# Summary

The Real-Time Notification System (RTNS) is a cloud based solution for push notifications.

# Workflow

Subscribers open a websocket and subscribe to topics of their own choice. A separate REST endpoint accepts request for topics and notifies the subscribers of those topics by sending them a notification with the topic name over the opened websocket.

# API

## Websocket endpoint

### Subscribe

{“message”: “subscribe”, “topics”:[“topicName”, “secondTopicName”]}

### Unsubscribe:

{“message”: “unsubscribe”}

### Client disconnect:

same as unsubscribe

## REST endpoint /notify

POST {“topics”:[“topicName”, “secondTopicName”]}

# Under the hood

## Subscribe

Calls an AWS Lambda function which stores the subscription in DynamoDB. The connection ID is used as a subscriber ID. A subscription is set of topics a specific subscriber is interested in.

## Unsubscribe

Calls an AWS Lambda function which deletes all records of that subscriber from the DynamoDB table. The connection ID is used as a subscriber ID. The same happens on disconnect.

## Notify

Accepting a POST request with a set of topic names, the desired final result of this call is all related subscribers to receive a push notification. As the subscribers could be thousands, the process is split in two.

The original call will trigger a lambda function. The function will build notification objects based on queries on the DynamoDB store. It will then store them in Amazon SQS (notification queue).

Another lambda function is subscribed to the notification queue. It will process the notifications in batches and push them to the subscribers by calling the Api Gateway.

If some connections are gone, their Ids will be sent to the “gone queue” and will be deleted from the DynamoDB store asynchronously by a subscribed lambda function.

# Projects overview

## RTNS.Core

Contains core technology-agnostic concepts. Most of the functionality is left to the implementation. E.g. synchronous or asynchronous notifications.

## RTNS.AWS

An AWS implementation of the core concepts. Adopts asynchronous notifications by using a queue and a subscriber lambda for scalability. The store is a DynamoDB table with inverted GSI for lookup of subscribers by topics.

## RTNS.AWS.Subscriptions and RTNS.AWS.Notifications

Define the lambda handlers for the websocket and rest API Gateways.

## RTNS.Core.Tests

Unit test for the core library.

## RTNS.IntegrationTests

Use the smoke tests to test if a deployment is working properly.

# Deployment

The whole solution is defined in a cloudformation template.

The deployment scripts require you to have a named AWS profile on your local machine. The DEV and QA versions will deploy directly to the selected environments. The PROD version will package the cloudformation file and upload it to an S3 bucket for the DEVOPS to deploy.