

#Ping Me

A Fast and Elegant Messaging System

Project Group 11

Vaishali Koul (013738543)

CMPE-273 Enterprise Distributed Systems – Fall 2019

11.26.2019

TABLE OF CONTENTS

1	TABLE OF CONTENTS	2
2	EXECUTIVE SUMMARY	3
3	INTRODUCTION/BACKGROUND	4
4	OBJECTIVES	5
5	DESIGN	6
6	IMPLEMENTATION	10
7	RESULTS	22
8	CONCLUSION	31
9	RECOMMENDATION	32
10	REFERENCES	33
11	APPENDIX	34

EXECUTIVE SUMMARY

A fault tolerant Messaging System that utilizes Kafka (A message streaming platform) to queue messages, replacing the unreliable traditional message brokers. The Application allows a user to Sign up, Sign in, add channels (group like), post and send messages directly to other users and channels.

Reasons that make this Application Fault Tolerant -

1. Kafka queues the messages so even if the Consumer (Backend Spring Boot) server is down and as long as the producer (The client side react frontend) is up, messages can still be posted which will be saved to the database once the backend server comes up. The front end Application which acts as the Kafka producer has been replicated in sets of three served by an Elastic load balancer ensuring its high availability.
2. The database has been replicated in sets of three (The Application makes use of Cassandra, an AP System). Hence even if a database server is down, the system will still be able to store messages.
3. The Application has been deployed on cloud (AWS). Each server, the Backend, the front end servers as well as the Database instances have been replicated and are being served from behind with and ELB (Elastic Load Balancer) which handles load by forwarding requests fairly among the associated instances.
4. Since Cassandra sacrifices consistency for Availability, Eventual consistency is attained by the system once the network partition has been overcome.

BACKGROUND

This Application is motivated by one of the system outages that Slack (An already existing messaging application) went through.

A resource conflict (two or more jobs tried to acquire the same system resource) lead to a major slowing of the system execution. This lead to the Redis Memory exceeding the maximum limit. Since there was no free memory left, no jobs could be enqueued or dequeued either (because even dequeuing jobs requires some free memory). This lead to the failure of many queue dependent jobs and a major setback to the production. Slack had to invest a lot of resources to overcome the problem and get the system up and running again.

The older architecture that slack used makes use of traditional messaging brokers.

Reasons to their failure -

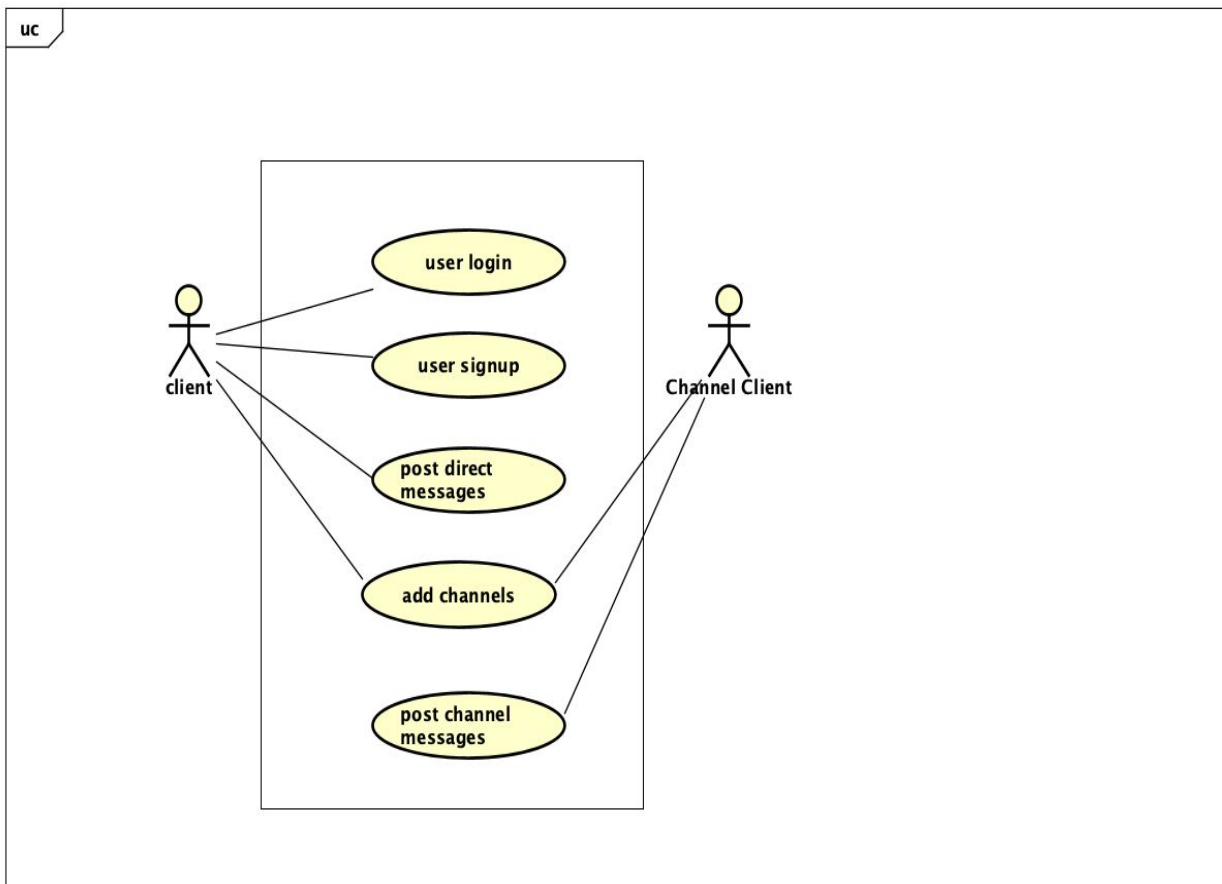
1. It assumes that the consumption from the queue will always be very fast. But in certain cases, the enqueueing may get faster which makes the cache run out of memory, blocking even the dequeuing of jobs.
2. As the queue size increases, it gets harder to dequeue or empty the queue when required.
3. The jobs were highly dependent on the database (Redis in Slack's case), adding new jobs/Queues would exhaust the database even more.

OBJECTIVES

The developed aims to achieve the following objectives:

- Development of a messaging application which allows users to post fast, durable messages to other users and groups.
- Integration of the developed messaging application with a Buffer space (Kafka message Queue) that increases the durability of the stored messages in cases of complete memory consumption and network partition.
- Integrating a durable messaging system which allows the separation of concerns between scaling and database (Database independent message queueing).
- Kafka ensures message delivery at least once (Delivery guarantee). Message offsets that help consumers (The Backend Spring Boot Application here) track the point till which messages have been consumed from the Kafka queue.
- Deployment of the messaging application with the Kafka broker to the cloud (AWS) where replicated instances are maintained which are served with Elastic Load Balancers to forward requests fairly.

DESIGN

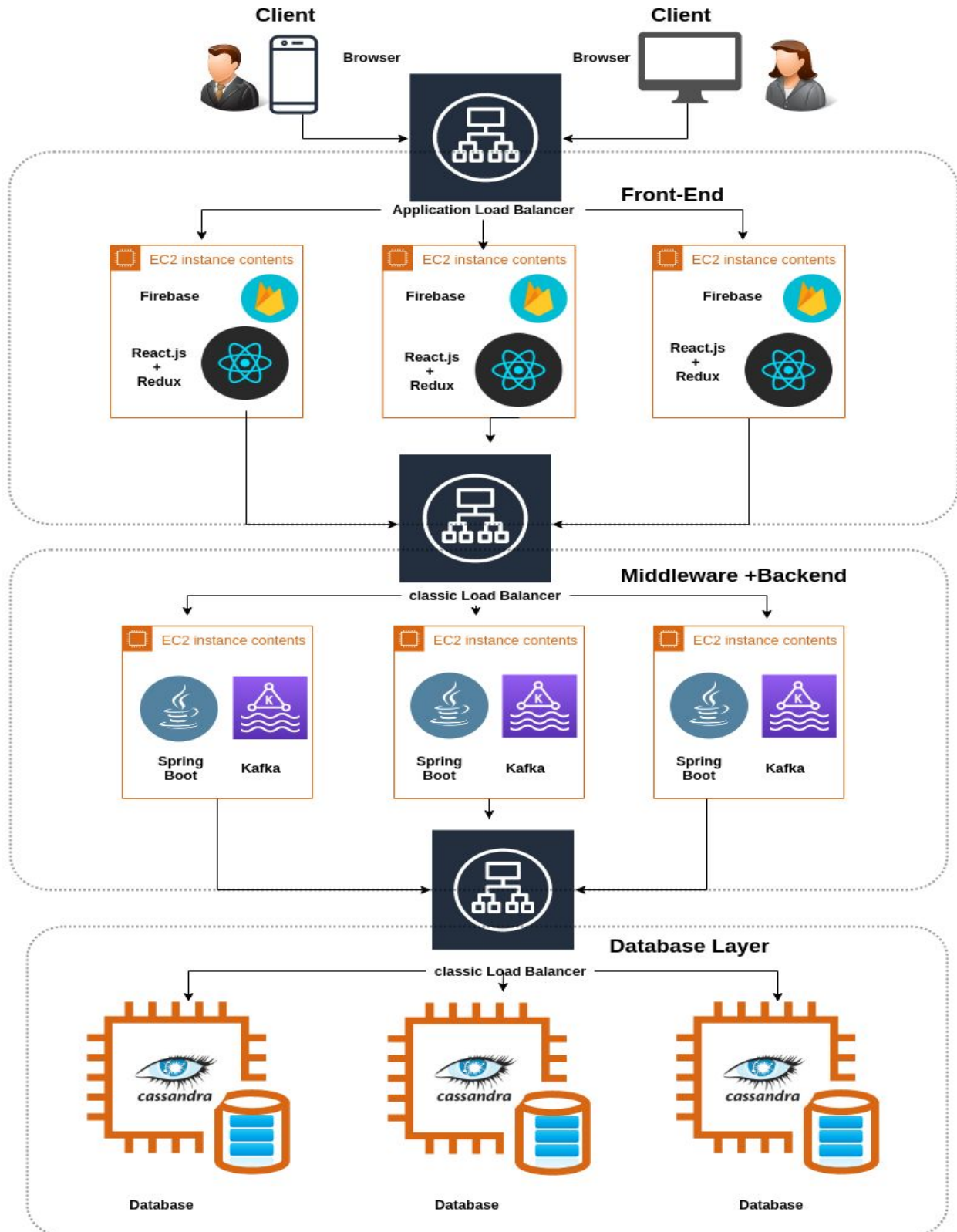


I. Use Case Diagram

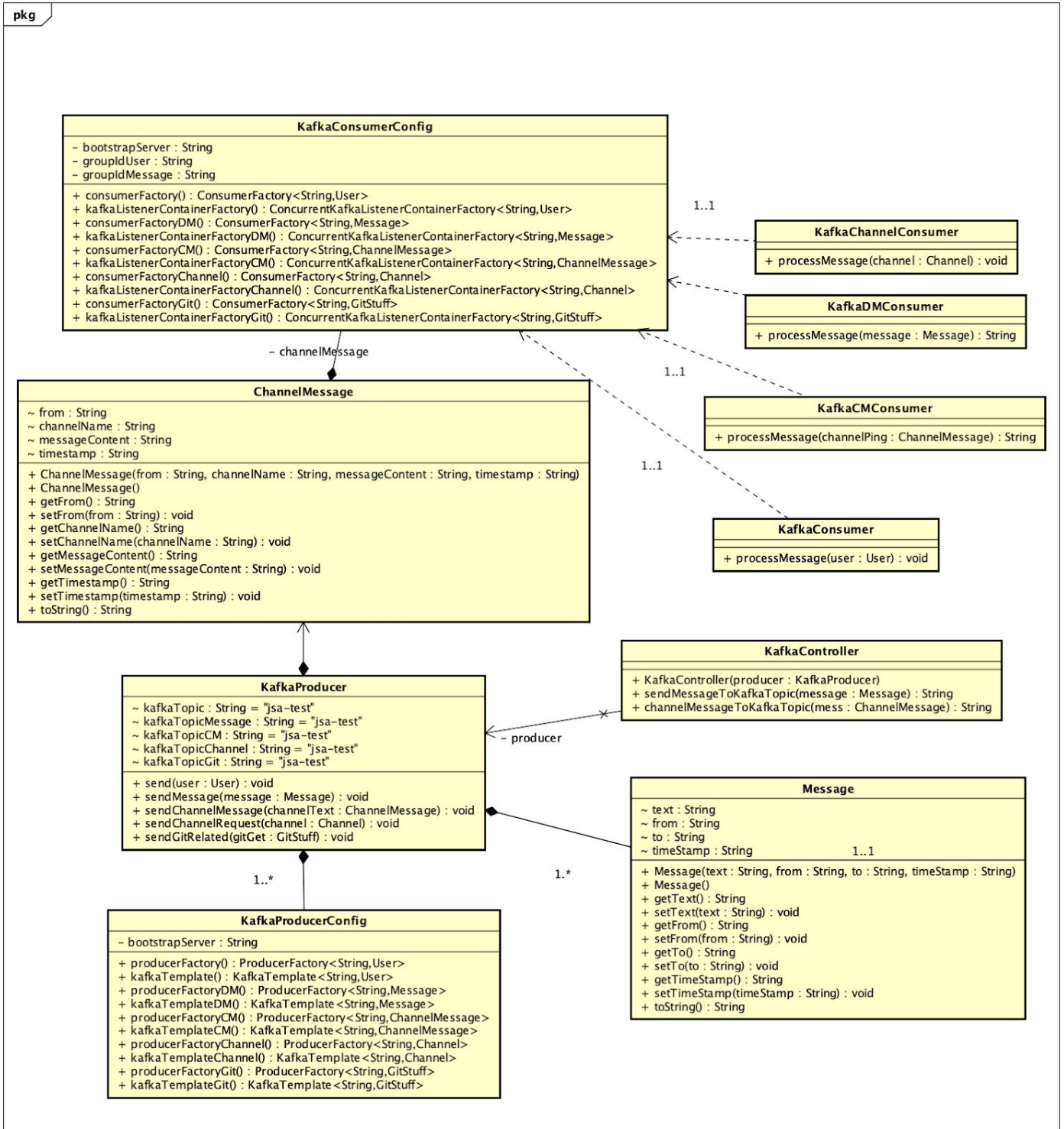
A user can perform the following functionalities -

1. Sign up to the messaging application
2. Sign in using their credentials
3. Post Direct Messages
4. Post channel Messages
5. Add new channels

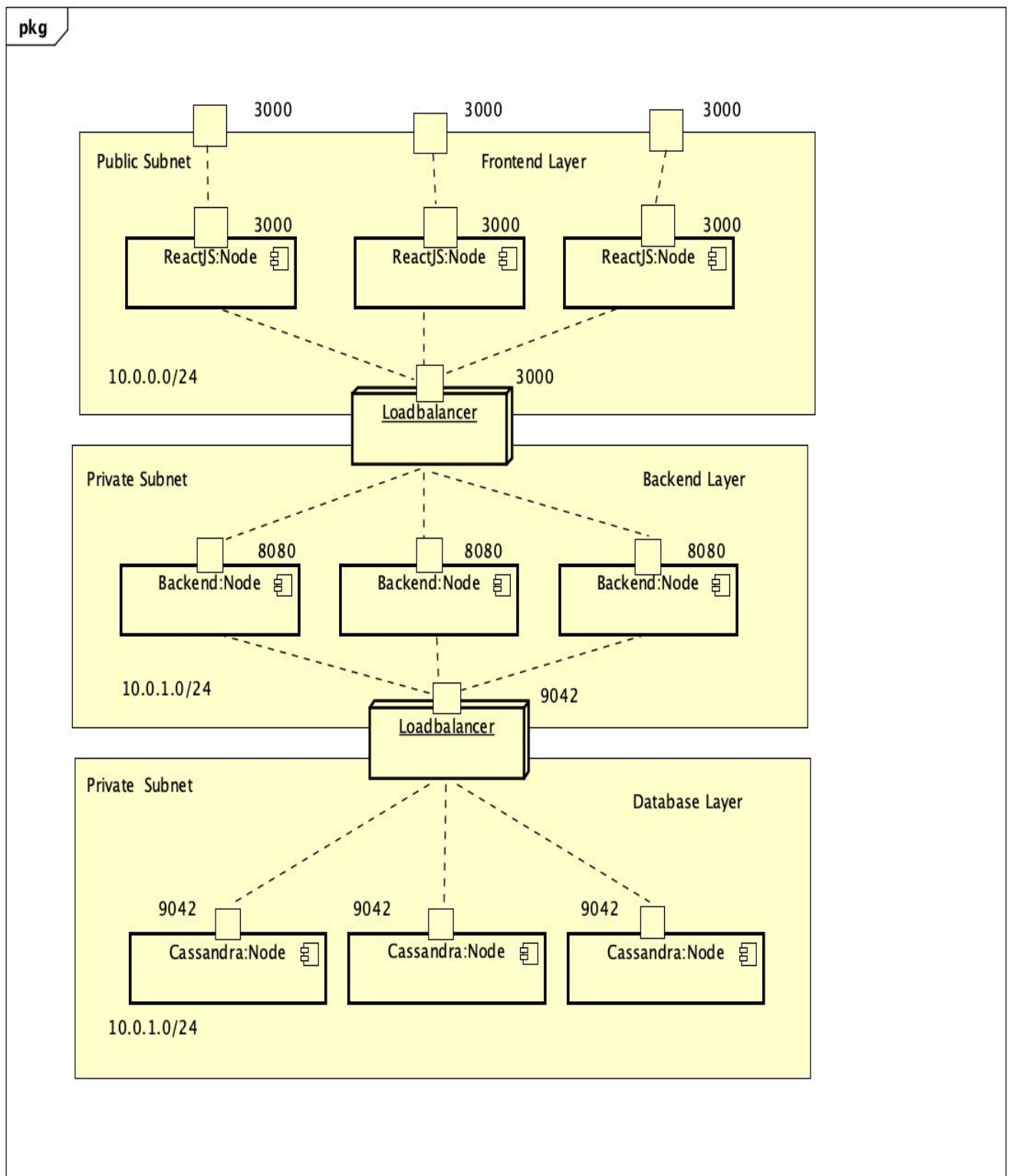
II. ARCHITECTURE



III. CLASS DIAGRAM



III. DEPLOYMENT DIAGRAM



IMPLEMENTATION

- **Technologies Used**

Java, Spring Boot, Cassandra (Backend Database)

Kafka (Middleware message Broker), React Redux (Frontend)

Firebase (Caching) and Amazon Web Services (Cloud Deployment)

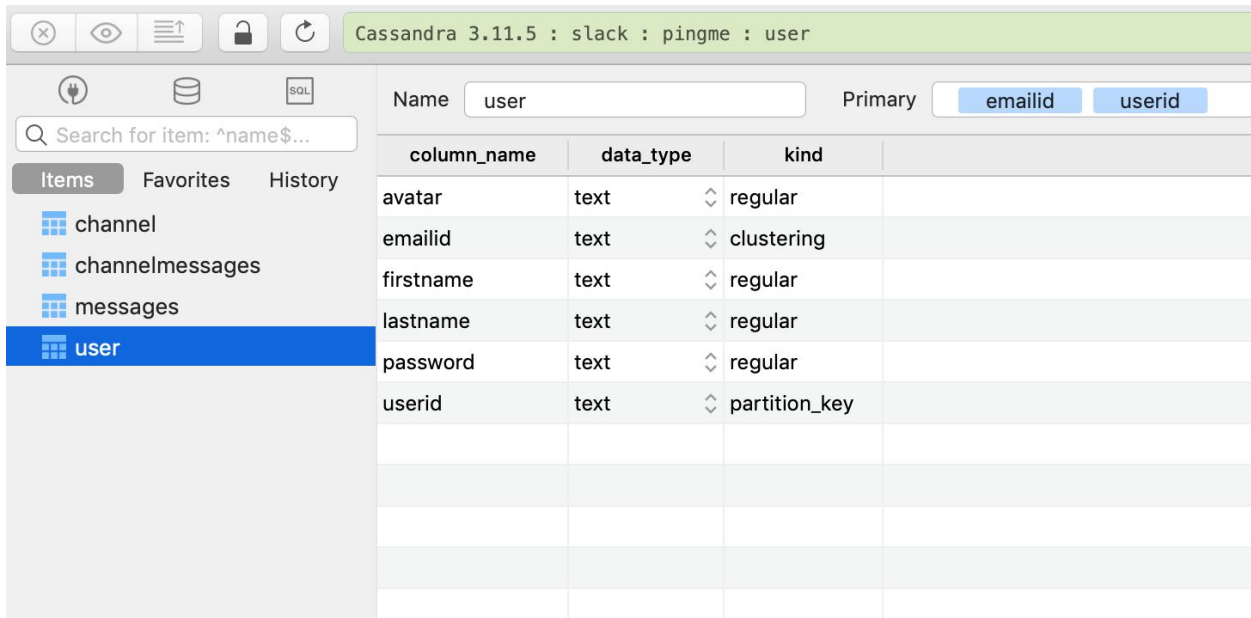
- **Cassandra KeySpace Creation :**

```
CREATE KEYSPACE pingme WITH replication = {'class':'SimpleStrategy',  
'replication_factor' : 3};
```

- **Cassandra Tables**

1. **User**

```
CREATE TABLE user(  
  avatar text,  
  emailid text,  
  userid text,  
  firstname text,  
  lastname text,  
  password text,  
  PRIMARY KEY(emailid, userid)  
);
```



The screenshot shows the Cassandra 3.11.5 GUI. The top bar indicates the current context is 'slack : pingme : user'. On the left, a sidebar lists database items: 'channel', 'channelmessages', 'messages', and 'user' (which is selected and highlighted in blue). The main panel displays the table 'user' with its primary keys 'emailid' and 'userid'. Below this, a table lists the columns and their properties:

column_name	data_type	kind	
avatar	text	regular	
emailid	text	clustering	
firstname	text	regular	
lastname	text	regular	
password	text	regular	
userid	text	partition_key	

```
SELECT * from user;
```

The screenshot shows the Cassandra 3.11.5 GUI with the 'user' table selected. The table contains the following data:

userid	emailid	avatar	firstname	lastname	password
ping	ping@gmail.com	gravatar.com/ping	ping	ping	pingping
batman	batman@gmail.com	gravatar.com/batman	batman	wayne	batmanbatman
ironman	ironman@gmail.com	gravatar.com/ironman	iron	man	ironman
spartan	spartan@gmail.com	gravatar.com/spartan	sammy	spartan	sammyspartan
spiderman	spiderman@gmail.com	gravatar.com/spiderman	spider	man	spiderman
dan	dan@gmail.com	gravatar.com/dan	dan	mattes	danmattes
blackwidow	blackwidow@gmail.com	gravatar.com/blackwidow	black	black	blackblack

2. Channel

```
CREATE TABLE channel(  
  channelid uuid PRIMARY KEY,  
  channelname text,  
);
```

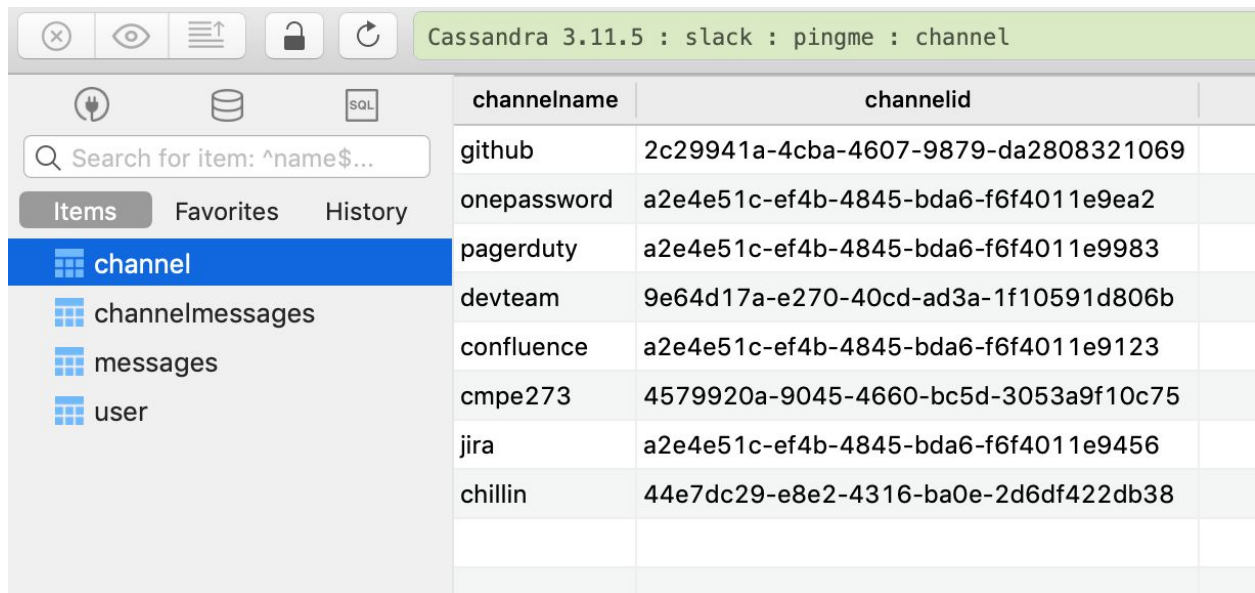
The screenshot shows the Cassandra 3.11.5 GUI with the 'channel' table selected. The table structure is as follows:

column_name	data_type	kind
channelid	uuid	regular
channelname	text	partition_key

Data Insertion Query

```
INSERT INTO channel (channelid, channelname)  
VALUES("CMPE273");
```

```
SELECT * FROM Channel;
```

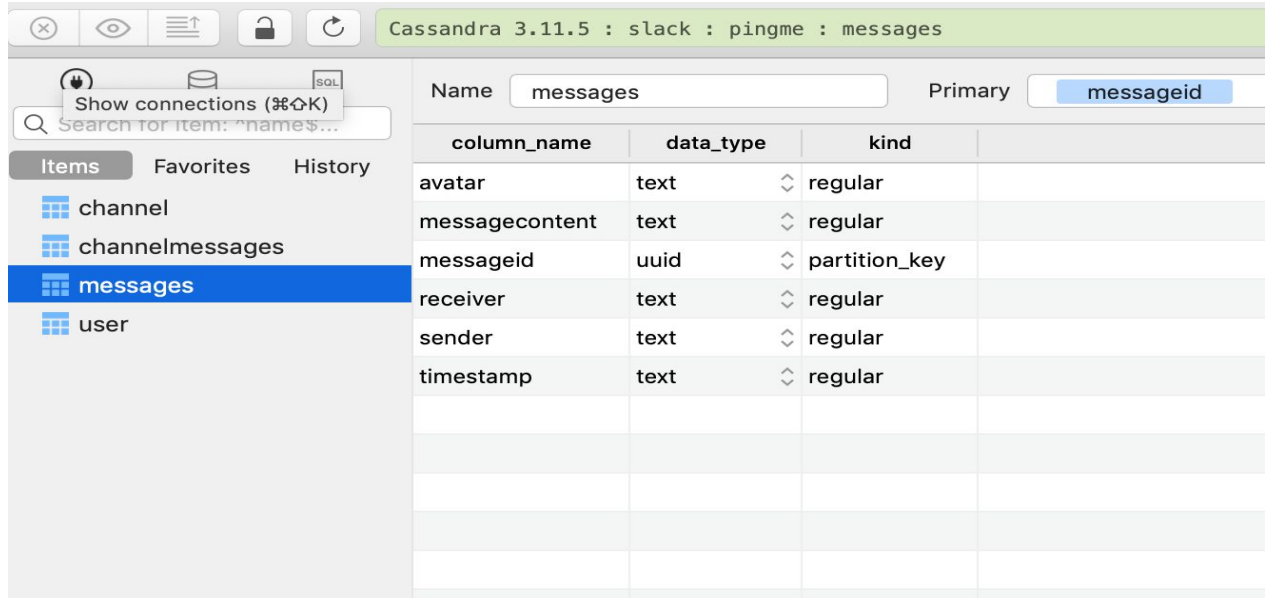


The screenshot shows a database client window titled "Cassandra 3.11.5 : slack : pingme : channel". The interface includes a search bar with the text "Search for item: ^name\$...", tabs for "Items", "Favorites", and "History", and a sidebar with a tree view containing "channel", "channelmessages", "messages", and "user". The "channel" item is selected, displaying a table with two columns: "channelname" and "channelid".

channelname	channelid
github	2c29941a-4cba-4607-9879-da2808321069
onpassword	a2e4e51c-ef4b-4845-bda6-f6f4011e9ea2
pagerduty	a2e4e51c-ef4b-4845-bda6-f6f4011e9983
devteam	9e64d17a-e270-40cd-ad3a-1f10591d806b
confluence	a2e4e51c-ef4b-4845-bda6-f6f4011e9123
cmpe273	4579920a-9045-4660-bc5d-3053a9f10c75
jira	a2e4e51c-ef4b-4845-bda6-f6f4011e9456
chillin	44e7dc29-e8e2-4316-ba0e-2d6df422db38

3. Messages

```
CREATE TABLE messages(  
  avatar text,  
  messageid uuid PRIMARY KEY,  
  sender text,  
  receiver text,  
  messagecontent text,  
  timestamp text,  
);
```



```
SELECT * FROM messages;
```

Cassandra 3.11.5 : slack : pingme : messages

messageid	avatar	messagecontent	receiver	sender	timestamp
a46d05c9-ff58-45ca-845b-0ff7f5cf8bd4	https://www.gravatar.com/avatar/94d093eda664ad...	Hi Spiderman!	spiderman	testuser	12 Nov
dcfa9bc1-1dde-4ff9-9953-f41d81d4b8fa	https://www.gravatar.com/avatar/94d093eda664ad...	hell yeah !	spartan	batman	12 Nov
f93c7f0c-85de-468f-9291-c641a52a0860	https://www.gravatar.com/avatar/94d093eda664ad...	Hey Dan	dan	srinivas	12 Nov
9d57f656-79dd-46f7-964d-3f5de0665745	https://www.gravatar.com/avatar/94d093eda664ad...	Hey Spiderman!	spiderman	newuser	12 Nov
c2649bfd-3b1f-4179-823c-c59d5e6b93e6	https://www.gravatar.com/avatar/94d093eda664ad...	msdnjcfklv	ironman	batman	12 Nov
006526d2-8be5-4ba9-9f3e-031700bcd56	https://www.gravatar.com/avatar/94d093eda664ad...	Hey Batman!	batman	ironman	12 Nov
1731ea51-646c-4694-a8bc-0d6738d2bf73	https://www.gravatar.com/avatar/94d093eda664ad...	Hey batsy	batman	spartan	12 Nov

4. Channelmessages

```
CREATE TABLE channelmessages(
  messageid uuid PRIMARY KEY,
  avatar text,
  channelname text,
  timestamp text,
  sender text,
  messagecontent text,
);
```

Cassandra 3.11.5 : slack : pingme : channelmessages

Name: channelmessages Primary: messageid

column_name	data_type	kind
avatar	text	regular
channelname	text	regular
messagecontent	text	regular
messageid	uuid	partition_key
sender	text	regular
timestamp	text	regular

Search for item: ^name\$...

Items Favorites History

- channel
- channelmessages
- messages
- user

SELECT * FROM channelmessages;

Cassandra 3.11.5 : slack : pingme : channelmessages

messageid	avatar	channelname	messagecontent	sender	timestamp
561b2292-492a-4295-aaa7-eac9b76d111f	https://www.gravatar.com/avatar/94d093eda664ad...	github	Hey this is sammy the spartan texting	spartan	12 Nov
fffeab68-092e-41ae-904c-284de169c056	https://www.gravatar.com/avatar/94d093eda664ad...	github	Hey!	batman	12 Nov
58416870-81b6-4ccd-b0eb-245344baa346	https://www.gravatar.com/avatar/94d093eda664ad...	jira	Hey!	batman	12 Nov
c11b3c8a-5953-44b4-9cca-020ebb2fb581	https://www.gravatar.com/avatar/94d093eda664ad...	github	This is Dan texting on github channel	dan	Nov 12
1bc4f116-a02f-45a4-b592-a942f1af079d	https://www.gravatar.com/avatar/94d093eda664ad...	chillin	Hey Spiderman!	spartan	12 Nov
c11b3c8a-5953-44b4-9cca-020ebb2fb585	https://www.gravatar.com/avatar/94d093eda664ad...	pagerduty	This is Spiderman!	spiderman	Nov 12

Search for item: ^name\$...

Items Favorites History

- channel
- channelmessages
- messages
- user

To check if the Cassandra Database is running
Command : `sudo service cassandra status`

```
ubuntu@ip-10-0-0-86:~$ sudo service cassandra status
cassandra.service - LSB: distributed storage system for structured data
Loaded: loaded (/etc/init.d/cassandra; generated)
Active: active (running) since Tue 2019-11-26 14:42:17 UTC; 15min ago
Docs: man:systend-sysv-generator(8)
Process: 838 ExecStart=/etc/init.d/cassandra start (code=exited, status=0/SUCCESS)
Tasks: 86 (limit: 1152)
CGroup: /system.slice/cassandra.service
└─1183 java -Xloggc:/var/log/cassandra/gc.log -ea -XX:+UseThreadPriorities -XX:ThreadPriorityPolicy=42 -XX:+HeapDumpOnOutOfMemoryError -Xss256k -XX:StringTableSize=1000003 -XX:+AlwaysPreTouch
Nov 26 14:42:16 ip-10-0-0-86 systend[1]: Starting LSB: distributed storage system for structured data...
Nov 26 14:42:17 ip-10-0-0-86 systend[1]: Started LSB: distributed storage system for structured data.
```

The three node cassandra cluster looks like this. Internal Ips as **10.0.0.86**, **10.0.0.242** and **10.0.0.126**

```
ubuntu@ip-10-0-0-86:~$ sudo nodetool status
Datacenter: us-west
=====
Status=Up/Down
// State=Normal/Leaving/Joining/Moving
-- Address      Load      Tokens     Owns (effective)  Host ID
UN 10.0.0.242    382.7 KiB  256        100.0%            b09451a7-1
6ef-47df-935b-a56887a3403d 1a
UN 10.0.0.86     310.12 KiB 256        100.0%            bf3a0747-
a1b6-4454-a1b9-d5dbb6f345e5 1a
UN 10.0.0.126    340.1 KiB 256        100.0%            61a62179-5
5292-4dd3-8e56-46ba20fcac5e 1a

ubuntu@ip-10-0-0-86:~$ cqlsh 10.0.0.242 9042
Connected to Slack at 10.0.0.242:9042.
[cqlsh 5.0.1 | Cassandra 3.9 | CQL spec 3.4.2 | Native protocol v4]
Use HELP for help.
cqlsh> use pingme;
cqlsh:pingme>

ubuntu@ip-10-0-0-242:~$ sudo nodetool status
Datacenter: us-west
=====
Status=Up/Down
// State=Normal/Leaving/Joining/Moving
-- Address      Load      Tokens     Owns (effective)  Host ID
UN 10.0.0.242    382.7 KiB  256        100.0%            b09451a7-
10ef-47df-935b-a56887a3403d 1a
UN 10.0.0.86     310.12 KiB 256        100.0%            bf3a0747-
a1b6-4454-a1b9-d5dbb6f345e5 1a
UN 10.0.0.126    340.1 KiB 256        100.0%            61a62179-
5292-4dd3-8e56-46ba20fcac5e 1a

ubuntu@ip-10-0-0-242:~$ cqlsh 10.0.0.86 9042
Connected to Slack at 10.0.0.86:9042.
[cqlsh 5.0.1 | Cassandra 3.9 | CQL spec 3.4.2 | Native protocol v4]
Use HELP for help.
cqlsh> use pingme;
cqlsh:pingme>

ubuntu@ip-10-0-0-126:~$ sudo nodetool status
Datacenter: us-west
=====
Status=Up/Down
// State=Normal/Leaving/Joining/Moving
-- Address      Load      Tokens     Owns (effective)  Host ID
UN 10.0.0.242    382.7 KiB  256        100.0%            b09451a7-
10ef-47df-935b-a56887a3403d 1a
UN 10.0.0.86     310.12 KiB 256        100.0%            bf3a0747-
a1b6-4454-a1b9-d5dbb6f345e5 1a
UN 10.0.0.126    340.1 KiB 256        100.0%            61a62179-
5292-4dd3-8e56-46ba20fcac5e 1a

ubuntu@ip-10-0-0-126:~$ cqlsh 10.0.0.86 9042
Connected to Slack at 10.0.0.86:9042.
[cqlsh 5.0.1 | Cassandra 3.9 | CQL spec 3.4.2 | Native protocol v4]
Use HELP for help.
cqlsh> use pingme;
cqlsh:pingme>
```


Cassandra Load Balancer

The screenshot shows the AWS Management Console interface for a Cassandra Load Balancer. The left sidebar contains navigation links for various AWS services. The main content area displays the 'Instances' tab for the load balancer named 'cassandra'. A table lists three instances, each with an Instance ID, Name, Availability Zone, Status, and Actions. Below the table, there is a section for 'Edit Availability Zones' showing the Subnet ID, Subnet CIDR, Instance Count, and Health status.

Instance ID	Name	Availability Zone	Status	Actions
i-0224d5b2a584d9560	2Cassandra	us-west-1a	InService (1)	Remove from Load Balancer
i-0032ad9a6344fb590	3Cassandra	us-west-1a	InService (1)	Remove from Load Balancer
i-051c782054dcb8f42	1Cassandra	us-west-1a	InService (1)	Remove from Load Balancer

Availability Zone	Subnet ID	Subnet CIDR	Instance Count	Healthy?	Actions
us-west-1a	subnet-0ed9e2495104ea1c5	10.0.0.0/24	3	Yes	-

The screenshot shows the AWS Management Console interface for a Cassandra Load Balancer, displaying the 'Basic Configuration' and 'Port Configuration' tabs. The 'Basic Configuration' section provides details about the load balancer's name, DNS name, type, scheme, availability zones, creation time, hosted zone, status, and VPC. The 'Port Configuration' section shows the port configuration for the load balancer, indicating that 9042 (TCP) is forwarding to 9042 (TCP) and that stickiness options are not available for TCP protocols. The 'Security' section shows the source security group for the load balancer.

Name	Value	Creation time	Hosted zone	Status	VPC
Name	cassandra	November 15, 2019 at 9:17:02 PM UTC-8	Z368ELLRRE2KJO	3 of 3 instances in service	vpc-0ab2db50b23f527db
* DNS name	cassandra-321448472.us-west-1.elb.amazonaws.com (A Record)				
Type	Classic (Migrate Now)				
Scheme	internet-facing				
Availability Zones	subnet-0ed9e2495104ea1c5 - us-west-1a				

Port Configuration

Port Configuration: 9042 (TCP) forwarding to 9042 (TCP)
Stickiness options not available for TCP protocols

Security

Source Security Group: sg-013e25d53e7020c0e, cassandra
quick-create-1 created on Friday, November 15, 2019 at 9:15:39 PM UTC-8

START THE KAFKA SERVERS

ZOOKEEPER

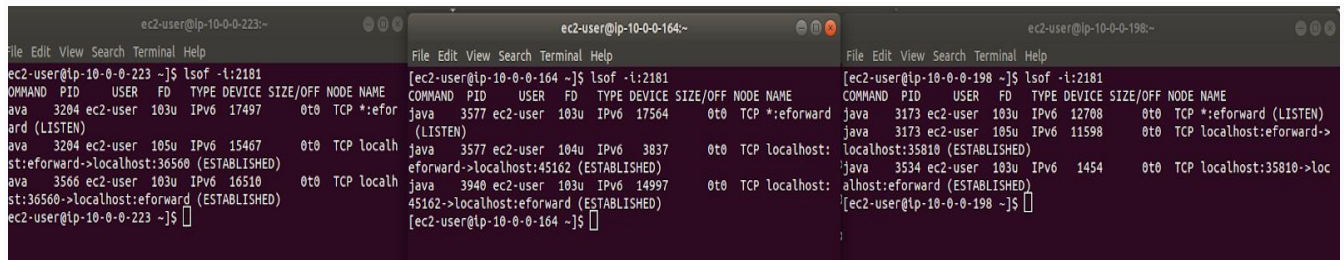
```
>$ nohup bin/zookeeper-server-start.sh config/zookeeper.properties >
~/zookeeper-logs &
```

KAFKA SERVER

```
>$ nohup bin/kafka-server-start.sh config/server.properties > ~/kafka-logs &
```

To check if Kafka servers are up and running :

```
>$ lsof -i:2181
```



The image shows three terminal windows from different EC2 instances, each displaying the output of the command `lsof -i:2181`. The output lists the processes (java) and their network connections (FD, TYPE, DEVICE, SIZE/OFF, NODE NAME) for the port 2181. The connections are established to localhost:2181.

Instance IP	Process	FD	Type	Device	Size/Off	Node Name
10-0-0-223	java 3204	ec2-user 103u	IPv6	17497	0t0	TCP *:eforward (LISTEN)
10-0-0-164	java 3577	ec2-user 103u	IPv6	17564	0t0	TCP *:eforward (LISTEN)
10-0-0-198	java 3173	ec2-user 103u	IPv6	12708	0t0	TCP *:eforward (LISTEN)

START THE BACKEND SERVER

Build the Spring Kafka code, Since maven is being used -

```
> $ mvn package
```

A java jar gets created in the target directory from the project folder.

```
> $ cd target
```

```
> $ java -jar SpringKafkaSendConsumeJavaObject-0.0.1.jar
```

The three instance terminals look something like this -



The image displays three terminal windows side-by-side, each showing the output of a Kafka application. The windows are titled 'ec2-user@ip-10-0-0-223:~/Backend/SpringKafkaSendConsumeJavaO...', 'ec2-user@ip-10-0-0-164:~/Backend/SpringKafkaSendConsumeJavaObject...', and 'ec2-user@ip-10-0-0-198:~/Backend/SpringKafkaSendConsumeJavaObject/target...'. Each terminal shows a series of log messages from the 'k.l.KafkaMessageListenerContainer' and 'k.c.c.internals.AbstractCoordinator' classes. The logs indicate the process of joining groups, setting newly assigned partitions, and successfully joining groups. The messages are timestamped and include log levels like INFO and DEBUG. The logs are repeated for each of the three instances, showing the same sequence of events.

The Backend Load Balancer registers all the three Backend instances

Services

Resource Groups

Administrator @ 8212-2269-8261

N. California

Support

New EC2 Experience

Tell us what you think

EC2 Dashboard

Events

Tags

Reports

Limits

INSTANCES

Instances

Instance Types

Launch Templates

Spot Requests

Savings Plans

Reserved Instances

Dedicated Hosts

Capacity Reservations

IMAGES

AMIs

Bundle Tasks

ELASTIC BLOCK STORE

Volumes

Snapshots

Lifecycle Manager

NETWORK & SECURITY

Security Groups

Elastic IPs

Placement Groups

Create Load Balancer

Actions

Filter by tags and attributes or search by keyword

< 1 to 2 of 2 >

Name	DNS name	State	VPC ID	Availability Zones	Type	Created At	Monitoring
BackendLoadBalancer	BackendLoadBalancer-1314...		vpc-0ab2db50b23f527db	us-west-1a	classic	November 15, 2019 at 9:42:...	
cassandra	cassandra-321448472-us-w...		vpc-0ab2db50b23f527db	us-west-1a	classic	November 15, 2019 at 9:17:...	

Load balancer: BackendLoadBalancer

Description

Instances

Health check

Listeners

Monitoring

Tags

Migration

Basic Configuration

Name

BackendLoadBalancer

Creation time

November 15, 2019 at 9:42:57 PM UTC-8

* DNS name

BackendLoadBalancer-1314010567.us-west-1.elb.amazonaws.com (A Record)

Hosted zone

Z368ELLRRE2KJ0

Type

Classic (Migrate Now)

Status

3 of 3 instances in service

Scheme

internet-facing

VPC

vpc-0ab2db50b23f527db

Availability Zones

subnet-0ed9e2495104ea1c5 - us-west-1a

Port Configuration

Port Configuration

3000 (TCP) forwarding to 8080 (TCP)

Stickiness options not available for TCP protocols

Security

Source Security Group

sg-0209c7db00629f5be, Spring-Boot-Secure

Spring-Boot-Secure

Edit security groups

aws

Services

Resource Groups

Administrator @ 8212-2269-8261

N. California

Support

New EC2 Experience

Tell us what you think

EC2 Dashboard

Events

Tags

Reports

Limits

INSTANCES

Instances

Instance Types

Launch Templates

Spot Requests

Savings Plans

Reserved Instances

Dedicated Hosts

Capacity Reservations

IMAGES

AMIs

Bundle Tasks

ELASTIC BLOCK STORE

Volumes

Snapshots

Create Load Balancer

Actions

Filter by tags and attributes or search by keyword

< 1 to 2 of 2 >

Name	DNS name	State	VPC ID	Availability Zones	Type	Created At	Monitoring
BackendLoadBalancer	BackendLoadBalancer-1314...		vpc-0ab2db50b23f527db	us-west-1a	classic	November 15, 2019 at 9:42:...	
cassandra	cassandra-321448472-us-w...		vpc-0ab2db50b23f527db	us-west-1a	classic	November 15, 2019 at 9:17:...	

Load balancer: BackendLoadBalancer

Description

Instances

Health check

Listeners

Monitoring

Tags

Migration

Connection Draining: Enabled, 300 seconds (Edit)

Edit Instances

Instance ID	Name	Availability Zone	Status	Actions
i-0ef8e8e912c9e5c7d	2SPRING-LARGE	us-west-1a	InService	Remove from Load Balancer
i-01e197e86cfe4b661	1SPRING-LARGE	us-west-1a	InService	Remove from Load Balancer
i-02b2aa2088f21fe38	3SPRING-KAFKA	us-west-1a	InService	Remove from Load Balancer

Edit Availability Zones

Availability Zone	Subnet ID	Subnet CIDR	Instance Count	Healthy?	Actions
us-west-1a	subnet-0ed9e2495104ea1c5	10.0.0.0/24	3	Yes	-

Start the Front-End Servers

```
cd /path/to/Front-end/Code
```

```
npm install
```

```
Npm start
```

Register the three Front-end servers with the Elastic Load Balancer

The screenshot displays the AWS Management Console interface for an Elastic Load Balancer. The left sidebar shows navigation options like EC2 Dashboard, Events, Tags, Reports, Limits, INSTANCES, IMAGES, ELASTIC BLOCK STORE, and NETWORK & SECURITY. The main content area shows a list of load balancers with columns for Name, DNS name, State, VPC ID, Availability Zones, Type, Created At, and Monitoring. The 'Front-End-Load-balancer' is selected and highlighted in blue.

Below the list, the configuration details for the 'Front-End-Load-balancer' are shown. The 'Basic Configuration' section includes:

- Name:** Front-End-Load-balancer
- * DNS name:** Front-End-Load-balancer-2089848249.us-west-1.elb.amazonaws.com (A Record)
- Creation time:** November 26, 2019 at 8:14:30 AM UTC-8
- Hosted zone:** Z368ELLRRE2KJ0
- Type:** Classic (Migrate Now)
- Status:** 3 of 3 instances in service
- Scheme:** internet-facing
- VPC:** vpc-0ab2db50b23f527db
- Availability Zones:** subnet-0ed9e2495104ea1c5 - us-west-1a

The 'Port Configuration' section shows:

- Port Configuration:** 3000 (TCP) forwarding to 3000 (TCP)
- Stickiness options:** not available for TCP protocols

The 'Security' section shows:

- Source Security Group:** sg-0209c7db00629f5be, Spring-Boot-Secure
- Default VPC security group:** sg-0281c4fd040180c49, default

The screenshot displays the AWS Management Console interface for configuring an Elastic Load Balancing (ELB) instance. The left sidebar shows the navigation menu with categories like INSTANCES, IMAGES, and ELASTIC BLOCK STORE. The main content area shows the 'Create Load Balancer' page with a table of existing load balancers. Below this, the 'Front-End-Load-balancer' is selected, and the 'Instances' tab is active. This tab shows a table of instances currently associated with the load balancer, all in 'InService' status. Below the instances table, the 'Availability Zones' section shows that the load balancer is configured with three instances across the 'us-west-1a' availability zone.

Name	DNS name	State	VPC ID	Availability Zones	Type	Created At	Monitoring
BackendLoadBalancer	BackendLoadBalancer-1314...		vpc-0ab2db50b23f527db	us-west-1a	classic	November 15, 2019 at 9:42:...	
cassandra	cassandra-321448472.us-w...		vpc-0ab2db50b23f527db	us-west-1a	classic	November 15, 2019 at 9:17:...	
Front-End-Load-balancer	Front-End-Load-balancer-20...		vpc-0ab2db50b23f527db	us-west-1a	classic	November 26, 2019 at 8:14:...	

Load balancer: Front-End-Load-balancer

Connection Draining: Enabled, 300 seconds (Edit)

Edit Instances

Instance ID	Name	Availability Zone	Status	Actions
i-0898fa612419cb294	01_FRONT_END	us-west-1a	InService ⓘ	Remove from Load Balancer
i-0d3ff224af258306	03_FRONT_END	us-west-1a	InService ⓘ	Remove from Load Balancer
i-09a5525879c589bfa	02_FRONT_END	us-west-1a	InService ⓘ	Remove from Load Balancer

Edit Availability Zones

Availability Zone	Subnet ID	Subnet CIDR	Instance Count	Healthy?	Actions
us-west-1a	subnet-0ed9e2495104ea1c5	10.0.0.0/24	3	Yes	-

The Application starts running at port 3000 and the Elastic Load Balancer is listening on port 3000.

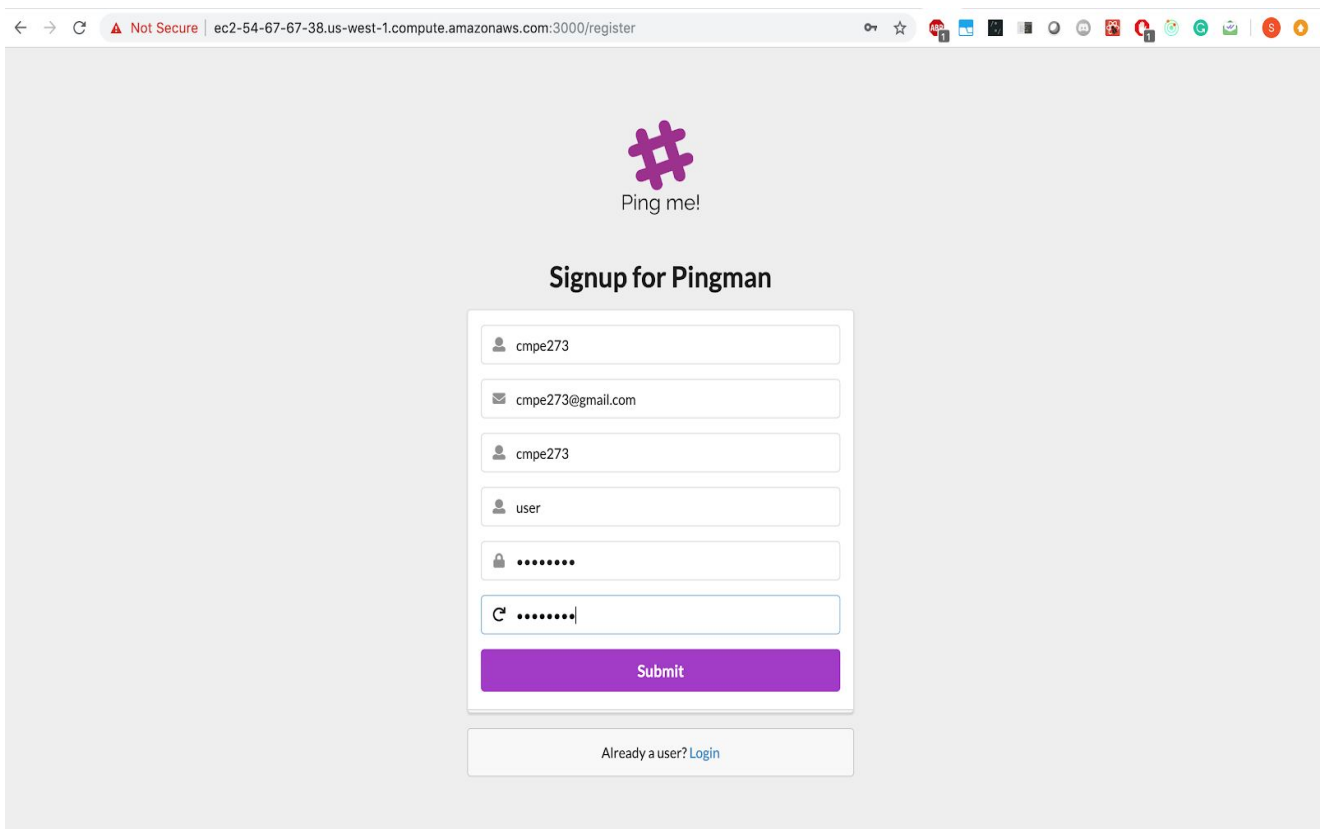
The Application can be accessed using the ELB's DNS Name followed by the port Number.

RESULTS


The Application can be accessed using the DNS Name of the Front-End Load Balancer followed by port 3000.

DNS_NAME:3000/login


The user needs to first register to access the Application. Say user cmpe273 tries to register





← → ↻ ⚠ Not Secure | ec2-54-67-67-38.us-west-1.compute.amazonaws.com:3000/register



Ping me!


Signup for Pingman


 cmpe273

 cmpe273@gmail.com

 cmpe273

 user

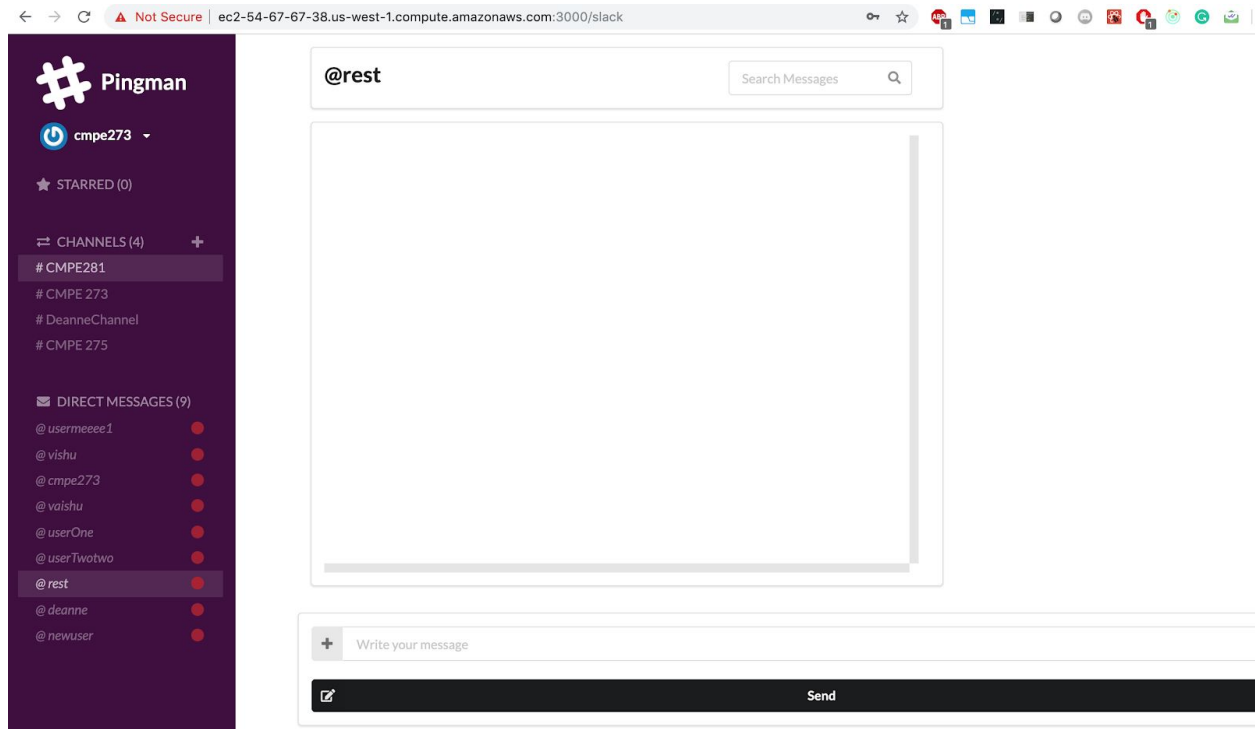




Submit

Already a user? [Login](#)

Once logged in, the application window looks like




```
[ubuntu@ip-10-0-0-242:~$ cqlsh 10.0.0.242 9042
Connected to Slack at 10.0.0.242:9042.
[cqlsh 5.0.1 | Cassandra 3.9 | CQL spec 3.4.2 | Native protocol v4]
Use HELP for help.
[cqlsh> use pingme
[ ... ;
[cqlsh:pingme> select * from user;
```

emailid	userid	avatar	firstname	lastname	password
useme@gmail.com	usermeeee1	avataaam	useme	everyday	passstthewater
vishu@gmail.com	vishu	gravatar.com/vishu	Vaishali	Kaul	00120012
vaishali@gmail.com	vaishu	gravatar.com/vaishu	Vaishali	Koul	Vaishali
user01.gmmi.com	user0ne	avatarmy	first101	last01	passss101
user02.yahoo.com	userTwotwo	avatarmy	first102	last02	passss102
restuser@gmail.com	rest	gravatar.com/rest	rest	user	restuser
deanne@gmail.com	deanne	gravatar.com/deanne	deanne	charan	deannecharan
newuser@gmail.com	newuser	gravatar.com/newuser	new	user	newuser

```
(8 rows)
[cqlsh:pingme> select * from user;
```

emailid	userid	avatar	firstname	lastname	password
useme@gmail.com	usermeeee1	avataaam	useme	everyday	passstthewater
vishu@gmail.com	vishu	gravatar.com/vishu	Vaishali	Kaul	00120012
cmpe273@gmail.com	cmpe273	gravatar.com/cmpe273	cmpe273	user	cmpeuser
vaishali@gmail.com	vaishu	gravatar.com/vaishu	Vaishali	Koul	Vaishali
user01.gmmi.com	user0ne	avatarmy	first101	last01	passss101
user02.yahoo.com	userTwotwo	avatarmy	first102	last02	passss102
restuser@gmail.com	rest	gravatar.com/rest	rest	user	restuser
deanne@gmail.com	deanne	gravatar.com/deanne	deanne	charan	deannecharan
newuser@gmail.com	newuser	gravatar.com/newuser	new	user	newuser

```
(9 rows)
cqlsh:pingme>
```

After registration, the user details get stored in cassandra (Backend database). The Screenshot above shows the cassandra state before and after the user, cmpe273 registration.

As the front-end stores user credentials in Firebase Database, The Firebase Database maintains the state before and after registration as shown in the following Screenshots

Search by email address, phone number, or user UID

[Add user](#)

Identifier	Providers	Created	Signed In	User UID ↑
newuser@gmail.com	📧	Nov 26, 2019	Nov 26, 2019	4IUto0KkjHWgJ24jnNOedizR3gv2
testy@gmail.com	📧	Nov 26, 2019	Nov 26, 2019	RzF0WIZZBrRrAmuSYOpYSU3JpB...
dan@gmail.com	📧	Nov 26, 2019	Nov 26, 2019	ZB233PlrT9RWHrK2rNr8XzXvjLG2
mssrinivasbhargav@gmail.c...	📧	Nov 9, 2019	Nov 10, 2019	syPNpoTd8JOct4TeHXmmhbYpL0...

Rows per page: 50 1-4 of 4

As it can be observed that the User Authentication credentials are stored by Firebase. CMPE273 user is not found until registered at a later stage.

Search by email address, phone number, or user UID

[Add user](#)

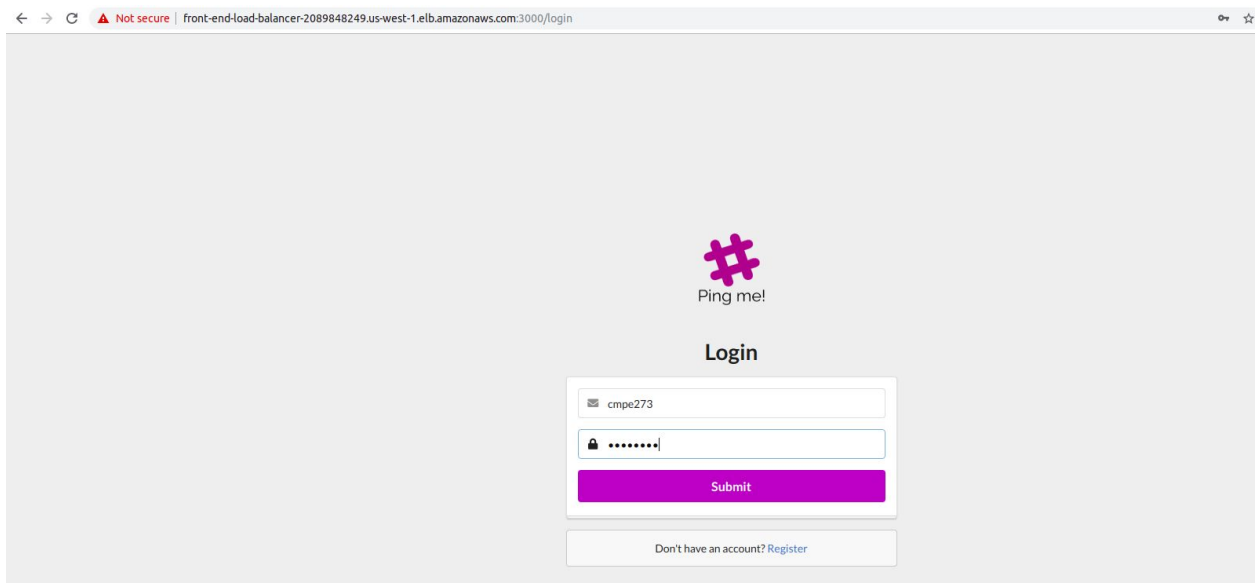
Identifier	Providers	Created	Signed In	User UID ↑
newuser@gmail.com	📧	Nov 26, 2019	Nov 26, 2019	4IUto0KkjHWgJ24jnNOedizR3gv2
testy@gmail.com	📧	Nov 26, 2019	Nov 26, 2019	RzF0WIZZBrRrAmuSYOpYSU3JpB...
dan@gmail.com	📧	Nov 26, 2019	Nov 26, 2019	ZB233PlrT9RWHrK2rNr8XzXvjLG2
cmpe273@gmail.com	📧	Nov 26, 2019	Nov 26, 2019	nLqBcOPV24UF75IiHn36dNbuVAu1
mssrinivasbhargav@gmail.c...	📧	Nov 9, 2019	Nov 10, 2019	syPNpoTd8JOct4TeHXmmhbYpL0...

Rows per page: 50 1-5 of 5


User Login Module

For the user to login, first the user credentials are checked in the firebase database and if not found there, the backend database cassandra is looked into and the user details are called from there. The user needs to enter login details to access the application and post messages.

User “cmpe273” trying to log in.



← → ↻ ⚠ Not secure | front-end-load-balancer-2089848249.us-west-1.elb.amazonaws.com:3000/login


Ping me!

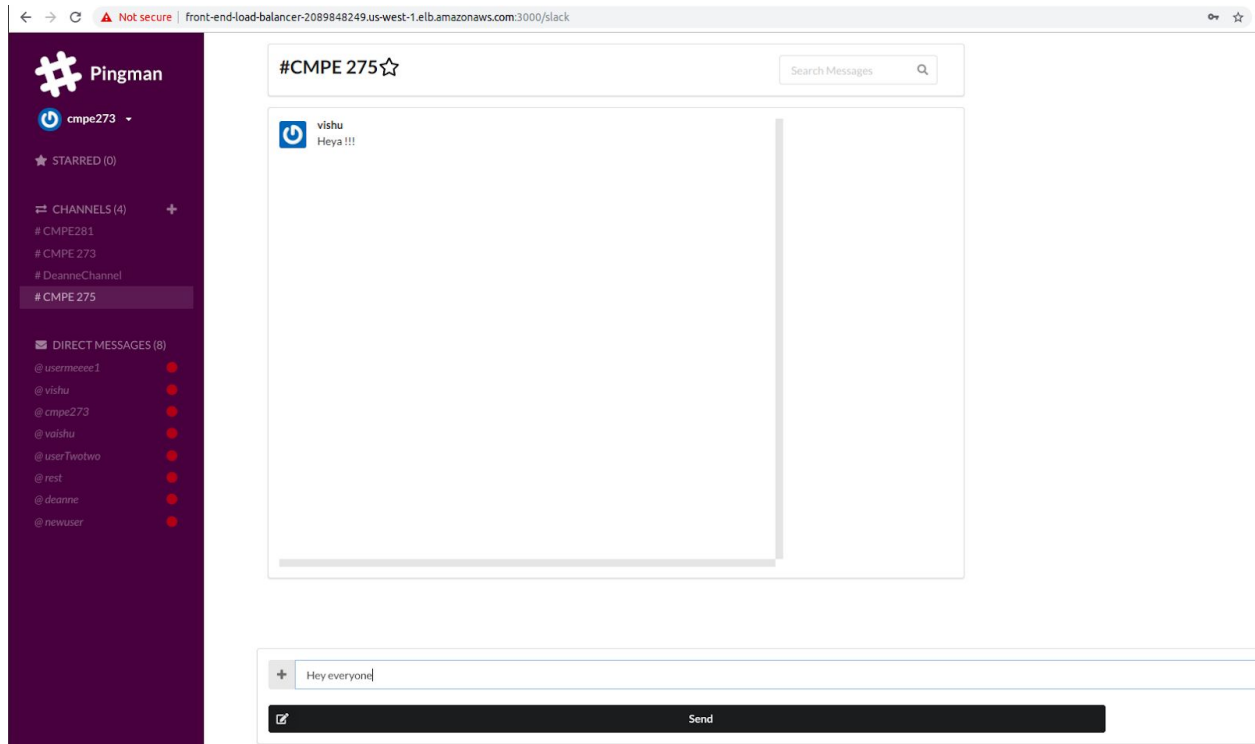
Login

Submit

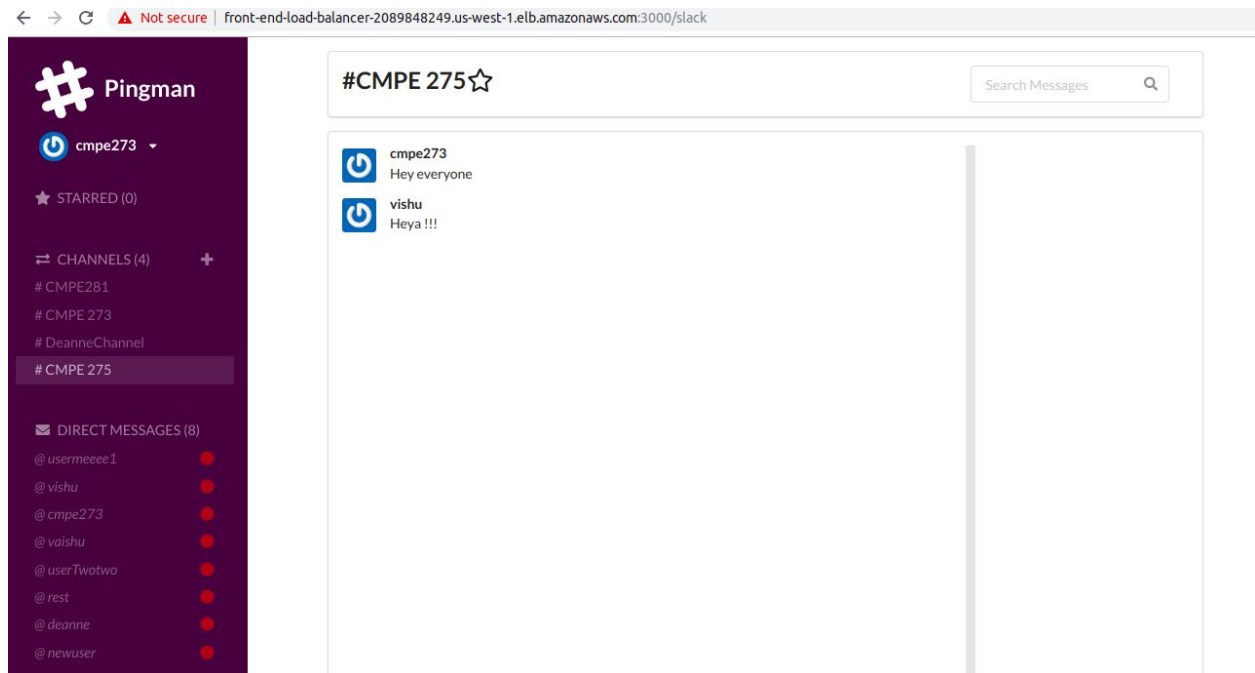
Don't have an account? [Register](#)

User “cmpe273” logged in

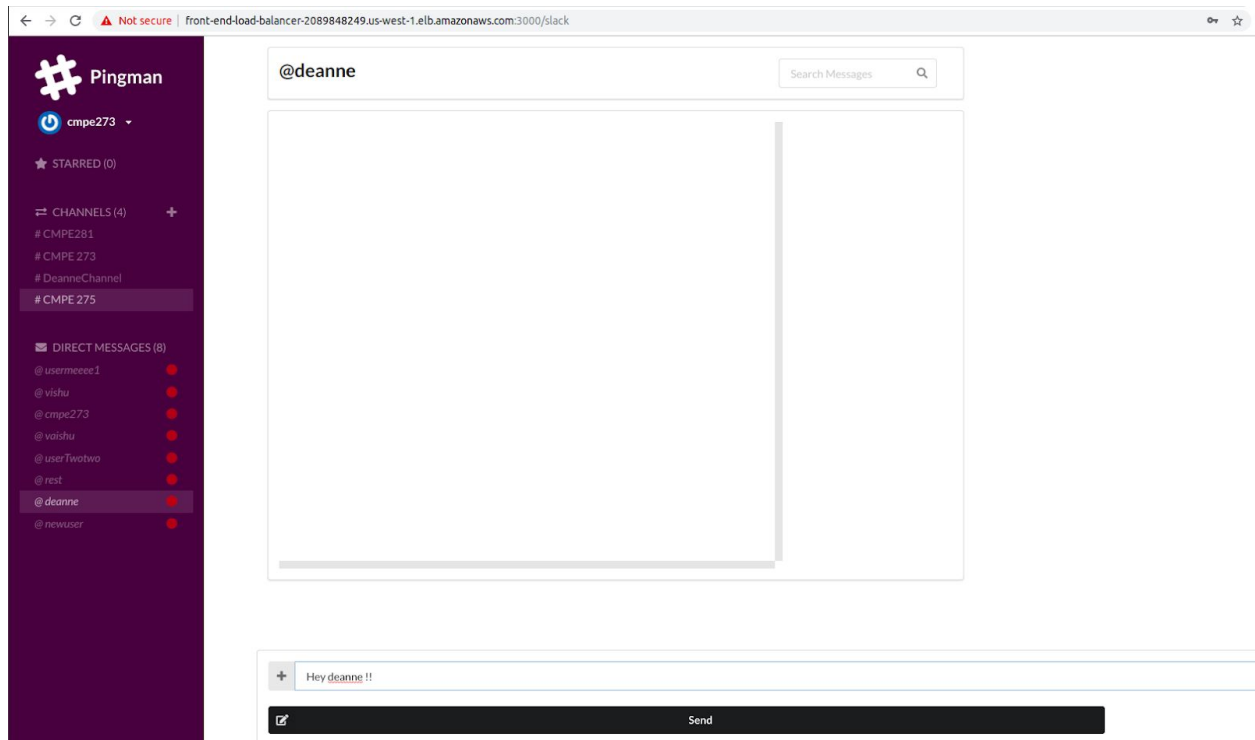
User “cmpe273” viewing channel “cmpe275” and about to send message “Hey everyone” there



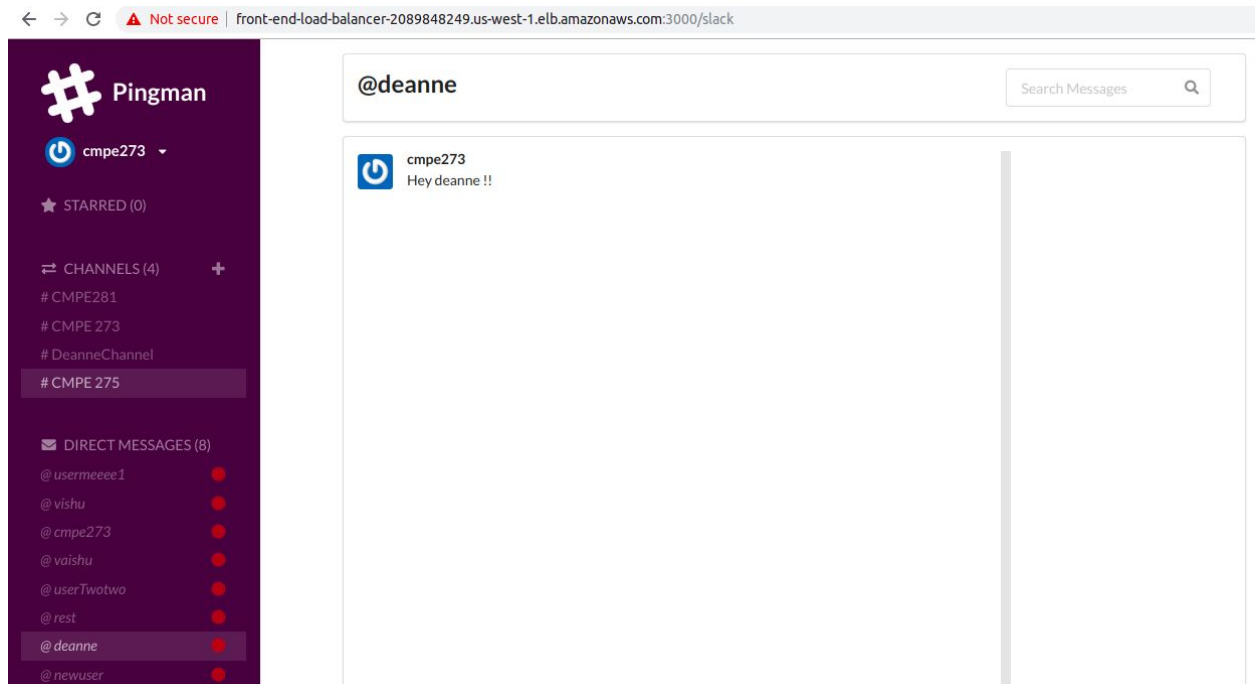
Followed by user “cmpe273” posting successfully the message



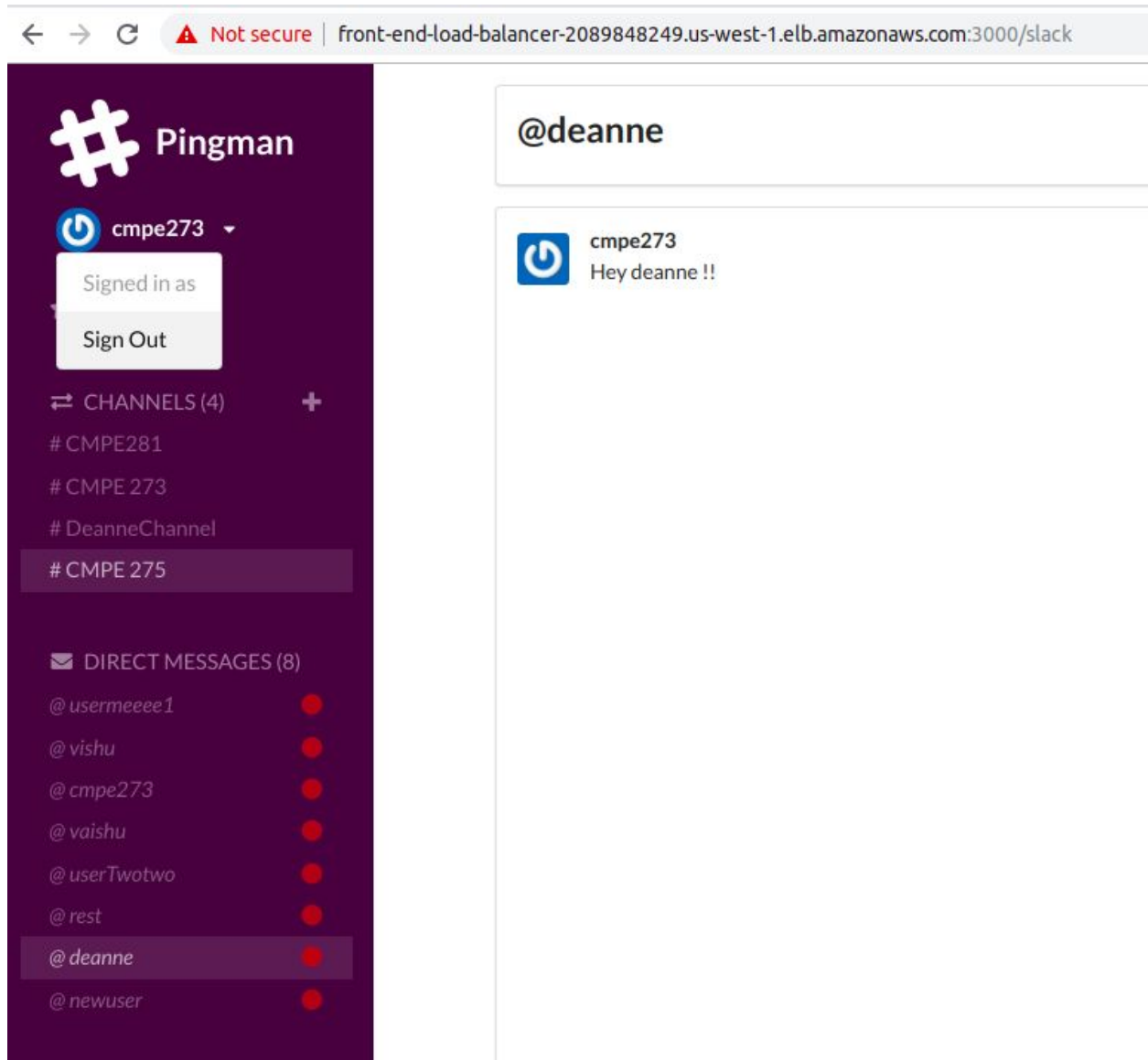
User “cmpe273” trying to post a “Hey deanne!!” message to another user “deanne”



Followed by successfully direct messaging “Hey deanne !!” to user “deanne”.



At the end , the user can sign out which will take the application to an initial login page for other users to access the application.



The steps explained previously give a walk through of the application. How the developed Application acts as a messaging system. Now the question arises, how does the application developed fulfil the idea for a fault-tolerant, durable messaging system.

1. At the Database Level -

The Application uses a Cassandra cluster with sets of 3 cassandra databases which maintain replicas of the data being generated (the messages in this case). Also for larger, load intensive systems more cassandra servers can be added to the cluster set. Additionally, if even one cassandra server goes down, the others would still be up and store data durably. The cluster set is being served by a Load balancer which forwards the requests in a fair manner within all the database servers which makes the Application capable of handling high load.

2. At the Backend (Spring Boot Level)-

The Backend servers are again a replica set of three with a load balancer at the end forwarding requests fairly. Even if one or two Backend servers go down, the system application will still be functioning.

3. At the Kafka (Middleware)

Kafka has itself been developed to be fault tolerant. As long as the producer (the client side) is accepting requests, the Kafka message queue will still be storing messages durably in its topics and queuing them. When the consumer (Spring Boot) comes back up, the messages can be inserted to the database.

4. At the Front-End Level

Generally distributed systems are all about taking care of the Backend, this messaging application makes sure that the front-end is fault tolerant too. The front end has been replicated in 3 servers to ensure availability during peak time.

5. The Load Balancers in the Deployment

Each replicated servers part has been served with Load Balancers to ensure that the load gets distributed in a fair fashion and no server gets overburdened while others are free.

CONCLUSION

Kafka replacing the conventional messaging systems increases an application's performance during outages by making sure that the messages get delivered at least once.

It queues the messages until the Consumer is ready to consume the messages back again. Additionally, contrary to traditional message systems, kafka has a retention period for messages and doesn't completely remove the message until the retention period is over.

Also, using a load balancer to ensure that requests get distributed fairly during sudden network traffic increase. This again ensures system performance and availability because rather than letting one server crash and die down during network spikes, it equally forwards the load. The biggest challenge for this project is the Spring kafka integration. Having the application hosted to cloud is a great idea in terms of allowing scalability and replicas in the Messaging application system. It queues the messages until the Consumer is ready to consume the messages back again.

RECOMMENDATION

The current Application has been developed and tested using data at a small scale but if released in production, this application may have to encounter huge traffic spike abruptly at times. Hence, it is necessary for it to be tested under such circumstances.

Due to resource constraints and budget issues, Real Time testing with massive data was not possible but should be considered in future.

Many more functionalities can be added, like auto git commits, searching for users registered with the application using their user names and some security measures should be considered to make the application more secure.

REFERENCES

1. [Overcoming Queue problems at Slack](#)
2. [Kafka: A distributed Messaging system for log processing](#)
3. [Integrating Apache Kafka with Spring](#)
4. [Confluent : An Intro of Kafka with Spring Boot](#)
5. [Creating a Spring Boot Apache Kafka Application](#)

APPENDIX

1. Apache Kafka - A message streaming platform that helps in transmitting messages between systems and applications. It also helps create data pipelines for streaming.
2. Kafka Topic - An SQL Table like entity which is just a bunch of continuously being produced records. Used to store messages in Kafka.
3. Kafka Producer - Pushes messages to the kafka topics [The react front-end in this case].
4. Kafka Consumer - Pulls messages from the Kafka topic [The Backend Spring boot Application that inserts data into the databases].
5. Message offset - Kafka does not store messages with Ids. Instead, there are message offsets that Kafka Broker stores and the consumer is responsible for acknowledging offsets with the broker, hence keeps track of the amount of messages that have been consumed.
6. Elastic Load Balancer - Handles the incoming traffic and distributes the requests fairly among all the EC2 instances or servers that it is serving.
7. Cassandra - An AP Database that is used to store data and has capability of handling massive amounts of requests.