Visado 3 - Microservicios y Docker (v.1.0)

# Introducción

Basándose en su implementación de UNQfy del visado anterior ahora ud. deberá agregar a UNQfy la funcionalidad de “Notificar a usuarios de actualizaciones en datos de un artista”. Entre otras cosas, esto implica implementar un nuevo servicio web para enviar notificaciones por email.

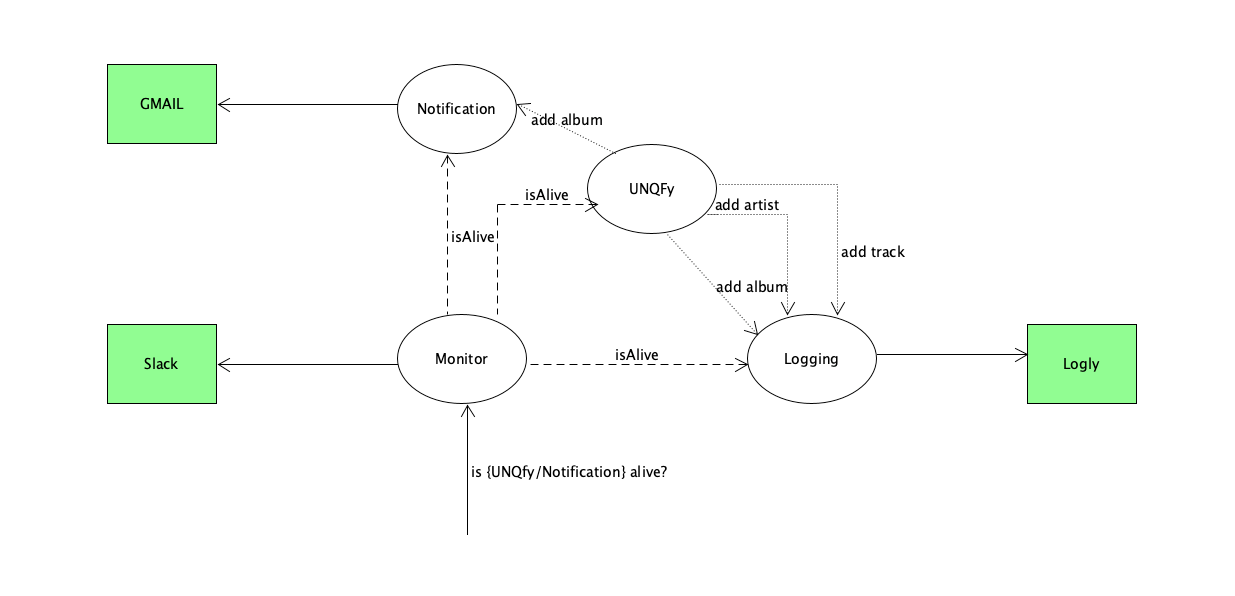
También se desea tener un registro (log) de diferentes sucesos del sistema, para poder auditar y debuggear en caso de ser necesario. Por cuestiones de backup y durabilidad de los datos queremos guardar los logs en dos lugares:

* localmente (en un archivo) y
* utilizando un servicio de terceros ( Loggly)

Estos servicios ( Unqfy, notificador y logger) deben ser monitoreados y, se desea avisar al equipo de desarrollo en caso de que algún servicio no esté funcionando. El equipo de desarrollo será avisado mediante un canal de Slack.

Finalmente deberá ejecutar los servicios de notificaciones, de logging, monitoreo y UNQfy en sus respectivos containers docker.

A continuación presentamos un diagrama general de los servicios y el flujo de información:



# Fase 1 - Notificando a “interesados” sobre nuevos álbumes.

Un *interesado* es una persona que desea saber cuando ocurren determinados eventos (como el agregado de un album) en UNQfy. Ellos no son, necesariamente, usuario de UNQfy.

Los *interesados*  desean saber cuando se publica un nuevo álbum de cierto artista de su interés. Cuando un nuevo álbum es agregado a un artista de su interés, el *interesado* recibirá un email con dicha información.

Tenga en presente desarrollar esto como un nuevo servicio (otra aplicación node+express). En principio la ejecutará en la misma máquina, pero luego lo harán en containers independientes.

Este servicio de notificaciones recibirá un mensaje desde UNQfy cuando se agrega un álbum y en función de las suscripciones enviará un email a quien corresponda avisando sobre el nuevo álbum.

### Recomendación de diseño

Para notificar a los *interesados* de un cambio y no “ensuciar” el modelo de UNQfy recomendamos usar el patrón [Observer](https://en.wikipedia.org/wiki/Observer_pattern). Para esto:

* Los objetos artista en UNQfy cumplirán el rol del observado (también llamado sujeto/Subject).
* Ud. debe implementar un objeto *observador* que, ante el agregado de un nuevo álbum de un artista, se comunicará con el servicio de notificación (vía el endpoint /api/notify), para notificar dicho cambio. Este objeto deberá agregarse como observer de todos los artistas.
* Cuando el servicio de notificación recibe el aviso (EP /api/notify), se buscan las suscripciones relacionadas con el artista al cual se le agregó el álbum, se arma y despacha el email para cada uno de los usuarios suscriptos a este artista.

## Servicio de notificaciones

El servicio de notificaciones debe proveer una API para suscribir y desuscribir usuarios al servicio. Esta API además debe permitir recibir notificaciones desde UNQfy, sobre los nuevos álbumes agregados. Este servicio enviará un email con la notificación a aquellos suscritos a dicho artista.

Escenarios de uso:

1. Una persona decide suscribirse a las novedades del artista Charly Garcia
   1. Debe obtener el id del artista a partir de la API de UNQfy
   2. Con ese ID se subscribirá en el servicio de notificaciones. El servicio de notificaciones debe usar la API de UNQfy para chequear que dicho artista exista. Si no existe el artista debe fallar de acuerdo a la tabla que se describe más abajo.
2. El usuario desea desubscribirse a las novedades del artista Charly Garcia
   1. Debe obtener el id del artista a partir de la API de UNQfy
   2. Con ese ID y el email con el que se subscribió invocará el endpoint listado más abajo.
   3. El sistema de notificación buscará si existe una entrada para ese artista y ese email, en caso de existir la borra, si no existe falla de acuerdo a la tabla que se describe más abajo.
3. A un artista de UNQfy se le agrega un album
   1. El servicio de Notificaciones se enterará del cambio
   2. Buscará a los usuarios interesados en ese artista.
   3. Les enviará una notificación al respecto
4. Se elimina un artista en UNQfy.
   1. Deberá eliminar todas las subscripciones para un artista en el servicio de notificaciones utilizando uno de los EP listado más abajo.

Note que el servicio de notificaciones se enterará de TODOS los agregados de álbumes en UNQfy.

A continuación se describe la API del servicio de notificaciones:

|  |  |
| --- | --- |
| **Titulo** | Suscribir un email a un artista |
| **Descripción** | Subscribe un email a un artista. Si el email ya está suscrito no hace nada.  Este EP chequea usando la API de UNQfy que el artista exista. |
| **Endpoint** | POST /api/subscribe |
| **Argumentos en Body** | {  "artistId": <arstistID>,  "email": "unemail@unDominio.com"  } |
| **Respuesta** | Status code: 200  Body: - (vacío) |
| **Errores** | 1. Artista Inexistente. 2. Falta un argumento. (email, artistId) 3. JSON invalido |

|  |  |
| --- | --- |
| **Titulo** | Desuscribir un email a un feed |
| **Descripción** | Desubscribe un email de un feed. Si el email no esta suscrito no hace nada.  Este EP chequea usando la API de UNQfy que el artista exista. |
| **Endpoint** | POST /api/unsubscribe |
| **Argumentos en Body** | {  "artistId": <arstistID>,  "email": "unemail@unDominio.com"  } |
| **Respuesta** | Status code: 200  Body: - (vacío) |
| **Errores** | 1. Artista Inexistente 2. Falta un argumento. (artistId, email) 3. JSON invalido |

|  |  |
| --- | --- |
| **Titulo** | Notificar a los usuarios suscritos a un feed |
| **Descripción** | Notifica vía mail, un mensaje a todos los usuarios suscritos a este artista. Se utilizará el subject, from y cuerpo de email pasados en el body del request.  Este EP chequea usando la API de UNQfy que el artista exista. |
| **Endpoint** | POST /api/notify |
| **Argumentos en Body** | {  "artistId": <artistID>,  "subject": "Nuevo Album para artsta Chano",  "message": "Se ha agregado el album XXX al artista Chano"  } |
| **Respuesta** | Status code: 200  Body: - (vacío) |
| **Errores** | 1. Artista Inexistente. 2. Falta algún argumento en el json. (artistID, subject, message). 3. JSON inválido. 4. Falló el envío de la notificación |

|  |  |
| --- | --- |
| **Titulo** | Obtener todos los emails suscritos a un artista |
| **Descripción** | Retorna el listado de suscriptores para el artista <artistID>.  Este EP chequea usando la API de UNQfy que el artista exista. |
| **Endpoint** | GET /api/subscriptions?artistId=<artistID> |
| **Argumentos en Body** |  |
| **Respuesta** | Status code: 200  Body: - {  "artistId": <artistID>,  "subscriptors": [<email1>, <email2>]  } |
| **Errores** | 1. Artista Inexistente. 2. Falta algún argumento en el json. (artistID). 3. JSON inválido. |

|  |  |
| --- | --- |
| **Titulo** | Eliminar todos los emails suscritos a un artista |
| **Descripción** | Borra todas las suscripciones para un artista (útil cuando se borra un artista en UNQfy).  Este EP chequea usando la API de UNQfy que el artista exista. |
| **Endpoint** | DELETE /api/subscriptions |
| **Argumentos en Body** | {  "artistId": <artistID>,  } |
| **Respuesta** | Status code: 200  Body: - |
| **Errores** | 1. Artista Inexistente. 2. Falta algún argumento en el json. (artistID). 3. JSON inválido. |

El manejo de errores también es parte de la API. A continuación se describen:

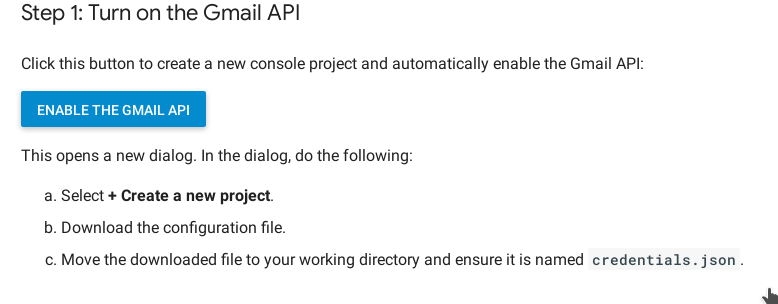
|  |  |  |
| --- | --- | --- |
| **Acción** | **Status Code Respuesta** | **Body Respuesta** |
| Falló el envío de la notificación | 500 | {  status: 500,  errorCode: "INTERNAL\_SERVER\_ERROR"  } |
| El artista al que se intenta suscribir/desuscribir/notificar/borrar no existe | 404 | {  status: 404,  errorCode: "RELATED\_RESOURCE\_NOT\_FOUND"  } |
| Falta un argumento en el JSON. | 400 | {  status: 400,  errorCode: "BAD\_REQUEST"  } |
| Se envía un Json invalido en el Body | 400 | {  status: 400,  errorCode: "BAD\_REQUEST"  } |
| URL invalida/inexistente | 404 | {  status: 404,  errorCode: "RESOURCE\_NOT\_FOUND"  } |
| Fallo inesperado | 500 | {  status: 500,  errorCode: "INTERNAL\_SERVER\_ERROR"  } |

### 1.1.2. Mandando e-mails.

Para enviar emails desde nuestros servicios, vamos a utilizar la API de GMail.

Para utilizar la API necesitan: una cuenta de gmail, crear un proyecto en developers.google.com y habilitar la API de GMAIL para el proyecto.

Para esto, vayan a <https://developers.google.com/gmail/api/quickstart/nodejs> y realice el paso 1.



Ahora, que ya tienen el proyecto y las credenciales, necesitan obtener un token que puedan utilizar para hacer requests a la API de GMail:

1. Copie en el directorio **gmail-tools/get-token** el archivo **credentials.json**.
2. Dentro del directorio **gmail-tools/get-token**, ejecute “**npm install**”.
3. Ejecute “**node getOAuthToken.js**”.
4. Haga click en la Url que aparece en consola. (deberá logearse en gmail y darle permisos a la aplicación)
5. Una vez que le dio permisos obtendrá un código, el cual tiene que pegarlo en la consola donde ejecutó **getOAuthToken.js**.
6. Listo!

Si todo salió bien, su token se habrá generado en el archivo **token.json**. Utilizando este token usted podrá hacer request a la API de GMail. Para eso vamos a usar una librería provista por google que nos permitirá realizar requests de una forma más amigable. En el directorio **gmail-tools/send-mail-example** encontrará un ejemplo de cómo enviar dicho email.

Para enviar un email en su aplicación node, deberá:

* Agregar como dependencia a su aplicación la librería googleapis:
  + **npm install --save googleapis**
* Copiar los archivos **token.json** y **credentials.json**.
* Copiar el módulo **gmailClient.js** ubicado en **gmail-tools/send-mail-example**.
* Utilizar la función **getGmailClient** exportada por el módulo, para obtener el cliente de la API de GMail.

## Fase 2 Servicio de Logging

Toda las aplicaciones necesitan de un registro de sucesos (log), ya sea para debugging o auditoría. En general los logs no se suelen ver, pero son muy valiosos para intentar entender y corregir errores cuando es necesario. Es por eso que deberá implementar un servicio de logging que guarde los eventos localmente y también los almacene en la nube (usando Logly).

Usted deberá implementar un servicio con node+express con las siguientes características:

* El log debe poder activarse y desactivarse. Durante el visado deberá mostrar como activa y desactiva este servicio.
* El log deber tener dos partes, el mensaje en si mismo y el nivel del mensaje (error, warning, info, debug)
* El log se deberá guardar localmente en un archivo de texto.
* Cada entrada del log se enviará a Logly

Para enviar logs a loggly deberá registrarse en <https://www.loggly.com/signup/>. Una vez registrado, siga los pasos en <https://www.loggly.com/docs/node-js-logs-2/> para poder enviar sus logs.

Este servicio será utilizado por UNQfy para volcar información sobre la actividad de UNQfy. Es decir, se desea que ciertos eventos que ocurren en el sistema sean logueados.

**Deben notificarse todas las acciones que alteran los datos en el modelo, es decir las altas y bajas de temas, albumes y artistas. No deben notificarse las búsquedas o lecturas**

## Fase 3 Monitoreo y aviso en Slack.

Toda aplicación en la nube debe ser monitoreada. En este caso deberá implementar un servicio que sirve para saber si UNQfy, el servicio de notificación y el servicio de logging están funcionando o no. Este servicio deberá informar en slack si alguno de los servicios ha dejado de funcionar. De esta manera el equipo de desarrollo puede monitorear la actividad de UNQfy con comodidad.

El monitor debe estar chequeando periódicamente el estado de los servicios y notificar vía slack en caso de falla de alguno de los mismos. También debe notificar si el servicio volvió a la normalidad,

Se espera que en Slack aparezcan mensajes con la forma:

* “[HORA DEL INCIDENTE] El servicio X ha dejado de funcionar”
* “[HORA DEL INCIDENTE] El servicio X ha vuelto a la normalidad”

Para ello cada grupo dispondrá de una URL en slack la cual deberá enviar request POST al Endpoint documentado en <https://api.slack.com/methods/chat.postMessage>.

A continuación se muestra un un ejemplo mínimo utilizando curl:

**curl -X POST -H 'Content-type: application/json' --data '{"text":"Hello, World!"}' <URL\_PROVISTA\_POR\_LA\_CATEDRA>**

El servicio de Monitoreo debe ser independiente de UNQfy. Esto quiere decir que debe correr en una instancia de Node/Express independiente para luego poder ejecutarlo en su propio container.

Se desea que este servicio provea la siguiente funcionalidad. Ud. debe definir los endpoints correspondientes.

* Debe permitir saber si el estado de los servicios. Es decir, se le envía un request al servicio de monitoreo y este debe responder si los servicios se encuentra activos o no.
* Debe monitorear periódicamente el estado de los servicios y notificar a slack en caso de falla de alguno de los mismos.
* Debe poder desactivarse/activarse el monitoreo periódico.

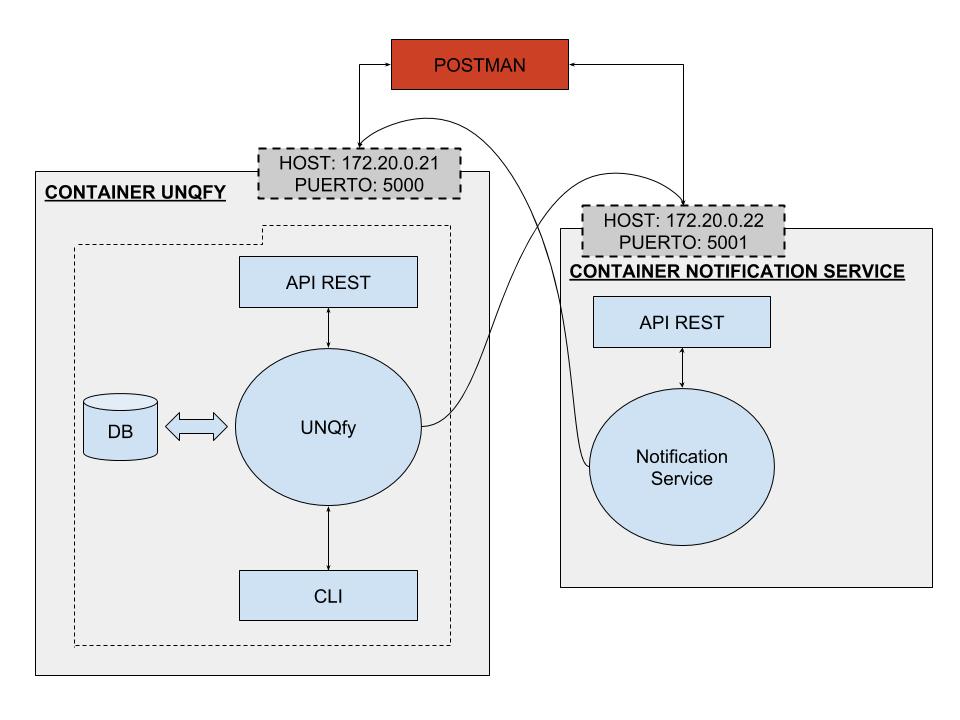
# Fase 4 - Corriendo los servicios en containers docker

Una vez que tenga las fases 1 y 2 terminada. Deberá correr los servicios en diferentes containers.

Para eso deberá crear 4 imágenes docker utilizando un Dockerfile para cada servicio y luego correrlas utilizando docker run. Para crear estas imágenes puede basarse en el dockerfile utilizado en la práctica 7, también recomendamos que vuelva a leer [Dockerizing a Node.js web app](https://nodejs.org/en/docs/guides/nodejs-docker-webapp/).

Dado que los servicios estarán aislados en sus propios containers, es necesario realizar la configuración correspondiente para que puedan comunicarse.

Por ejemplo si consideramos solamente UNQfy y el servicio de notificaciones (ud puede extrapolar esta explicación al caso del servicio de monitoreo y de logging) considere que es necesario decirle a UNQfy en que host y puerto estará corriendo el servicio de notificaciones y viceversa.



Para esto vamos a configurar ambos servicios con IPs y puertos fijos.

UNQfy correrá en 172.20.0.21:5000 y el servicio de notificaciones en 172.20.0.22:5001.

Para esto deberá:

* Crear una red en docker (ejecutar por única vez):
  + docker network create --subnet=172.20.0.0/16 unqfynet
* Al correr el container, especificar la IP y la red:
  + docker run --net unqfynet --ip 172.20.0.21 -p 5000:<dockerPort> <otrosArgs> ...
  + docker run --net unqfynet --ip 172.20.0.22 -p 5001:<dockerPor> <otrosArgs> …
* Luego, en UNQfy podrá acceder al servicio de notificaciones:  
  const NOTIFICATION\_BASEURL = ‘http://’ + 172.20.0.22+ ‘:’ + ‘5001’ + ‘/api’;  
  rp.get(NOTIFICATION\_BASEURL/notify).then( … )
* Y en el servicio de notificaciones podrá acceder a la API de UNQfy:  
  const UNQFY\_BASEURL = ‘http://’ + 172.20.0.21+ ‘:’ + ‘5000’ + ‘/api’;  
  rp.get(UNQFY\_BASEURL/artists).then( … )