

### Task 3

Joona Pesonen 014713574

#### **Deliverables:**

Github link: <https://github.com/NorthernLion/DSTask3>

Report: **At the start of this pdf**

Screenshots: **At the start of Part 2 (explaining how the code works)**

#### **Part 1.**

Push architecture also known as server-based protocol is a way of messaging in which the server initiates the messaging by pushing a new message to all the clients it has. Determining who the clients of the server are can be done for example by hard coding them into the server files or using a publish-subscribe style architecture where client subscribes to a certain topic by contacting the server and telling what topics it is interested in. In other words it is mandatory that server keeps track of its clients.

Pull architecture also known as client-based protocol is a way of messaging in which the client initiates the messaging by requesting aka pulling the information from the server. In pulling architectures client needs to know where it can reach the server (usually through URL) and server doesn't need to know anything about the client beforehand. When a new message arrives at the server the server does nothing else than updates its message and then lets the clients to do all the work by requesting the information when they need it.

Because of certain network limitations there are cases where Push architecture is done through pulling by clients. In such cases often the client for example changes its address too often for server to be able to keep track of its origin. In such cases Each client can pull updates from server every 0.1 seconds for example to reach style of push architecture through use of pull architecture.

I already mentioned the main differences for pull and push architectures. These differences lead to certain trade offs. For example in push when new message arrives the server has to send out update to every client at the same time, which is not possible in reality. When the amount of clients increases there will be delays between the sending of the first message and the last message to clients. In pull architectures pull requests from single client can happen more often than the state of the server changes. This means that a client can request state of the server even though the state of the server has not changed. This leads to more than necessary amount of messaging.

To put it simply pull and push architecture are the opposite of each other and choosing between the two systems causes trade-offs. To give examples of areas in which certain architecture is often superior: Push-based in updating replicas of the main unit, in transaction between permanent and replica servers, in situations where high degree of consistency is required. Pull-based architectures are often used in client caches and websites in general. In general Push based is more efficient when read to update ratio is high and pull based when it is low.

## Part 2.

I first setup the FireBase and generated the keys shown in the following picture

The screenshot displays the 'Project credentials' section of the Firebase console. It includes a table of server keys, a section for iOS app configuration, and a section for web configuration with a table of web push certificates.

**Project credentials**

[Add server key](#)

Key	Token
Server key	AAAAAT7gFYkg:APA91bGxC007WUhs5k8Pctj5EhAYjOPfXKg1SOKem_5NV-nCSmDbgaq-doZJIUIM-sObtt2mw2Y14saEnEigmFwwkX7BvzYghopRHsynSfly4qjAJsdXz2Xalj4DxgxxVGUX6FvtmN_
Legacy server key <a href="#">?</a>	AlzaSyDsfa2CusyM0hcRi_PmAeW9TGsNGUCFzg
Sender ID <a href="#">?</a>	342389776968

**iOS app configuration**

You don't have an iOS app

**Web configuration**

**Web Push certificates**

Firebase Cloud Messaging can use Application Identity key pairs to connect with external push services. [Learn more](#)

Key pair	Date added
BPORiNidvxz0JwCGIjQ0iD-7md5-L0cvyEkVxMxcq51UB8EKJ406dADelrNGyGMt4L_hDTBoOX20r10y8IFz6k	12 Dec 2018

Code explained:

1. In order to send new message to FCM server run "Python Messaging.py -message="This is a message that will be sent in the body to all recipients"

Messaging.py sends messages with topic 'ds' to FCM. FCM responds with parameter "name" that it has created for the message sent and automatically forwards the message to clients subscribed to the topic.

```

northernpike@NoOneKnows:~/github/DSTask3$ python3 messaging.py --message="This is a message that will be sent in the body to all recipients"
FCM request body for message using common notification object:
{
  "message": {
    "topic": "ds",
    "notification": {
      "body": "This is a message that will be sent in the body to all recipients",
      "title": "FCM Notification using Push"
    }
  }
}
Message sent to Firebase for delivery, response:
{
  "name": "projects/distributedsystemsjp/messages/6755525632657114885"
}

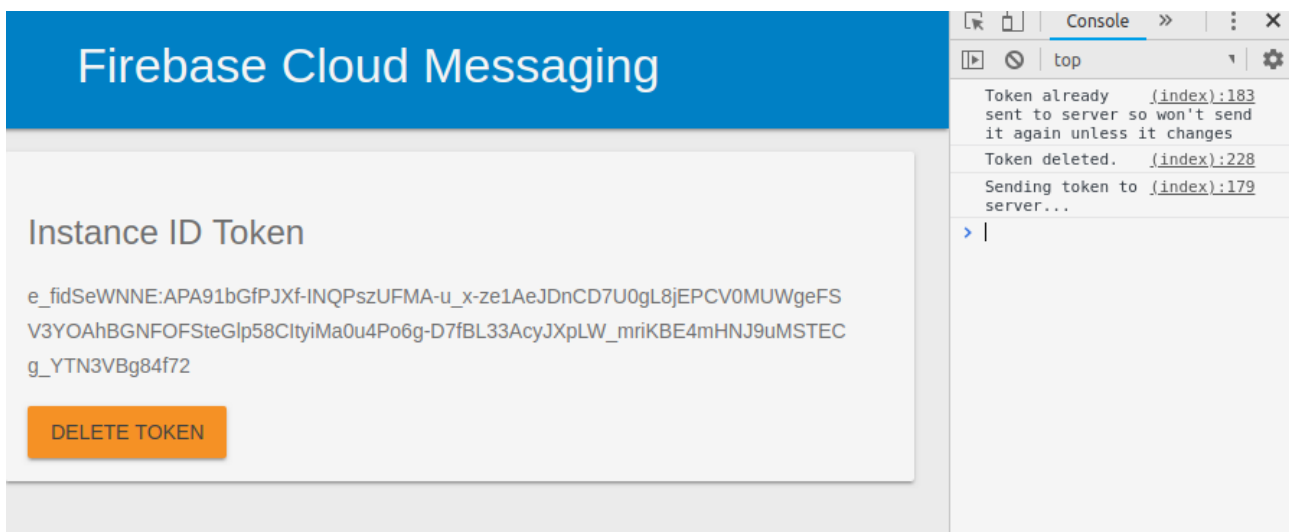
northernpike@NoOneKnows:~/github/DSTask3$ python3 messaging.py --message="Second message"
FCM request body for message using common notification object:
{
  "message": {
    "notification": {
      "body": "Second message",
      "title": "FCM Notification using Push"
    },
    "topic": "ds"
  }
}
Message sent to Firebase for delivery, response:
{
  "name": "projects/distributedsystemsjp/messages/6860877367160644472"
}

northernpike@NoOneKnows:~/github/DSTask3$ /

```

I had a lot of trouble making the client (The one subscribing to the server) using python (Where is the library?) so I ended up doing it in Node.js as it had the best documentation by FCM. I uploaded my client code to FireBase itself so it could use the API provided by FCM better. Using the API without uploading the code to FireBase seemed like a pain to do.

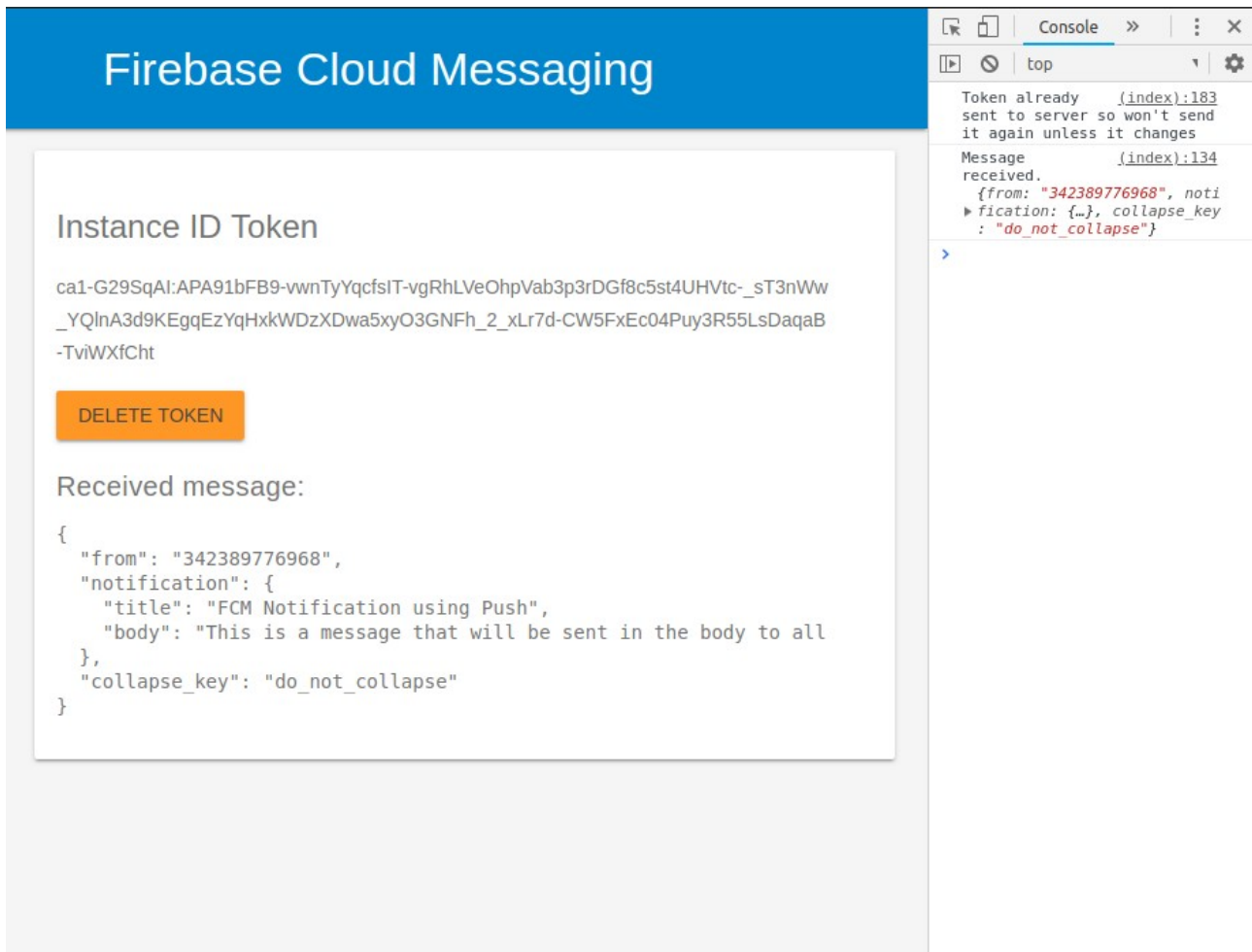
This is what the client looks like when started:



The screenshot shows the Firebase Cloud Messaging console. On the left, under 'Instance ID Token', a long alphanumeric token is displayed: `e_fidSeWNNE:APA91bGfPJXf-INQPszUFMA-u_x-ze1AeJDnCD7U0gL8jEPCV0MUWgeFSV3YOAhBGNFOFSteGlp58CltyiMa0u4Po6g-D7fBL33AcyJXpLW_mriKBE4mHNJ9uMSTECg_YTN3VBg84f72`. Below the token is an orange button labeled 'DELETE TOKEN'. On the right, the 'Console' tab is open, showing a log with the following messages: 'Token already (index):183 sent to server so won't send it again unless it changes', 'Token deleted. (index):228', and 'Sending token to (index):179 server...'. The console also shows a cursor at the bottom of the log area.

You can find the client running at: <https://distributedsystemsjp.firebaseio.com/> and multiple clients can be opened to simulate many clients being connected to the server. (Note different tabs do not work! You need to open separate browsers) Clicking delete token will delete the token and generate new one that will be sent to the server. The code of the client is in the public folder.

When messaging.py is ran the client will receive the message as following:



The screenshot displays the Firebase Cloud Messaging (FCM) console interface. The main content area shows the 'Instance ID Token' for a specific instance, with a long alphanumeric string and a 'DELETE TOKEN' button. Below this, the 'Received message' is shown as a JSON object. The console on the right side of the interface displays the log output, indicating that the token was already sent to the server and a message was received.

**Instance ID Token**

ca1-G29SqAl:APA91bFB9-vwnTyYqcfsIT-vgRhLveOhpVab3p3rDGf8c5st4UHVtc-\_sT3nWw\_YQlnA3d9KEgqEzYqHxkWDzXDwa5xyO3GNFh\_2\_xLr7d-CW5FxEc04Puy3R55LsDaqaB-TviWXfCht

**DELETE TOKEN**

**Received message:**

```
{
  "from": "342389776968",
  "notification": {
    "title": "FCM Notification using Push",
    "body": "This is a message that will be sent in the body to all"
  },
  "collapse_key": "do_not_collapse"
}
```

**Console Log:**

```
Token already (index):183
sent to server so won't send
it again unless it changes
Message (index):134
received.
{from: "342389776968", noti
fication: {...}, collapse_key
: "do_not_collapse"}
```

This is the basic idea behind the code. For measuring the times required in the following tasks I simply used DateTime logging at the python program and node client to calculate the time.

1. Well I would like to start off with the fact that measuring the time in the sender and client is hard. Because I implemented the two programs in different languages I had to use different libraries to calculate the time. Also I had to decide when I want to log the time in the programs. My decision was that I log the time on the sender of the message on the line before it sends the message to FCM and on the client when the message arrives. This means that the time calculation is started before the message has reached the actual FCM!

When using random sizes (varying from 1 B - 4 KB) I averaged 287 ms between receiving and sending times.

**2.**

min - 4 letter payload (around 16 bytes)

medium - 200 letter payload ( around 800 bytes)

max - 3000 letter payload (4KB)

The averages of sending 25 in each case

min - 293 ms

medium - 287 ms

max - 295 ms

From this we can see that such small difference (4 KB) in message size is meaningless. Most of the delay is caused from computing and networking of the messages. If FCM would accept messages over the size of 4 KB we could see larger differences in times.

**3.** The average inter between arriving messages was 500 ms. (0.5 seconds when trying to send them as fast as possible trough bash scripting. This delay was mainly caused by FCM and my messaging.py.

**4.** In my implementation as stated above the actual bottleneck was the process sending messages to FCM. I was able to send message every 500 ms without degrading performance.

In theory the answer to this question is that the maximum amount of messages depends on multiple things. First off the amount of clients that are receiving the messages on PUSH bases from the server limit the server a lot. If message delivery needs to be guaranteed the server will end up being the bottle neck at certain point.

In reality FCM is a cloud based system that should be able to handle any amount of clients trough the use of distributed servers. But as I am simply using the free to use system I believe the limit will be easy to reach.

**5.** Question about reliability of push architecture depends largely on the reliability on software architecture, networking architecture and server it has been built on. There are many ways push architecture can be implemented. My implementation uses python programming for server and node.js client to receive messages and FCM for implementing the push architecture. The main question about reliability cannot really be answered from the amount of testing I have done for this assignment as I have only used web based client and python client to test receiving the push messages. In reality what FCM is mostly used is for IOT and Mobile devices that generally move around more than my desktop. Reading from the Internet such as stack overflow, blogs and Fire Base web page it seems reliability of FCM is not very good and people are constantly complaining FCM delivering messages several seconds or even hours late. For me the messaging worked just fine the delay of 0.3 s from sender to receiver and 0.5 sec between messages worked just fine for me.