finiasz Initial release 512ce1c · yesterday

dynamodb	Initial release	yesterday
transfer.olvid.io	Initial release	yesterday
.gitignore	Initial release	yesterday
LICENSE	Initial release	yesterday
README.md	Initial release	yesterday

README AGPL-3.0 license

Olvid message distribution server

This repository contains the source code of the Lambda functions used in the Olvid message distribution server hosted at AWS. This code was stripped of some elements related to the paid version of Olvid (e.g., requests from Keycloak servers in the Enterprise version, or requests from the Olvid Store), but the rest of the code is untouched.

Structure

The project is made of two main parts:

- the `dynamodb` folder contains the code of the main Olvid distribution server with most Lambda functions behind a REST API Gateway at (https://server.olvid.io), and those starting with `ws` behind a WebSocket API Gateway at (https://ws-server.olvid.io).
- the `transfer.olvid.io` folder contains the code of the *transfer* server which is used when adding a device to your Olvid profile.

Inside both folders, an `src` folder contains a subfolder for each Lambda function.

The code is compiled with gradle, and the `build.gradle` file in each folder shows the external dependencies of each function. Compilation produces a ZIP file which can be uploaded to AWS Lambda.

The `CryptoLib` folder is common to all Lambda and also contains a `Constants.java` file which shows the structure of the DynamoDB tables used by the server, or the various SQS queues used internally. Some fields have been *anonymized* to protect our infrastructure.

Inside a Lambda function

Each Lambda function extends a different `RequestHandler` depending on the type of request they respond to.

Most functions inherit from our `Base64RequestStreamHandler` which is used for functions exposed in our REST API. This API uses our binary `Encoded` format for POST requests to API Gateway, which are encoded in base64 together with an API version (passed by the app in a header of the request). This API version is how backward compatibility is ensured: our AWS server can still be used with the very first version of the app 😎. This includes versions that required a *proof of work* to upload a message!

Other functions inherit from:

- `RequestHandler<S3Event, Void>` to react to object creations on S3,
- `RequestHandler<SQSEvent, Void>` to process messages from a queue,
- `RequestHandler<ScheduledEvent, Void>` for time-based triggers (like a cron),
- `RequestHandler<DynamodbEvent, Void>` to react to DynamoDB insertions or deletions.

Interactions with the server

Looking at the code, you can follow the different interactions between apps and the server. Suppose Alice is sending a picture to Bob, the following queries will be made:

1. Alice uploads the encrypted message payload (the text part of the message and metadata about the attachment) along with an encrypted miniature (40x40 pixels) of the picture and a header for Bob's device using the `uploadMessageAndGetUids` entry point. She receives a signed S3 URL at which to upload the encrypted picture.
2. Alice uploads the encrypted picture to S3 using the signed URL.
3. An S3 event triggers the `monitorS3AttachmentChunks` Lambda function. This function checks that all attachments of the messages are fully uploaded and sends a push notification to Bob's device using the `notifyMessageComplete()` method of the `PushLib` library.
4. Bob receives the push notification, which wakes up his app to go list messages on the server.
5. Listing messages requires a valid server session. If Bob's device has no valid session, it creates one by calling the `requestChallenge` and `getToken` entry points.
6. Bob then calls `downloadMessagesAndListAttachments` to download the message's encrypted payloads, and the S3 URL to download the picture from.
7. Bob's device decrypts the message payload, then:
   - it sends a *delivery receipt* to Alice's device using the `uploadReturnReceipt` entry point,
   - it downloads the miniature preview using the `downloadMessageExtendedContent` entry point and decrypts it,
   - the picture's size being below the *automatic download threshold*, it is downloaded from S3 and decrypted.
8. The insertion of the delivery receipt in DynamoDB triggers the `wsSendReturnReceipts` Lambda, which

About

Source code of the lambda functions used in the Olvid message distribution server hosted on AWS

Readme
AGPL-3.0 license
Activity
Custom properties
5 stars
0 watching
0 forks
Report repository

Releases

No releases published

Packages

No packages published

Languages

● Java 100.0%

sends the receipt to Alice's device through it's WebSocket connection.

9. The message is marked as delivered, and Alice and Bob can now both enjoy a great picture.

The future: Olvid multiserver

We are currently working on another version of this message distribution server which will be portable and easily deployed to any cloud provider, or even a home server. We believe a multiserver architecture is the next step towards giving users full control over their data and guaranteeing their privacy and sovereignty.

Follow us on LinkedIn, X, or Bluesky for the latest news.

Feedback

If you have any feedback on this repository, feel free to contact us at opensource@olvid.io.

License

This code is licensed under the GNU Affero General Public License v3. The full license is available in LICENSE.