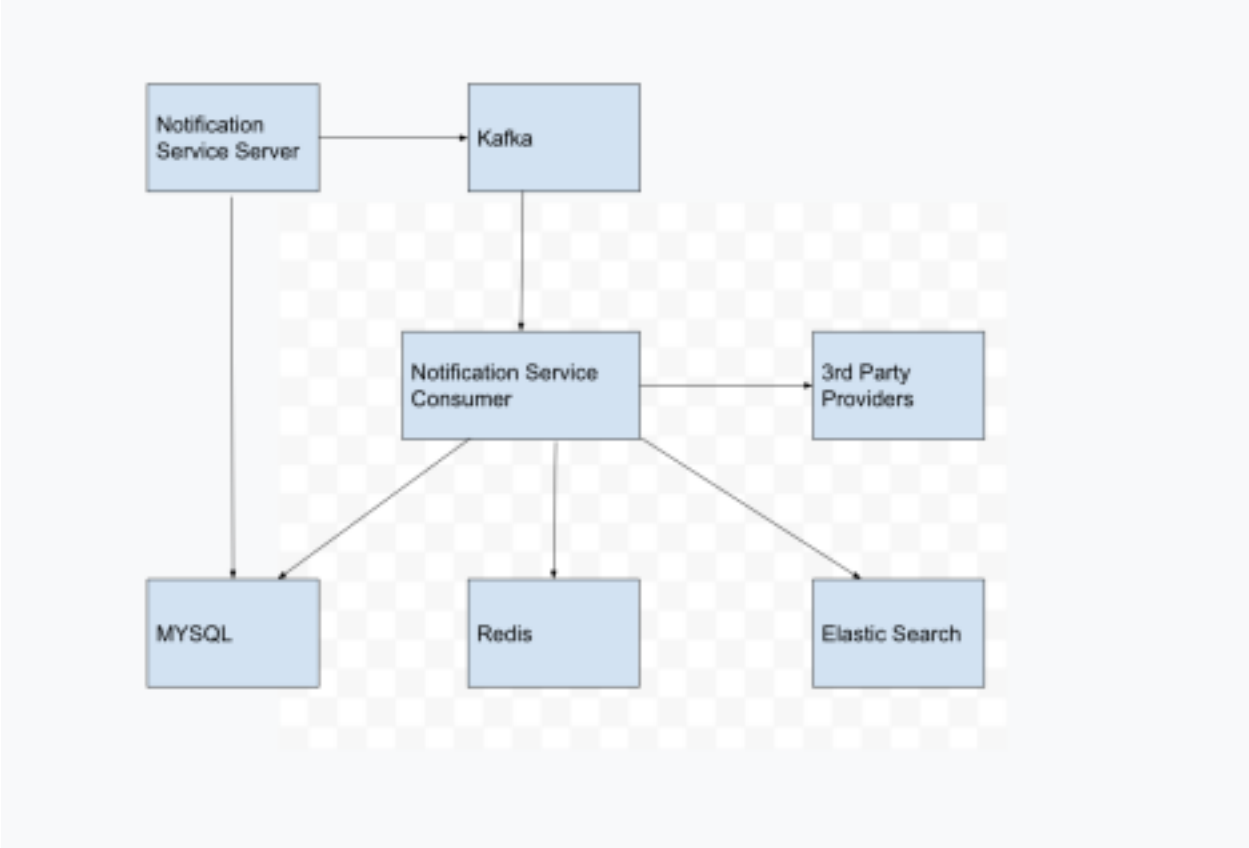
Notification Service

Goal

We will build a Notification Service from scratch which can send SMS to users. Idea is to get hands-on experience in Java, Springboot, Mysql, Kafka, Redis, Elastic Search.

Design



Flow

1. Notification Service will get the phone number and message to send in /sms/send api. It will insert the details in sms_requests table and publish request_id to kafka topic -> notification.send_sms
2. Notification consumers will get the message id from kafka, and do the following steps.
    a. get the request details from db,
    b. check if number is blacklisted via redis,
    c. call the 3rd party api to send sms,
    d. update the status,message_id,failure_details if any in DB,
    e. index the details in elastic search.
3.

Sample Mysql DB Schema

Table: sms_requests

id
phone_number
message
status
failure_code
failure_comments
created_at
updated_at

Use appropriate data types.

Notification Service API

Send SMS API

POST /v1/sms/send	

Body	<pre>{
	  "phoneNumber": "+919940630272",
	  "message":"Welcome to Meesho!"
	}</pre>

Success Response:	<pre>{
	  "data": {
	  "requestId": "some_unique_id",
	  "comments": "Successfully Sent"
	  }
	}</pre>

Sample Failure Response:
Http status : 400
	<pre>{
	  "error": {
	  "code": "INVALID_REQUEST",
	  "message": "phone_number is mandatory"
	  }
	}</pre>

Add a phone_number to blacklist -> Update cache and DB

POST /v1/blacklist	

Body	<pre>{
	  "phoneNumbers": [
	  "+91904353454534",
	  "+91980998787678"
	  ]
	}</pre>

Success Response:	<pre>{
	  "data": "Successfully blacklisted"
	}</pre>

Remove a number from blacklist

DELETE /v1/blacklist	

Body	<pre>{
	  "phoneNumbers": [
	  "+91904353454534",
	  "+91980998787678"
	  ]
	}</pre>

Success Response:	<pre>{
	  "data": "Successfully whitelisted"
	}</pre>

Get list of blacklisted numbers

GET /v1/blacklist	

Success Response:	<pre>{
	  "data": [
	    "+91904353454534",
	    "+91980998787678"
	  ]
	}</pre>

Get details of sms from request_id (Get from Mysql DB)

GET /v1/sms/<request_id>	

Success Response:	<pre>{
	  "data": {
	    "id":
	    "phoneNumber":
	    "message":
	     ……. All other details from the table …...
	  }
	}</pre>

Sample Failure Response:
Http status : 404
	<pre>{
	  "error": {
	    "code": "INVALID_REQUEST",
	    "message": "request_id not found"
	  }
	}</pre>

Get all sms sent to phone number between given start and end time (With Pagination) (Get from elastic search)

Please use the above contract as examples to design responses and requests for these.

Get all sms containing given text (With Pagination) (Get from elastic search).

Please use the above contract as examples to design responses and requests for these.

3rd Party API

This is a 3rd party api, we get charged for each SMS. Please use it limitedly. Do not share it anywhere.

POST https://api.imiconnect.in/resources/v1/messaging	
Headers	Key: <Will be Shared>

Body	<pre>[
	  {
	    "deliveryChannel": "sms",</pre>

	<pre>"channels": {
	"sms": {
	"text": "Hello, Greetings from Meesho.Click here to know more about us:https://meesho.com."
	}
	},
	"destination": [
	{
	"msisdn": [
	"+919940630272"
	],
	"correlationId": "some_unique_id"
	}
	]
	}
	]</pre>

Implementation Requirements

- Your service should include instrumentation by using https://github.com/Meesho/instrumentation
- You should also log using log4j and include sleuth for distributed tracing. Use https://www.baeldung.com/spring-cloud-sleuth-single-application for context.
- The service classes for your codebase should be unit tested. Guidelines:- https://meesho.atlassian.net/wiki/spaces/EW/pages/2247917618/Unit+Testing

Recommended Practices

- All api should have an authentication header
- Add logs wherever required. Debugging should be easier
- Handle Exceptions, API timeouts
- Use spring boot and spring-jpa. Avoid using other ORM'S
- Structure your code into controllers,services,repositories,models,constants,utils
- Discuss with your buddy, different failure cases that can happen in this system and solutions for them.
- Discuss with your buddy on how different components in this system can be scaled.
- Deploy your code in the sandbox environment and verify to get hands-on on working with remote machines.

Getting Started

- Building a web server - https://spring.io/guides/gs/rest-service
- Accessing data from Mysql DB - https://spring.io/guides/gs/accessing-data-jpa/ Spring Kafka - https://www.confluent.io/blog/apache-kafka-spring-boot-application/
- Elastic Search Java Client:- https://www.elastic.co/guide/en/elasticsearch/client/java-rest/current/java-rest-high.html

Deployments

- The service needs to be deployed in sandbox environments.

General Notes

- You will be evaluated and plagiarism checks will be done.