:
FROM: user2-localhost:1234
TO: peer2-localhost:5001
TYPE: SHOPPING_LIST_OP
PUSH on shoppingList: continente
---------- Sending message ----------
Message:
FROM: user2-localhost:1234
TO: peer2-localhost:5001
TYPE: SHOPPING_LIST_OP
PEER2
serverId: peer2
isPeer: true
peers: [Node:peer1 State:ALIVE
, Node:peer2 State:ALIVE
]
Ring:
peer1[ALIVE]->continente->peer2[ALIVE]->node2[ALIVE]->node1[
ALIVE]->
---------- Shopping Lists Stored ----------
[continente]
---------- Incoming message ----------
Message:
FROM: user1-localhost:1234
TO: peer2-localhost:5001
TYPE: SHOPPING_LIST_OP
PULL on shoppingList: continente
---------- Sending message ----------
Message:
FROM: peer2-localhost:5001
TO: user1-localhost:1234
TYPE: SHOPPING_LIST_OP
NODE1
serverId: node1
isPeer: false
peers: [Node:peer1 State:ALIVE
, Node:peer2 State:ALIVE
]
Ring:
peer1[ALIVE]->continente->peer2[ALIVE]->node2[ALIVE]->node1[ALIVE]
->
---------- Shopping Lists Stored ----------
[continente]
---------- Incoming message ----------
Message:
FROM: user1-localhost:1234
TO: peer2-localhost:5001
TYPE: SHOPPING_LIST_OP
PULL on shoppingList: continente
---------- Sending message ----------
Message:
FROM: user1-localhost:1234
TO: peer2-localhost:5001
TYPE: SHOPPING_LIST_OP
LOAD BALANCER 2
moreira@moreira:~/Documents/Faculdade/1/SDLE/proj/Shop
pingListApp$ java -cp target/ShoppingList-1.0-SNAPSHOT
.jar sdle.lb.LoadBalancer 4001 lb2
Configuration:
configFile: src/main/java/sdle/lb/conf
port: 4001
ip: localhost:4001
Nodes: {node1=localhost:5002, node2=localhost:5003, p
eer1=localhost:5000, peer2=localhost:5001}
Starting server at port 4001
LOAD BALANCER 1
TO: peer2-localhost:5001
TYPE: SHOPPING_LIST_OP
MESSAGE RECEIVED:
Message:
FROM: user2-localhost:1234
TO: lb1-localhost:4000
TYPE: SHOPPING_LIST_OP
REDIRECTING MESSAGE:
Message:
FROM: user2-localhost:1234
TO: peer1-localhost:5000
TYPE: SHOPPING_LIST_OP
MESSAGE RECEIVED:
Message:
FROM: user1-localhost:1234
TO: lb1-localhost:4000
TYPE: SHOPPING_LIST_OP
REDIRECTING MESSAGE:
Message:
FROM: user1-localhost:1234
TO: node1-localhost:5002
TYPE: SHOPPING_LIST_OP
NODE2
Configuration:
configFile: src/main/java/sdle/server/conf
port: 5003
ip: localhost:5003
serverId: node2
isPeer: false
peers: [Node:peer1 State:ALIVE
, Node:peer2 State:ALIVE
]
Ring:
peer1[ALIVE]->continente->peer2[ALIVE]->node2[ALIVE]->node1[ALIVE]
->
---------- Shopping Lists Stored ----------
[continente]
---------- Incoming message ----------
Message:
FROM: peer2-localhost:5001
TO: node2-localhost:5003
TYPE: SHOPPING_LIST_OP
REPLICATE on shoppingList: continente
---------- Sending message ----------

# Demo: Falha Coordenador

Cenário:

- User1 dá push da lista ‘aldi’
- NODE1 é coordenador da lista ‘aldi’
- NODE1 falha
- PEER1 é o novo coordenador
- PEER1 transfere a lista para o novo nó que pertence à preference list da lista ‘aldi’ (NODE2)
- User2 dá pull da lista ‘aldi’


Activities
X-terminal-emulator
dez 10 17:45
moreira@moreira: ~/Documents/Faculdade/1/SDLE/proj/ShoppingListApp
USER1
moreira@moreira:~/Documents/Faculdade/1/SDLE/proj/ShoppingListApp$ java -cp target/ShoppingList-1.0-SNAPSHOT.jar sdle.client.Cli -u user1 -p 1234 get aldi
USER2
moreira@moreira:~/Documents/Faculdade/1/SDLE/proj/ShoppingListApp$

# Limitações

- Apesar de totalmente funcional a nossa implementação apresenta limitações e tradeoffs:
  - O Load Balancer implementado revela ser muito simples, um trabalho futuro seria implementar um algoritmo mais aprimorado.
  - O facto de o load balancer redirecionar aleatoriamente implica um redirect por parte do nó recetor para o nó coordenador da lista, o que implica um aumento de latência. Outra solução seria tornar o load balancer mais 'inteligente' tendo este conhecimento do anel e sendo capaz de decidir qual o coordenador de uma lista. No entanto, consideramos que esta solução poderia originar um bottleneck visto que o load balancer teria que efetuar mais cálculos incluindo hashing.
  - Além disso, consideramos um trabalho futuro relevante implementar nós virtuais de modo a melhorar a redistribuição das listas no momento de adição e remoção de nós.

# Tecnologias

- Java – Linguagem principal
- Sqlite3 – Base de dados
- JDBC – Driver para Sqlite
- Java NIO – Comunicação
- Picocli – Command line Interface