Universitat de Girona
Escola Politècnica Superior

Projecte fi de grau

Estudi: Grau en Enginyeria Informàtica

Títol: Data pipeline de les comunicacions de Som Energia

Document: Memòria

Alumne: Pau Boix Tura

Tutor:Esteve del Acebo Peña
Departament: Informàtica, matemàtica aplicada i estadística
Àrea: Llenguatges I Sistemes Informàtics

Convocatòria (mes/any): Setembre 2022

Universitat de Girona
Escola Politècnica Superior

PROJECTE FI DE GRAU

Data pipeline de les comunicacions de Som Energia

Autor:
Pau BOIX TURA

Setembre 2022

Grau en Enginyeria Informàtica

Tutor:
Esteve DEL ACEBO PEÑA

Agraïments

Vull agrair a l'equip IT de Som Energia el seu suport i acompanyament durant la realització del projecte, especialment a en Roger Sanjaume i en Pol Monsó de l'equip de dades pel seu suport i per acompanyar-me des de l'inici al final revisant el codi, i ensenyant-me tot el necessari. També vull agrair molt especialment a en Joan Sarola per haver-me ajudat amb tots els problemes que hi havia a la infraestructura i a tot l'equip ERP pel seu suport. També vull agarair a en Francesc Casadellà, la Judith Frigolé i en Joan Basagaña la seva implicació amb el projecte.

Finalment, també vull agrair el suport i seguiment del projecte per part del meu tutor, Esteve del Acebo, al llarg del desenvolupament del treball.

Índex

CAPÍTOL 1

Introducció

1.1 Descripció de l'entorn

Som Energia és una cooperativa sense ànim de lucre de comercialització i generació d'energia verda amb més de 80.000 socis i 120.000 contractes que genera 24,6 GWh a l'any amb instal·lacions pròpies.

Som Energia es va crear a finals del 2010 i va començar a comercialitzar electricitat l'any 2011, des del inici també va iniciar projectes de generació renovable, entre els quals destaca el GenerationKWH que permet l'implantació d'energies renovables participades pels socis mitjançant aportacions com a alternativa a aquelles persones que no poden realitzar una instal·lació d'autoconsum. La seu sempre ha estat ubicada al Parc Científic i Tecnològic de la Universitat de Girona.

L'estructura empresarial de Som Energia està compost per quatre entitats principals:

El Consell Rector, format per persones sòcies, de manera voluntària i no remunerada que vetllen per la implementació de les directius d'actuació de la cooperativa.

Gerència, consistent en un conjunt de persones de l'equip de treball que gestionen la relació entre el Consell Rector i el dia a dia de l'Equip de Treball.

Equip de treball, integrat per treballadors assalariats de la cooperativa que realitzen les tasques del dia a dia. Actualment hi ha més de 100 treballadors.

Persones sòcies, que mitjançant votacions escullen les directrius d'actuació de la cooperativa. A més, cal destacar la figura dels grups locals, consistents en grups de persones sòcies que, segons el seu lloc geogràfic realitzen activitats i xerrades sobre la temàtica del model energètic actual, energies renovables, compromís social, i moltes d'altres temàtiques en concordança amb la cooperativa

1.2 Punt de partida

Quan em van proposar el projecte només s'havien fet alguns petits scripts per obtenir dades a Som Energia i encara no existia l'equip de dades. En el repo-

sitori de github només s'havien fet scripts quan havia sortit alguna necessitat específica des de l'equip tècnic d'obtenir algunes dades o indicadors.

Aquests scripts eren python pur i no estaven orquestrats per Airflow, sinó que hi havia un fitxer crontab que determinava cada quan s'havien d'executar aquests scripts.

Tot aquest programari es trobava allotjat al servidor de "Puppis", que també allotjava la base de dades Postgres amb algunes taules que guarden informació de diverses fonts, algunes taules que ja existien que seràn útils com la de bústies de Helpscout.

1.3 Justificació

Som Energia és la comercialitzadora millor valorada pels seus usuaris, [OCU 022] amb l'augment de feina dels últims mesos preocupa molt tot el que té a veure amb l'atenció al client. En el moment de començar el projecte preocupava especialment les trucades que es deixaven d'atendre i el temps de resposta al correus.

Aquest projecte però ve impulsat pel departament de comunicació, que notava que tenia la necessitat de tenir informació en alguna eina de business intelligence sobre l'impacte que té la feina que fan.

A part de tot el mencionat anteriorment, un dels 20 objectius per l'any 2022 per la cooperativa, és establir el model de com obtenir les dades per a poder obternir-ne indicadors, aquest objectiu es vol assolir amb el disseny i l'implementació de l'obtenció de dades d'aquest projecte.

1.4 Propòsit

El propòsit general d'aquest projecte és dotar de la informació necessaria a l'equip de comunicació per poder tenir una evolució històrica de les seves accions, poder preveure accions futures i a través d'això donar un valor afegit als altres equips de la coopeativa.

Per part de l'equip de dades el propòsit d'aquest projecte és començar a utilitzar Airflow per orquestrar els scripts d'obtenció de dades, i així tenir definit com s'utilitzarà.

1.5 Objectius

L'objectiu d'aquest projecte és construir els data pipelines que donin a l'equip de comunicació la informació de les fonts necessàries, transformar la informació i

visualitzar-la. D'aquesta forma poder avaluar millor la seva feina i millorar les comunicacions futures.

També es vol començar a utilitzar Apache Airflow en aquest projecte per després poder-se convertir en l'orquestrador de tots els processos ELT a Som Energia.

S'espera poder donar la informació necessaria per poder millorar l'atenció al client per poder tenir una resposta més ràpida, millorar la resposta a situacions de crisi on l'atenció al client es veu superada, també tenir un històric d'aquestes dades.

1.6 Justificació de canvis respecte al document

No he cregut convenient fer canvis pel que fa als apartats que es suggereixen al document, excepte la supressió de l'apartat d'implementació i proves perquè s'ha inclòs el contingut dins l'apartat d'implantació i resultats.

CAPÍTOL 2

Viabilitat

2.1 Viabilitat econòmica

Per calcular el cost salarial, tindré en compte el temps destinat a la recollida de requeriments, el temps de desenvolupament, i el temps de gestió del projecte.

No es tindrà en compte el cost dels proveidors de software d'on s'han extret dades ja que aquests ja es pagaven abans del desenvolupament del projecte. Tampoc considero necessari tenir en compte l'ordinador portàtil ni perifèrics utilitzats per desenvolupar el projecte ja que ja es disponia prèviament d'ells i són necessaris per poder dur a terme la meva jornada laboral amb normalitat.

Pel que fa als servidors utilitzats durant el projecte, ja existien o no han requerit canviar el contracte amb el nostre proveïdor.

Es calcula que pel projecte es destinaran 360 hores al projecte entre la presa de requeriments, l'anàlisi i el desenvolupament. També es calcula que es destinaran 25 hores a acompanyar-me en el projecte l'equip de dades en tot el procés i també el de sistemes en cas de que apareguin incidències durant el desenvolupament.

Concepte	**Temps**	**Cost unitari**	**Total**
Hores programador júnior	360h	10€/h	3600€
Hores programador sénior	25h	17€/h	425€
Total			**4025€**

Per tant és un cost assumible per la coopeativa.

2.2 Viabilitat en relació als recursos humans

Aquest projecte sabia que era possible ja que s'havien fet desenvolupaments similars a Som Energia anteriorment, també sabia que podia comptar amb l'equip de sistemes per donar-me suport amb les tasques que implicaven servidors o tecnologies més del seu àmbit com per exemple Docker.

També comptava amb l'equip de dades per ajudar a prendre decisions relacionades amb el projecte i problemes que sorgissin durant el desenvolupament ja que ells si que tenien expèriencia fent altres pipelines.

El impacte que tindrà el projecte per aquests treballadors serà ínfim i no repercutirà en les seves tasques diàries.

2.3 Viabilitat tecnològica

Com que tot el software utilitzat és de codi lliure o obert la seva documentació també és fàcilment accessible. També es disposa d'una bona base de coneixements sobre Python.

Pel que fa al hardware on s'allotja el programari, com que depèn d'un proveïdor també podem considerar que és viable.

Per la redacció de la memòria també s'ha utilitzat software lluire, utiltizant LaTeX i Emacs com a editor de text.

2.4 Viabilitat legal

Aquests projecte s'ha fet seguint el reglament general de protecció de dades. A més, Per treballar a Som Energia és necessari completar un curs sobre com complir amb el RGPD.

Durant l'elaboració del projecte mai s'ha tractat amb dades de categories especials i el tractament de dades s'ha fet seguint els principis del reglament. El tractament també ve consentit per la firma que es trobar en tots els correus de Som Energia:

T'informem que les teves dades identificatives i les contingudes en els correus electrònics i fitxers adjunts poden ser incorporades a les nostres bases de dades amb la finalitat de mantenir relacions professionals i/o comercials i, que seran conservades mentre es mantingui la relació. Si ho desitges, pots exercir el teu dret a accedir, rectificar i suprimir les teves dades i d'altres reconeguts normativament dirigint-te al correu emissor o a somenergia@delegado-datos.com

2.5 Valoració global de l'estudi de viabilitat

Després del mencionat en cadascun dels apartats podem afirmar que el projecte és viable.

CAPÍTOL 3

Metodologia

3.1 Metodologia de Planificació

El projecte s'ha fet seguint les meteologies que s'utilitzen actualment a Som Energia com són Scrum i eXtremme programming, el projecte s'ha fet seguint sprints de 3 setmanes que coincidien amb els de la resta de la cooperativa.

El rol d'Scrum Master i el de Product Owner els durà a terme la persona de referència del equip de dades. Es realitzaran reunions de planificació del sprint i retrospectiva del sprint però no Daily Standups ja que no es treballarà diariament en el projecte, sinó que es realitzaran només reunions quan es cregui adient per comentar temes del desenvolupament. El rol de stakeholder el durà a terme l'altre membre de l'equip de dades per la part més d'infraestructura i per la part dels resultats sobre comunicacions els stakeholders seràn els membres de l'equip de comunicació i l'equip d'atenció i suport.

3.2 Metodologia de desenvolupament

Per desenvolupar el projecte s'utilitzarà el sistema de control de versions git, i el repositori estarà allotjat a GitHub. Al repositori hi ha la branca main que és la que es troba en el servidor, per desenvolupar però es creen branques partint de main per poder realitzar una pull request a GitHub on els companys poden fer la revisió del codi i aprovar els canvis.

Sempre que sigui possible es realitzaran testos ja que seguirem Test-Driven Development. En aquests projecte només s'han realitzat per les tasques de transformació de dades.

CAPÍTOL 4

Planificació

Primerament, la intenció era entregar el projecte al Juny, però veient com s'acostava la data i encara estava lluny d'assolir els objectius vaig esperar fins el setembre.

La planificació del projecte està fortament lligada a la metodologia de desenvolupament, que és una adaptació de SCRUM Agile. Amb aquesta metodologia, en cada sprint es realitzen tasques de recull de requisits, d'anàlisi, d'implementació i de testeig.

Per a la realització del projecte es van planificar inicialment 4 Sprints, que després es van ampliar a 7 quan es va decidir entregar el projecte el setembre.

4.1 Sprints

4.1.1 Sprint 1

- Calendarització
- Anàlisi de requeriments
- Preparació entorn de treball
- Anàlisi de les fonts de dades d'on estem impactant les comunicacions

4.1.2 Sprint 2

- Presa de requeriments amb en Francesc
- Script d'Obtenció converses Helpscout
- Posada en marxa Airflow

4.1.3 Sprint 3

- Adaptació del script d'obtenció de converses a Airflow
- Proves Portainer
- Tasques comunes DAGs

4.1.4 Sprint 4

- Posada en producció d'Airflow amb Portainer
- Fer la imatge docker de somenergia-kpis
- Connectar mitjançant NFS Portainer
- Disseny de la transformació

4.1.5 Sprint 5

- Script de transformació
- Modelatge de les classes
- Formació SQLAlchemy ORM
- DAG transformació
- Tests script de transformació
- Script d'obtenció d'etiquetes

4.1.6 Sprint 6

- Formació Superset
- Desenvolupar Dashboard Superset
- Posada en producció DAG transformació i correció d'errors

4.1.7 Sprint 7

- Continuació: Desenvolupar dashboard Superset
- Presentació resultats
- Millorar dashboard amb el feedback del equip tècnic
- Millorar el projecte amb el feedback del tutor

CAPÍTOL 5

Marc de treball i conceptes previ

5.1 conceptes previs

5.1.1 Python

Tot el projecte està programat en Python, per tant per entendre el projecte es necessiten coneixements de programació i conèixer la sintàxi de Python. [Foundation 022b]

5.1.2 DAG

Un graf acíclic dirigit, DAG per les seves sigles en anglès, és el concepte central d'Airflow, organitzant les dependències i relacions entre les tasques a executar. [Airflow 022]

5.1.3 ORM

Un ORM (Object–relational mapping) és un model de programació que permet mapejar les estructures d'una base de dades relacional sobre una estructura lògica d'entitats amb l'objectiu de simplificar i accelerar el desenvolupament de les aplicacions.

Les estructures de la base de dades relacional queden vinculades amb les entitats lògiques o base de dades virtual definida a l'ORM, de manera que les accions CRUD (Create, Read, Update, Delete) a executar sobre la base de dades física es realitzen de manera indirecta per mitjà de l'ORM. La conseqüència més directa és la generació automàtica de codi SQL per fer les consultes i gestionar la persistència. [Muro 022]

5.1.4 Data Lake

Un data lake és un repositori d'emmagatzematge centralitzat que recull grans quantitats de dades de diferents fonts sense processar el format original. S'emmagatzemen fins que es necessitin per a les aplicacions d'anàlisi.

És a dir, mentre un data warehouse emmagatzema dades en taules i dimensions jeràrquiques, un data lake utilitza una arquitectura plana per emmagatzemar dades. [school 022]

5.1.5 ETL

Extract, transform, load. És un procés de tres fases on les dades s'extreuen, es transformen i es carreguen en una base de dades de sortida. Les dades es poden recopilar d'una o més fonts i també es poden enviar a una o més destinacions.

5.1.6 ELT

Extract, load, transform. És una alternativa per ETL utilitzada amb implementacions de data lake. A diferència dels processos ETL, en els models ELT les dades no es transformen en entrar al data lake, sinó que s'emmagatzemen en el seu format original. [Wikipedia 022a]

CAPÍTOL 6

Requisits del sistema

6.1 Requeriments funcionals

Per tal de prendre els requisits primer es va fer una reunió amb en Francesc Casadellà del equip de comunicació, on van sorgir una sèrie de requeriments enfocats a la necessitat del equip de comunicació de poder avaluar la seva feina. Al cap d'uns dies quan l'equip d'atenció i suport va saber que s'estava començant aquest projecte, al tractar-se de temes que també eren del seu interès, van col·laborar afegint requeriments i ajudant a definir el projecte.

També vaig parlar amb l'equip de dades per definir què havia de complir l'infraestructura per poder dur a terme el projecte.

6.1.1 Requeriments equip tècnic

- Poder visualitzar quants correus arriben a Som Energia
- Poder visualitzar quants correus es responen a Som Energia
- Poder conèixer el temps de resposta als correus
- Poder visualitzar les trucades que arriben a Som Energia
- Poder visualitzar les trucades que s'atenen a Som Energia
- Tenir un històric de les comunicacions
- Poder preveure l'impacte quan es fa una nova acció
- Quantificar el retorn en funció del públic
- millorar les respostes a les crisis
- tenir l'evolució històrica
- donar un valor afegit als equips
- quantificar l'impacte de les comunicacions

6.1.2 Requeriments equip dades

- Posar en funcionament un orquestrador de pipelines
- Que la infraestructura sigui reutilitzable per altres projectes

6.2 Requeriments no funcionals

Els requisits no funcionals de l'aplicació són:

- Al haver de ser reutilitzable ha de poder suportar entorns virtuals de python
- Al poder-se utilitzar en diferents projectes també diferents versions de python
- Al utiltizar-se per volums massius de dades ha de tenir un gran rendiment

Com que el projecte es desenvolupa a Som Energia hi ha un conjunt de decisions que no he pogut prendre jo. Per tant, aquestes decisions són, en realitat, requeriments no funcionals que m'han vingut marcats.

- codi lliure
- l'allotjament ha de ser en els servidors ja contractats

CAPÍTOL 7

Estudi i decisions

7.1 Apache Airflow

Apache airflow va néixer l'any 2014 a Airbnb com a resposta per solucionar els fluxes de feina cada cop més complexos que hi havia a la companyia. Airflow els hi permetia programar les tasques d'una manera sistemàtica i monitoritzar els resultats a través de la interfície d'usuari, el projecte és de codi obert i actualment està mantingut per l'Apache Software Foundation. [Wikipedia 022b]

Airflow està programat en python i permet programar pipelines amb codi python. [Apache 022]

7.1.1 Perquè Airflow

Pels objectius del projecte Airflow clarament era la millor solució de totes les disponibles. És fàcil i està ben documentat com migrar de cron a Airflow. Tot i que hi ha alternatives també basades en Python com per exemple Luigi si entrem a Github podem veure que clarament el seu desenvolupament no és tant actiu ni té tanta comunitat com Airflow. Una altra alternativa podria ser Prefect però el fet de no ser FOSS ja fa molt difícil la seva adopció per una organització com Som Energia. Si el motiu anterior ja era molt important, el seu preu fa completament inviable la seva utilització.

7.2 PostgreSQL

PostgreSQL és una base de dades relacional lliure i de codi obert que amplia el llenguatge SQL, combinat amb moltes funcions que emmagatzemen i escalen de manera segura les càrregues de treball de dades més complicades. [Group 022]

Un dels motius per utilitzar PostgreSQL és l'existència de Timescale, una extensió de PostgreSQL creada per time-series. Timescale proporciona un rendiment entre 10 i 100 cops superior a PostgreSQL o altres bases de dades noSQL. [TimescaleDocs 022]

Actualment tant PostgreSQL com Timescale ja s'utilitzen en altres projectes de SomE nergia i al equip de sistemes ja tenen coneixements de DBA per tant a més de ser la opció més potent a nivell de rendiment també era la més coneguda.

7.3 SQLAlchemy

SQLAlchemy és un conjunt d'eines SQL i ORM de Python, SQLAlchemy es basa en dos components i el CORE i el ORM, el Core és un conjunt d'eines que abstreuen el comportament SQL a Python, i un ORM construit a partir del Core. Tot i haver-hi altres alternatives com Pony i Psycopg, es va optar per SQLAlchemy ja que té totes les funcionalitats que necessitava, disposa d'una molt bona documentació i permetia concentrar tota la interacció amb la base de dades en un sol paquet.

7.4 Docker

La tecnologia Docker utilitza el nucli de Linux i les seves funcions, per dividir els processos i executar-los de manera independent. El propòsit dels contenidors és executar diversos processos i aplicacions per separat perquè es pugui aprofitar millor la infraestructura i alhora conservar la seguretat que s'obtindria amb els sistemes individuals.

Les eines de contenidors, com ara Docker, proporcionen un model d'implementació basat en imatges, que permet compartir una aplicació o un conjunt de serveis amb totes les dependències en diversos entorns. Docker també automatitza la implementació de les aplicacions (o els conjunts de processos que les constitueixen) a l'entorn de contenidors.

Docker és la tecnologia de contenidor més extesa tot i que la solució nativa de Linux és LXC. Al ser Docker la més extesa fa que Airflow ja incorpori la integració amb Docker a diferència de LXC, tot i que LXC no conta amb una gran comunitat, el que també és un problema comparat amb Docker. [RedHat 022]

7.5 Docker Compose

Compose és una eina per definir i executar aplicacions Docker de diversos contenidors. Amb Compose, s'utilitza un fitxer YAML per configurar els serveis de l'aplicació. Aleshores, amb una única comanda, es creen i inicien tots els serveis des de la configuració.

Al igual que en punt anterior Airflow ja té el seu fitxer YAML, que es pot adaptar fàcilment a les necessitats del projecte. Al haver-hi aquest gran suport va resultar més fàcil utilitzar Compose que instal·lar-lo al sistema. [Inc. 022a]

7.6 Wget

GNU Wget és un paquet de programari lliure per recuperar fitxers mitjançant HTTP, HTTPS, FTP i FTPS, els protocols d'Internet més utilitzats. També es va estudiar curl però al afegir més funcionalitats també era més pesat, el que el feia pitjor pel que es necessitava. [Foundation 022a]

7.7 Portainer

Portainer és una eina de gestió completa per a Docker. S'executa localment, oferint als desenvolupadors una interfície d'usuari rica per crear i publicar imatges de contenidors, desplegar i gestionar aplicacions, aprofitar la persistència de les dades i escalar horitzontalment les seves aplicacions.

Una vegada una aplicació s'ha desplegat en un contenidor, Portainer facilita als usuaris protegir, supervisar i mesurar el rendiment de la plataforma. L'eina nega la necessitat que els desenvolupadors aprenguin Infraestructura com a codi i els facilita la maximització de la seva eficiència. [Inc. 022b]

7.8 Apache Superset

Apache Superset és una plataforma moderna d'exploració i visualització de dades, Superset és ràpid, lleuger, intuïtiu i està carregat d'opcions que faciliten als usuaris de tots els conjunts d'habilitats explorar i visualitzar les seves dades, des de simples gràfics de línies fins a gràfics geoespacials molt detallats. [Foundation 022c]

A Som Energia també s'utilitza Redash, que també és de codi obert. M'he decantat per Superset ja que disposa de més opcions pel que fa a visualització de dades.

7.9 Trello

Com a eina per gestionar el projecte he optat per Trello ja que és amb la que estic més familiaritzat al ser la que s'utilitza habitualment a Som Energia. A

Trelllo les tasques són targetes, que es van movent de llista depenent del si ja estan fetes, en desenvolupament, o encara no s'han començat. [Joiner 022]

CAPÍTOL 8

Anàlisi i disseny del sistema

8.1 Anàlisi

8.1.1 Anàlisi de fonts dades

En l'anàlisi primer es van analitzar les fonts de dades. Actualment els canals per comunicar-se amb Som Energia són el telèfon, en horari de dilluns a divendres de 8h a 14h. També es pot a través de correu electrònic, enviant un correu a una de les diferents adreces que té la cooperativa. Al ser presents a les xarxes socials també rebem comentaris en aquelles on som presents. I també de forma esporàdica hi ha persones que ens visiten les oficines al Parc Científic i Tecnològic de la UdG.

Les fonts de dades que tenim d'aquests canals són fitxers.csv que ens proporciona el nostre proveïdor de la centraleta telefònica, l'API de helpscout pels correus electrònics, i l'API de twitter per les xarxes socials.

Cal destacar que al cap de poc de començar el projecte la cooperativa va decidir canviar el proveïdor de la centraleta telefònica, els fitxers que teniem del proveïdor original, ja eren bastant complexos i es necessiten coneixements sobre telefonia per entendre'ls bé. Tampoc sabiem quines eines ens facilitaria el nou proveïdor per obtenir dades de la nova centraleta, i quin format tindrien les noves dades.

És per això que durant el desenvolupament del projecte es descarta desenvolupar el pipeline de les trucades ja que suposaria una quantitat de temps inassumible extreure les dades de les dues fonts i el procés de transformació també es complica bastant. A més per poder treballar correctament amb les trucades no és tan fàcil com els correus ja que previament seria necessari una formació sobre extensions telefònniques i altres temes de la centraleta.

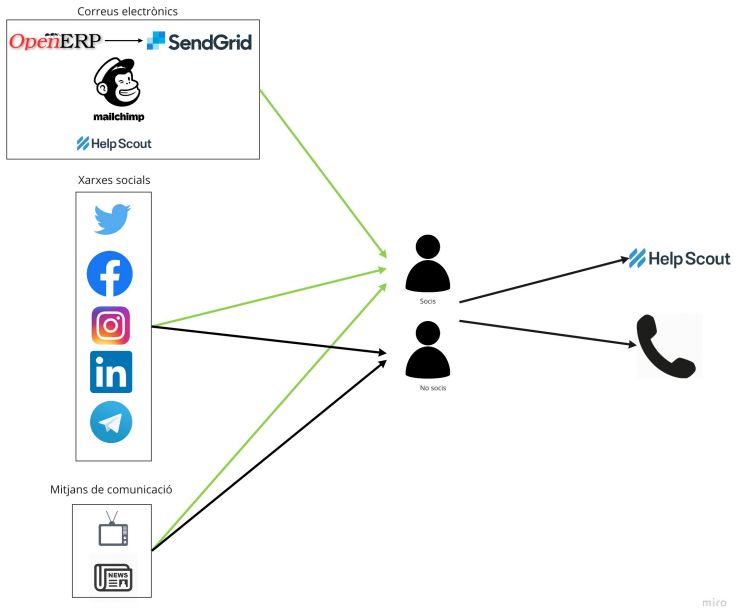


Figura 8.1: Esquema de les comunicacions de Som Energia

8.1.2 Anàlisi de processos

En l'anàlisi dels processos es detecta la necessitat d'un orquestrador per les nostres pipelines, ja siguin ETL o bé ELT, degut al gran nombre de scripts d'obtenció de dades que es preveuen.

8.2 Disseny

En el disseny es decideix utilitzar Airflow com a orquestrador de pipelines, i una arquitectura ETL per les dades.

Al tractar-se d'una arquitectura ETL tenim un data lake amb una taula anomenada *hs_conversation* que consta d'un identificador, un camp JSONB, que és JSON però PostgreSQL ho guarda en format binari per obtenir un millor rendiment a l'hora de fer consultes, on es guarda tot el contingut que retorna l'API,i tot i intentar ser fidel a la filosofia d'un data lake aquest model també incorpora els camps *data_interval_start* i *data_interval_end* amb índexs per cadascun d'ells per accelerar les tasques de transformació.

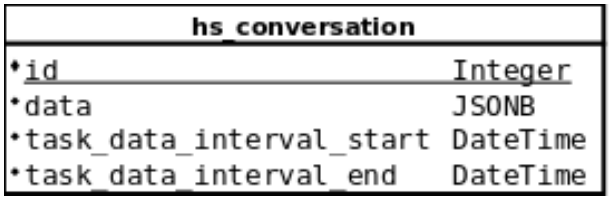


Figura 8.2: Model de dades del data lake

Pel data warehouse vaig decidir mantenir tots els camps originals que eren útils, a més del *data_interval_start* i *data_interval_end* amb índex que també tenia el data lake. El camp *tags* passa a ser una relació many to many amb la taula *hs_tag*, també existeix una relació 1 a 1 entre les converses i les bústies.

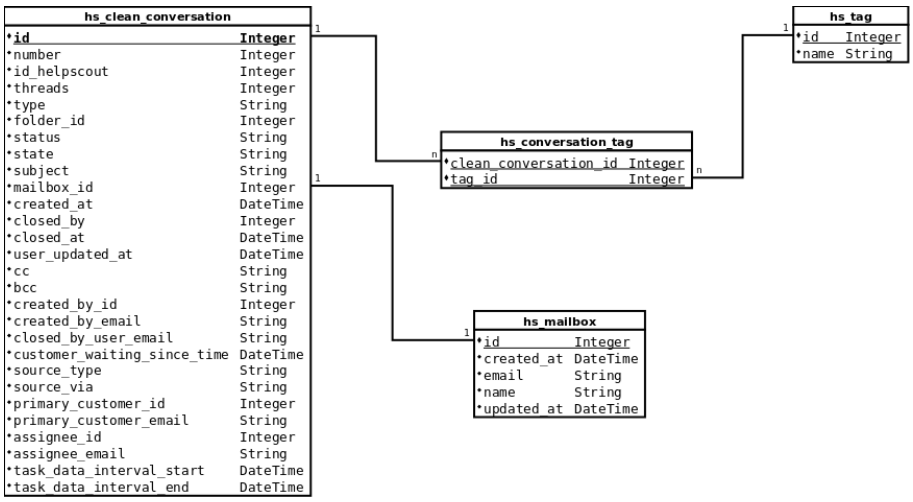


Figura 8.3: Model de dades del data warehouse

Així l'esquema global del projecte queda d'aquesta forma tenin en compte les fonts de dades, les tecnologies utilitzades, i les bases de dades utilitzades:

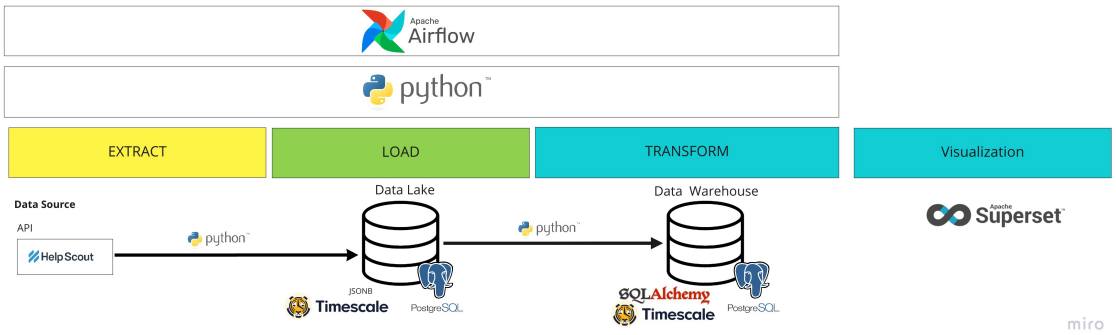


Figura 8.4: Esquema de la infraestrucutura del projecte

CAPÍTOL 9

Implantació i resultats

9.1 Procés de desenvolupament

El primer que vam implementar va ser el script per obtenir els correus mitjançant l'API de Helpscout. Aquest script busca els correus tancats ja que buscàvem el temps de resposta dels coreus.

Un cop fet el script vam decidir que utilitzaríem airflow per primera vegada i vaig investigar els DAGs per poder programar-ne un pel script.

De forma paral·lela també vaig està treballant en el script per obtenir les dades del twitter de Som energia.

Al intentar posar el script a Airflow vam haver d'adaptar-lo bastant, en primer lloc el script havia de ser idempotent per tal de poder ser reexecutat sense alterar el resultat, per això agafàvem només els que s'havien tancat entre dues dates, i també passàvem com a arguments els secrets de l'API de Helpscout i Postgres. [Astronomer 022a]

9.1.1 Desenvolupament de la infraestructura

En primer lloc vaig provar el PythonOperator amb èxit però si que vaig identificar el possible conflicte de dependències entre projectes si no utilitzavem cap virtualenv. Per sort també existia el PythonVirtualenvOperator que permetia executar Python dins d'un virtualenv, passant-li un fitxer amb els paquets que ha d'instal·lar. Però un cop vaig consultar la documentació i vaig descobrir que cada cop que s'executava creava i destruia el vritulenv em va generar un gran preocupació pel rendiment. [Astronomer 022b]

Després de consultar les meves preocupacions amb l'equip de dades i de sistemes vam acordar provar d'executar els nostres scripts dins d'un contenidor de Docker, i aquests contenidors s'executarien en un servidor de Portainer. També vam decidir passar de la instal·lació que teniem en aquell moment en el servidor a fer-la mitjançant docker-compose. Això és degut a que la instal·lació original havia generat alguns directoris duplicats, i al utilitzar Airflow mitjançant docker-compose en local feia que l'entorn de producció passés a ser més similar al local.

Al tenir Airflow mitjançant docker-compose també feia necessaria una eina com Portainer ja que sinó al socket docker.sock del servidor se li haurien d'haver

canviat els permisos d'execució, cosa que em van desaconsellar els administradors de sistemes per temes de seguretat.

Aquest plantejament solucionava el problema del rendiment i a la vegada en generava un altre, com desplegar el codi. La millor opció va ser generar la imatge amb només Python i el requeriments instalats i a través de NFS carregar un volum amb el codi dins del contenidor. Tot això va fer canviar també el DAG de forma que puguessim aconseguir de manera automàtica que s'actualitzes el codi i les imatges si hi havien canvis.

Al inici del pipeline s'executa la tasca de comprovar que existeix el repositori, aquesta és un BranchPythonOperator, ja que depenent del que retorni l'execució del codi seguirem una branca o una altra.

En cas de que no existeixi el repositori s'executarà la tasca de clonar el repositori, aquesta és un PythonOperator que mitjançant la llibreria paramiko ens connectem al servidor on s'esta executant Airflow i executem mitjançant paramiko el git clone del repositori. En aquest pas també es va estudiar l'us del SSHOperator, però en realitat també acabava utilitzant paramiko i no oferia tantes opcions com utilitzant paramiko directament. Es va optar per connectar-se mitjançant SSH al mateix servidor ja que dins del contenidor on s'executa Airflow no hi ha git instal·lat, i en cas de voler instal·lar-lo s'haurien de fer algunes modificacions importants el fitxer de configuració yaml que complicarien molt el seu manteniment.

En cas de que el repositori ja estigui clonat s'executarà la tasca de fer el git pull, al igual que en la tasca de clonar ens connectem mitjançant paramiko al servidor i s'executa un git pull, aquí es mira el text que retorna la comanda i depenent del que retorna al ser un BranchPythonOperator es procedeix a borrar l'imatge o executar el codi.

La tasca de borrar la imatge és un PythonOperator que utilitza la llibreria requests per fer una crida a la API de portainer per tal de borrar l'imatge actual.

La tasca per construir la image també és un PythonOperator que mitajançant la llibreria requests fa una crida a la API de portainer perquè contrueixi la imatge i li posi l'etiqueta corresponent a partir d'un Dockerfile allotjat al nostre repositori.

La imatge que he construit parteix de python 3.8 en la versió slim-buster ja que tot i ser la versió slim ja contenia tot el necessari per executar els programes. També s'instala wget per aconseguir el fitxer de requeriments que es troba a Github, i s'instalen mitjançant pip.

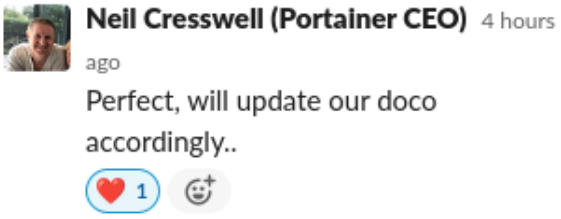


Figura 9.1: Resposta del CEO de Portainer després de comentar-li alguns problemes de la seva API per Slack.

Finalment cada DAG té un DockerOperator per tal d'executar el script de Python dins d'un contenidor que conté el codi. El funcionament de tot el procés en forma de graf es pot apreciar a les figures 9.2 i 9.3.

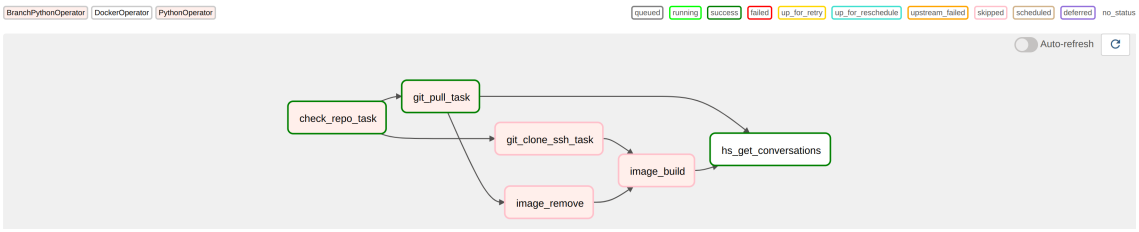


Figura 9.2: Graf quan no hi ha canvis

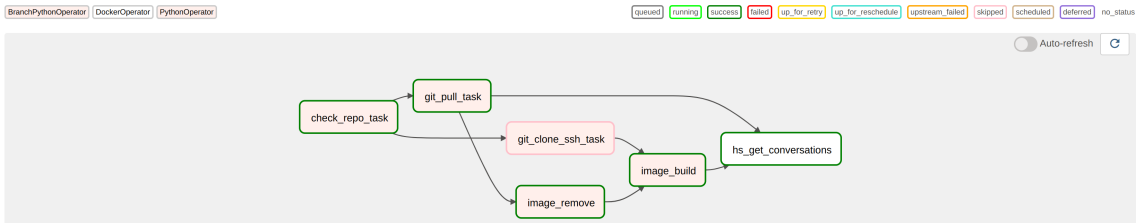


Figura 9.3: Graf quan hi ha canvis

Per carregar el codi dins del contenidor hi ha hagut dues aproximacions, la primera aproximació va ser utilitzar la funcionalitat de volums NFS de portainer, però després d'actualitzar-lo a una versió més nova va deixar de funcionar.

Llavors vaig decidir passar a declarar els volums en el propi DAG ja que el DockerOperator també ho suportava, i tampoc presenta cap desventatja respecte el que s'havia fet originalment.

S'ha intentat fer tot el màxim de genèric possible utilitzant les variables d'Airflow que s'accedeixen dins del codi mitjaçant JINJA templating o amb import Variable i fent un get. Això permetrà en cas que sigui necessari canviar el servidor on s'executen els contenidors o el propi servidor on està allotjat Airflow simplement canviant la variable que conté l'adreça dels servidors.

Al estructurar tot el DAG amb tasques perquè el sistema funcioni es necessiten dos enllaços simbòlics al directori /opt/airflow/dags, un apuntant al directori de dags i un altre al directori de tasques del repositori tal i com es veu a la figura 9.4.

```
airflow@scheat:/opt/airflow/dags$ ll
total 8
drwxrwxr-x 2 airflow root    4096 May 24 12:59 ./
drwxr-xr-x 7 airflow airflow 4096 May 25 11:09 ../
lrwxrwxrwx 1 airflow root      39 May 24 12:59 kpis -> /opt/airflow/repos/somenergia-kpis/dags/
lrwxrwxrwx 1 airflow root      45 May 23 17:37 kpis_tasks -> /opt/airflow/repos/somenergia-kpis/dags/tasks/
```

Figura 9.4: Soft links necessaris pel funcionament

9.1.2 Obtenció

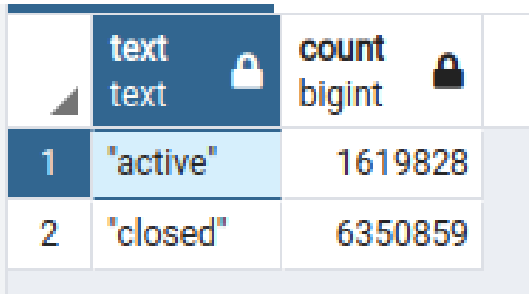
Un cop havia pogut fer proves i l'infraestructura funcionava vaig activar el DAG d'obtenció amb data d'inici al 20 de Març del 2020 i la variable catchup a Cert, el que implicava capturar-los tots des d'aquell moment. Es va triar aquesta data ja que representava al voltant d'uns dos anys en aquell moment i havent fet la prova amb només un mes, si no hi havia un gran canvi, era un volum de dades assumible.

Els correus obtinguts es guarden en format JSONB en una base de dades postgres, acompanyat d'un id, i el *data_interval_start* i *data_interval_end* de la tasca menys 1 setmana, ja que considerem que els correus tancats de fa una setmana que encara no s'han reobert ja no es reobriran o es tractaran com un cas diferent.

El vaig deixar executant-se durant el cap setmana per tornar al dilluns sense cap fail, amb els correus dels últims dos anys a la nostra base de dades.

Un cop havien finalitzat totes les execucions planificades, des de la interfície web d'Airflow ja vaig poder observar com algunes execucions havien durat molt més que d'altres. Això pot semblar natural, ja que no totes les hores es tanquen la mateixa quantitat de correus però investigant una mica era la API de Helpscout que no estava retornant la informació que li estàvem demanant.

De les tasques d'obtenció de dades no s'han realitzat testos, només de les de transformació.



	text text	count bigint
1	"active"	1619828
2	"closed"	6350859

Figura 9.5: Count de l'estat dels correus al pgAdmin després de l'execució inicial.

9.1.3 Transformació

Un cop obtingudes les primeres dades havia de dissenyar com fer el procés de transformació. Després de mirar les opcions disponibles vaig acordar amb l'equip de dades provar d'utilitzar sqlalchemy ORM, ja que en l'extracció també utilitzavem sqlalchemy per interactuar amb la base de dades.

Utilitzar SQLAlchemy també permet realitzar testos més robustos que els que s'havien fet anteriorment per tasques similars utilitzant b2btest.

Per dur a terme aquest procés de tansformació vaig programar dos DAGs, un per obtenir les etiquetes, i un altre per realitzar la pròpia transformació. El DAG de les etiquetes era necessari ja que vaig decidir establir una relació many to many entre converses i tags.

El DAG de les etiquetes va funcionar perfectament, però en el cas de la transformació de correus ens vam trobar en problemes inesperats.

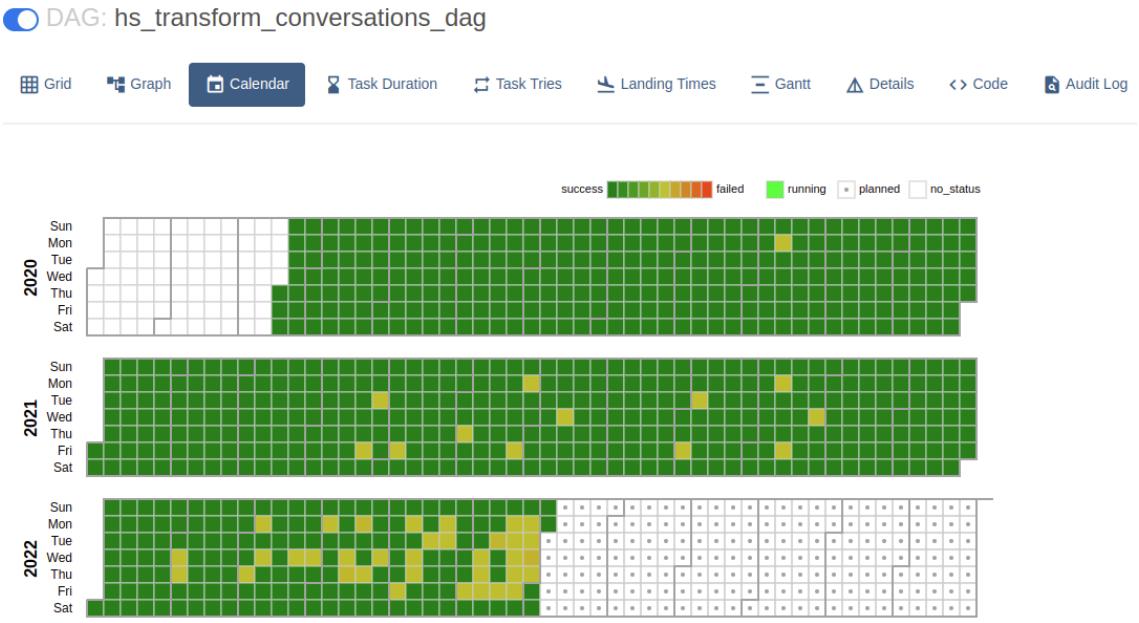


Figura 9.6: Resultat de l'execució del DAG de transformació.

Algunes de les dades obtingudes mancàven camps que donàvem per suposat que hi serien. Però gràcies a utilitzar Airflow vaig poder arreglar-ho molt fàcilment amb el següent procediment:

- Desactivar el DAG.
- Esborrar l'estat de les execucions en estat Failed.
- Corregir el script que es cridava.
- Reactivar el DAG.

Un cop reactivat es van reexecutar amb èxit totes les instàncies que havien fallat.

El DAG de trasnformació però té una estructura diferent a la resta de DAGs, ja que al dependre del DAG d'obtenció disposa d'un ExternalTaskSensor que fa que s'executi la tasca un cop ha acabat el DAG d'obtenció amb l'última tasca en estat success.

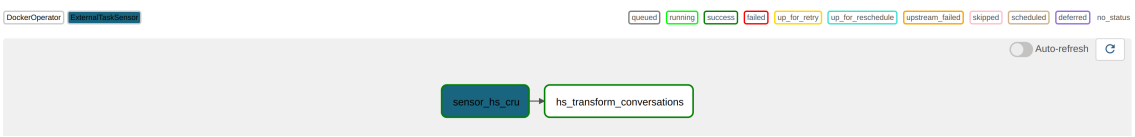


Figura 9.7: Graf del DAG de transformació.

De les tasques de transformació s'ha testejat tant les funcions que transformen les dades, com el correcte funcionament de les classes del ORM.

9.1.4 Visualització

Un cop ja tenia dades al data warehouse vaig començar amb la visualització. La base de dades Puppis ja estava connectada amb Superset, així que ja vaig poder començar a fer la visualització seguint la metodologia que requereix Superset. Primer amb el SQL editor es genera una query sobre la base de dades, i amb les dades que retorna creem una dataset. És a partir d'aquest dataset que ja es poden fer tots els gràfics que es creguin oportuns. Tal i com està fet Superset hi ha una pestanya Time, que ens permet escollir a partir de quina columna s'agafarà el temps de la visualització i també la seva granularitat. L'altre pestanya important és la de Query, on hem de definir quina funció apliquem a les dades.

Tots els gràfics que he realitzat els he anat afegint a un dashboard on també he realitzat l'anàlisi dels resultats obtinguts. D'aquesta forma les persones de l'equip tècninc de Som Energia que vulguin prendre decisions basades en les dades tenen tota la informació centralitzada en un mateix lloc.

Un dels problemes que presenta Superset és la d'escollir els colors, Superset simplement deixa triar una paleta de color i va canviant-los seguint la paleta, per tant cada cop que es recarrega el dashboard els gràfics es veuen de colors diferents.

9.2 Resultats

L'infraestructura d'Airflow va ser utilitzada per altres tasques de l'equip de dades, un cop vaig haver-lo utilitzat amb èxit també van començar a migrar alguns crons ja existents a Airflow copiant l'estructura del DAG.

Un cop ja havia fet proves amb el DAG de helpscout el vaig deixar executant-se durant el cap setmana per tornar al dilluns sense cap fail, amb els correus dels últims dos anys a la nostra base de dades. Tot i que no va funcionar tot a la primera el DAG de transformació actualment tampoc té cap execució amb fail.

De moment la infraestrucutura només ha fallat en una ocasió, el dia 26 de Juliol va haver-hi problemes amb Github, el que sumat a un control molt feble del que retorna el git pull, va provocar que la caiguda de Github es detectés com un canvi en el codi i es borrés la imatge. Al intentar crear la nova al intentar agafar el Dockerfile de GitHub no va tenir èxit i van començar a fallar totes les execucions.

9.2.1 Anàlisi de les comunicacions

9.2.1.1 Info

El resultat final de tota la feina feta ha estat un anàlisi dels correus a Som Energia a partir de les dades obtingudes a través de l'api de helpscout, les dades tractades van des del març del 2020 fins a dia d'avui. Parlaré de converses, que és com tracta helpscout els correus electrònics. Aquestes pertanyen a una bústia de correu (mailboxes) i les converses també es poden etiquetar (tags).

De les converses Helpscout disponem de l'assumpte, quan es va crear, quan es va tancar, quants correus van haver-hi en aquella conversa (threads), a quina mailbox s'ha tancat i quines tags té. Com que la majoria de la gent no està familiaritzada amb el concepte de converses em referiré a les converses com a correus excepte quan sigui necessari per entendre el gràfic. Al ser en realitat converses també vol dir que el temps de resposta en molts casos ens estem referint al temps necessari per tancar la conversa, necessitant múltiples correus.

Per començar primer mirarem en general tots els correus que s'han contestat a Som Energia dividits per bústia.

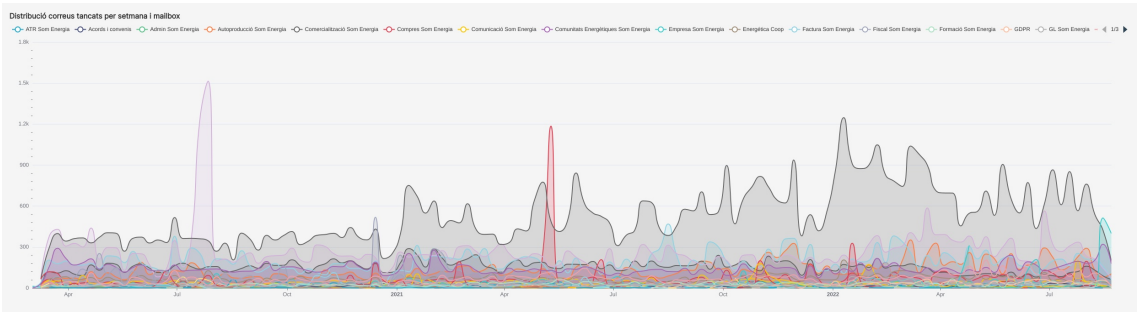


Figura 9.8: Distribució dels correus tancats per setmana i mailbox

En aquest primer gràfic de la figura 9.8 ja podem observar com clarament la bústia d'info és on més correus es tanquen setmanalment, només essent superat per gestió de cobraments i compres en dues ocasions.

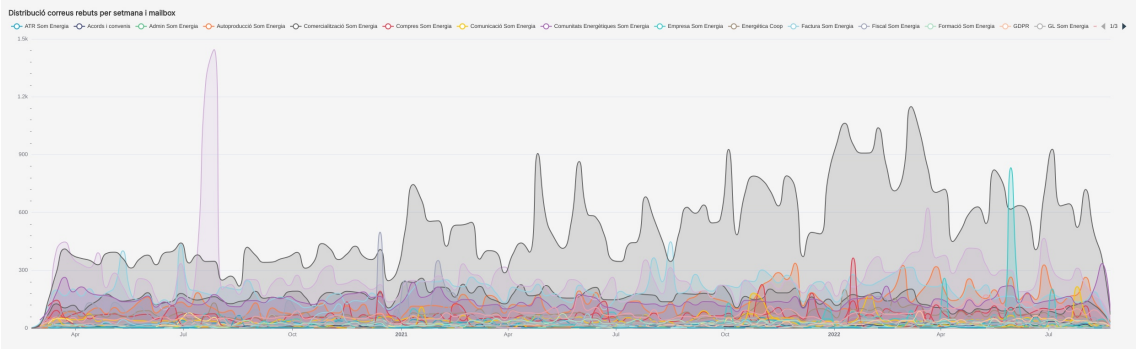


Figura 9.9: Distribució dels correus rebuts per setmana i mailbox

Mirant els correus per la seva data de creació a la figura 9.9 ja veiem però que el pic de compres es fruit d'haver deixat molts correus oberts i tancar-los de cop, no perquè hagin rebut molta feina de cop. També es confirma que info és on arriben la majoria de correus i que el pic de gestió de cobraments va ser real.

Un cop comprovat que la bústia d'info és la més important amb diferència, ara em centraré en la bústia d'info.

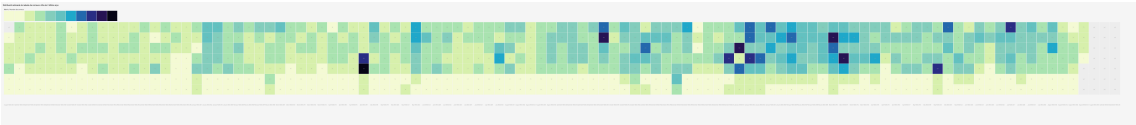


Figura 9.10: Distribució dels correus a info

Amb aquest calendari de la figura 9.10 ja s'aprecia com el nombre de correus que es reben a info és molt més alt últimament que fa dos anys. Es pot apreciar com progresivament cada cop és d'un color més blau, per tant rebem més correus, i als caps de setmana es reben molts pocs correus.

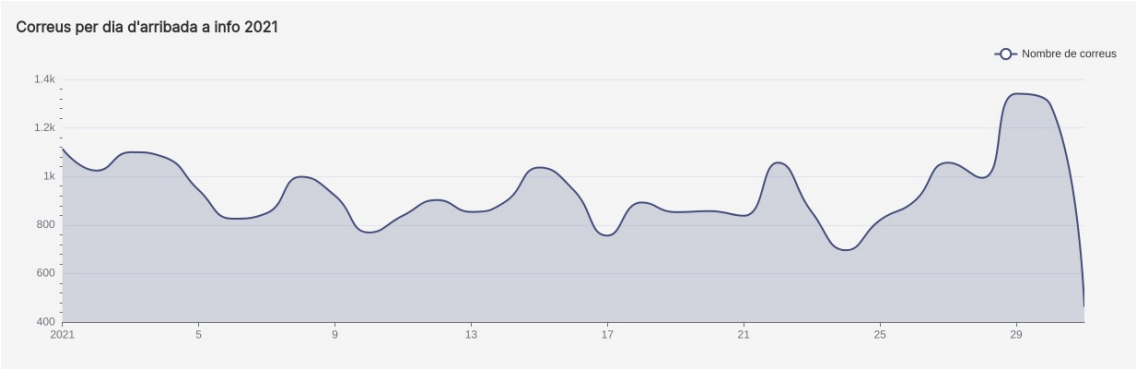


Figura 9.11: Distribució dels correus per dia d'arribada l'any 2021 a info

Quan intentem mirar l'estacionalitat a nivell de dia l'any 2021 a la figura 9.11 s'aprecia com els dies de final de mes són el que més feina hi ha, tenint també en compte que no tots els mesos tenen 31 dies, se situa al mateix nivell que la majoria de dies.

Figura 9.12: Correus rebuts a info i temps de resposta

Un cop identificat que info és la bústia més important passem a veure quina quantitat de correus es tanquen i quan es tarda de mitjana a tancar-los. S'observa a la figura 9.12 com el volum de correus ha anat augmentant al llarg del temps, i el temps de resposta també ho ha fet significativament, que es podria deure en part per la incorporació de noves persones, al haver de revisar els seus correus. També cal destacar que quan hi han pics de correus rebuts el temps de resposta baixa.

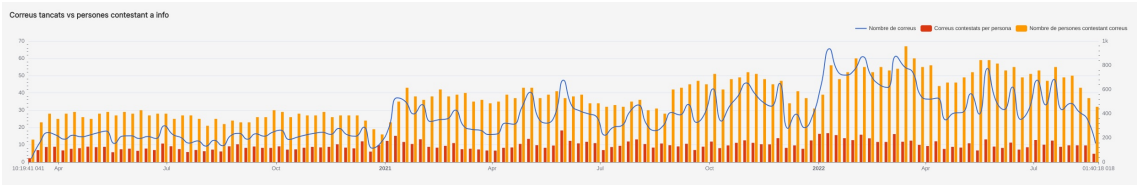


Figura 9.13: Correus rebuts a info, número de persones contestant i correus contestats per persona

Per fer el gràfic de la figura 9.13 només s'han tingut en compte els correus que estaven assignats a algú, però podem veure que tot i ser menys correus la tendencia és la mateixa que quan els tenim tots en compte, pel que fa al nombre de persones veiem que quan hi ha més correus també es posen més persones a contestar i el nombre de correus tancats per persona no canvia significativament.

9.2.1.2 Factura

Un dels temes que més preocupa internament a Som Energia és l'estat del equip de factura, ja que és l'equip que s'ha vist més afectat pels diferents BOEs introduint canvis a la factura de la llum degut al estat del mercat. En el moment d'iniciar el projecte un dels temes que més preocupava sobre l'atenció al client

era el temps que es tardava a contestar a les persones amb problemes en la seva facturació.

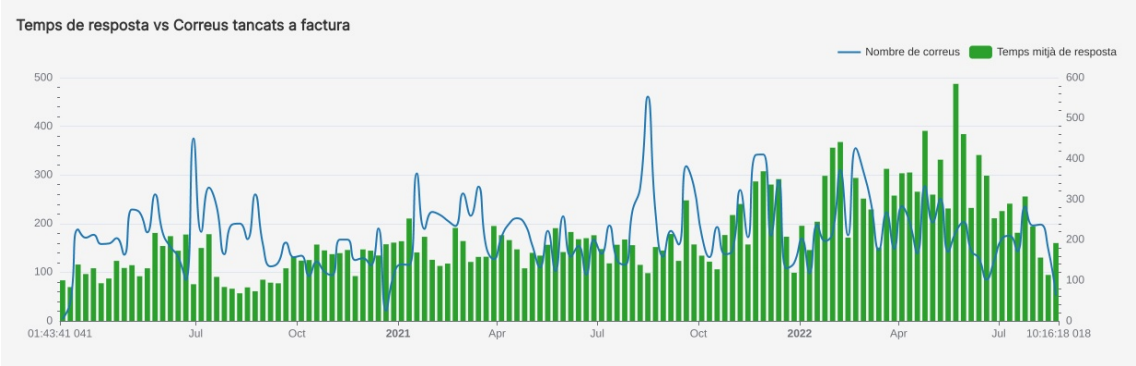


Figura 9.14: Correus rebuts a factura i temps de resposta

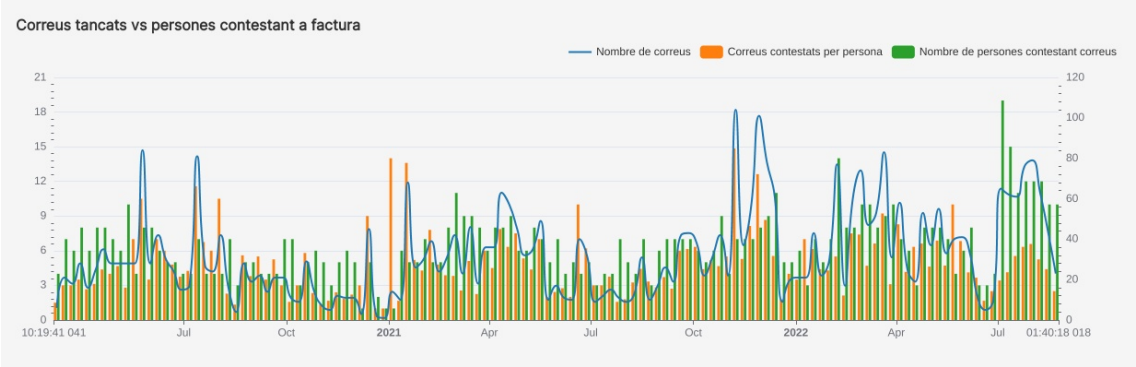


Figura 9.15: Correus rebuts a factura, número de persones contestant i correus contestats per persona

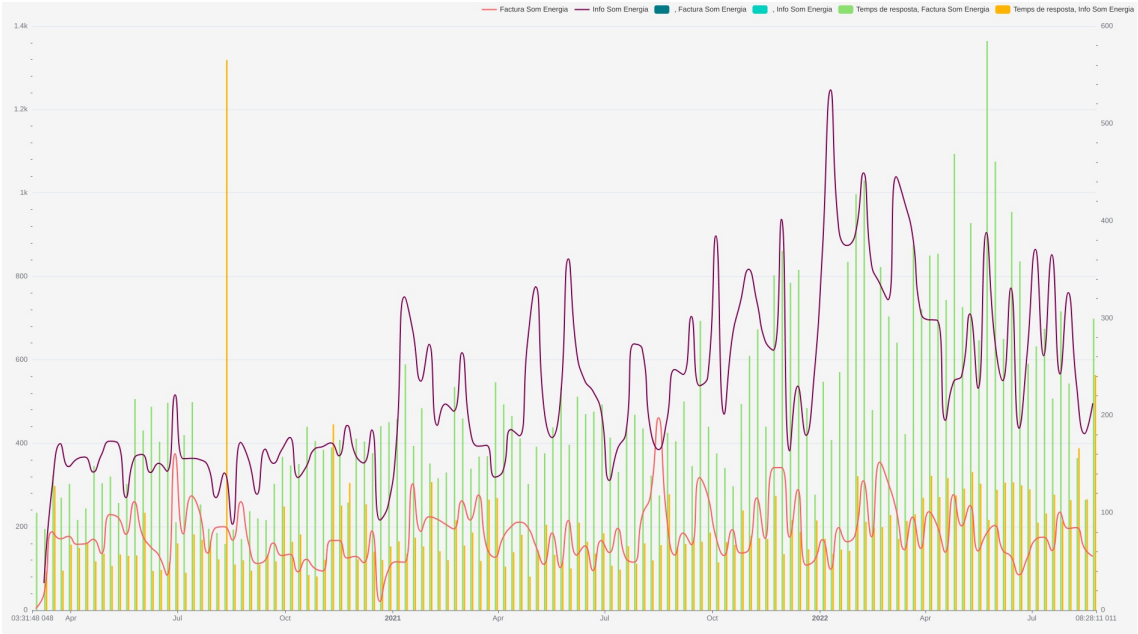


Figura 9.16: Correus tancats a info i factura i temps de resposta a info i factura

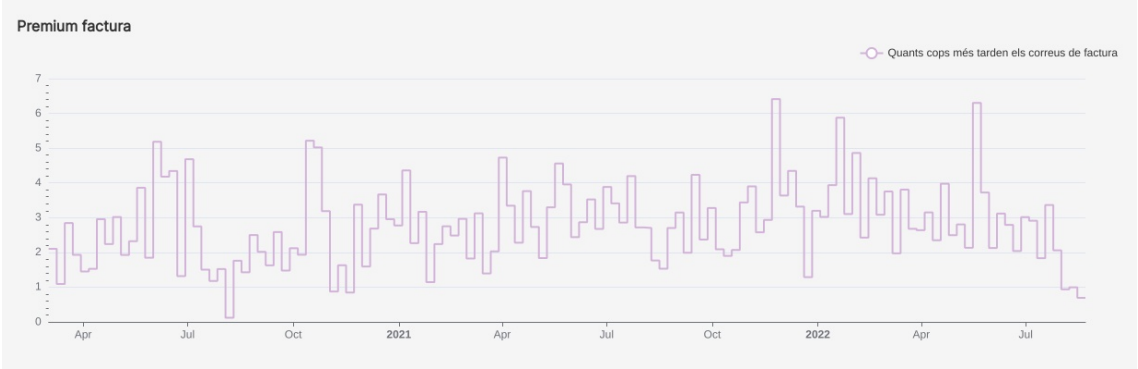


Figura 9.17: Premium Factura

Es pot veure com clarament hi ha menys gent contestant a factura que a info ja que a factura només contesta l'equip de factura. També es pot veure com es tarda més a contestar un correu de factura que d'info a la figura 9.17, a més aquesta diferència ha anat incrementant a mesura que passava el temps. Al contrari que a info, en el gràfic de la figura 9.15 on només es tenen en compte els correus assignats a algú veiem que n'hi ha bastants menys. El fet que el temps de resposta sigui més gran també es deu a que els correus de factura tinguin una major complexitat.

9.2.1.3 Bústies atenció al client

Tot i que factura és el que més preocupa, hi ha 5 bústies que publica al web Som Energia per contactar segons el tipus de consulta.

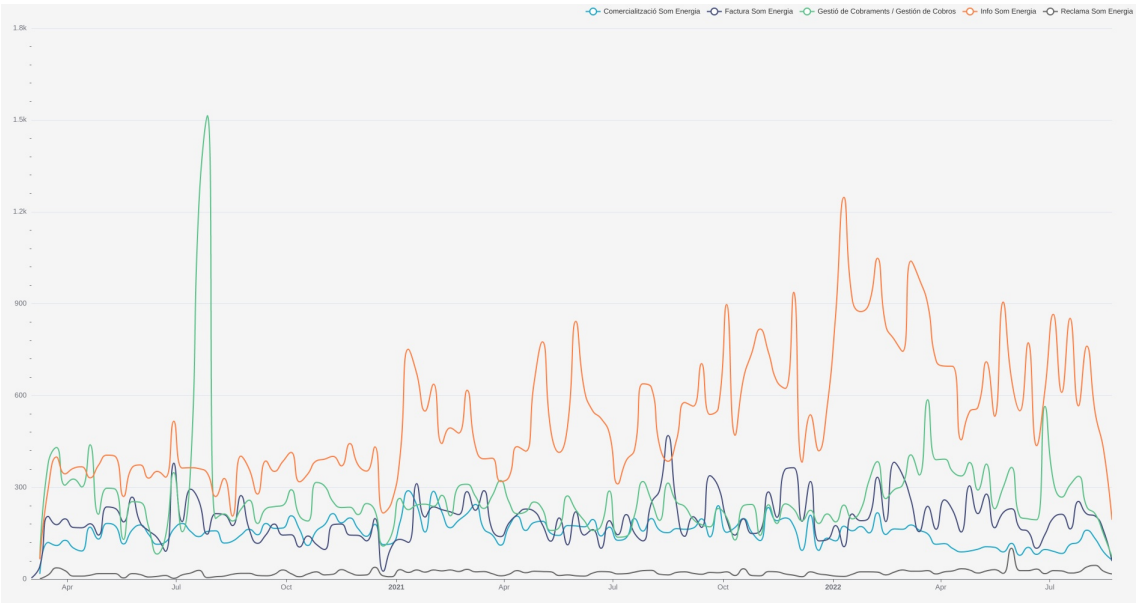


Figura 9.18: Correus tancats a les 5 principals bústies d'atenció al client

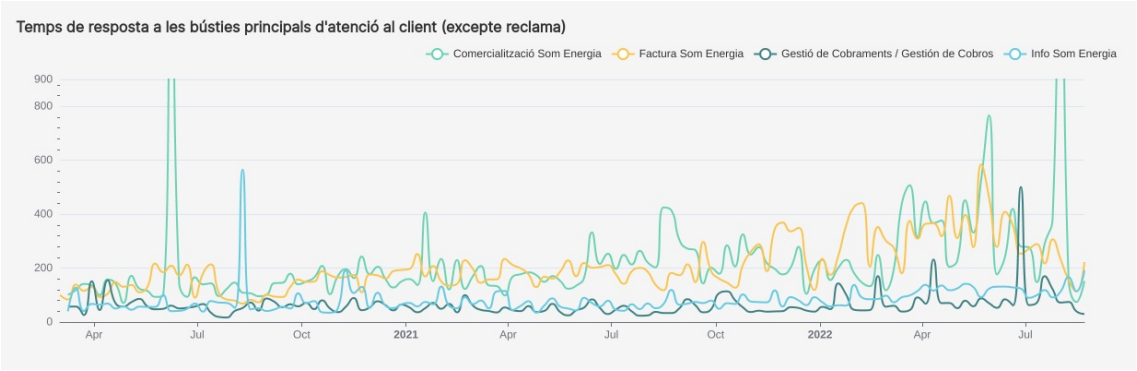


Figura 9.19: Temps de resposta a les 5 principals bústies d'atenció al client

Figura 9.20: Temps de resposta a la bústia de reclamacions

Observant la figura 9.18 i 9.20 sobre els correus tancats a les 5 principals bústies salta la sorpresa, ja que tot i que reclama rep molts pocs correus molt el seu temps de resposta és molt superior a la resta. També veiem a la figura 9.19 com gestió de cobraments té un temps de resposta molt similar al d'info, i comercialització que era l'altre que no haviem analitzat fins ara segueix un trajectoria molt similar a factura.

El gran temps de resposta de reclama (figura 9.20) es deu a que els temes de reclama en molts casos requereixen presentar coumentació, respostes per part de distribuidores o organismes oficials, i tot això fa que pugui passar molt de temps fins que no es resol una reclamació.

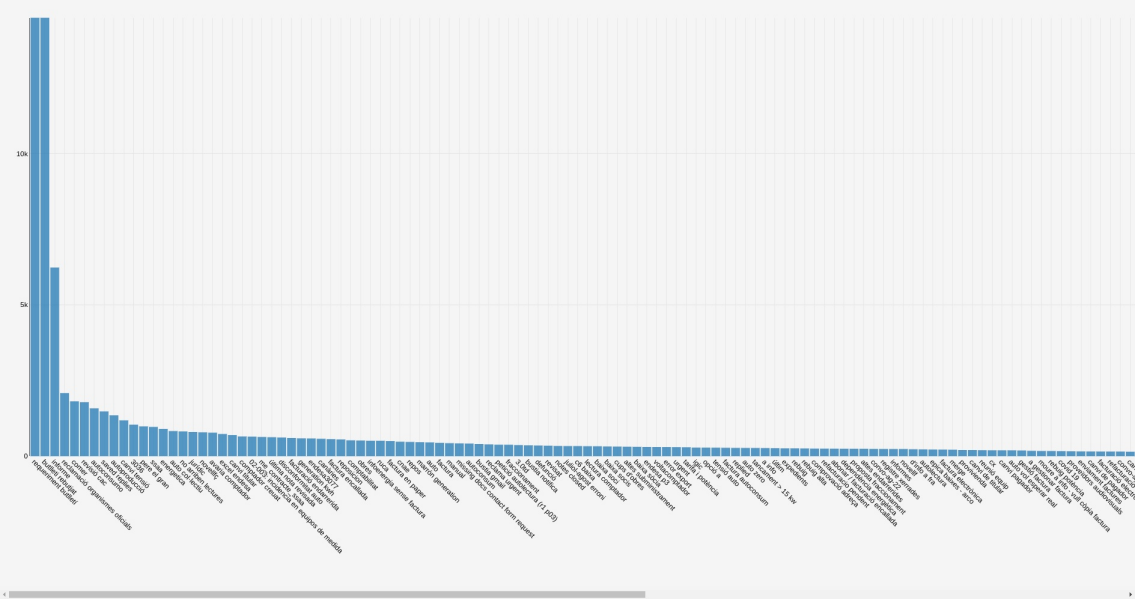


Figura 9.21: Etiquetes que més es tarda a contestar (eix y temps de resposta)

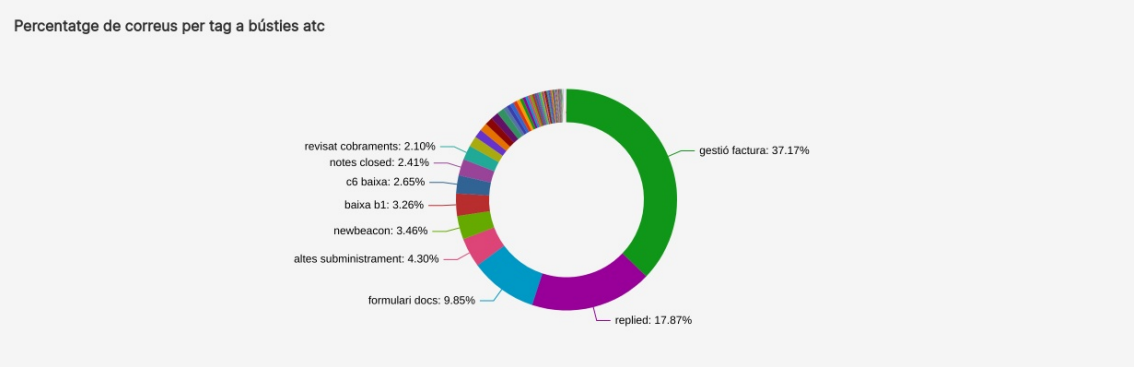


Figura 9.22: Distribució de les etiquetes

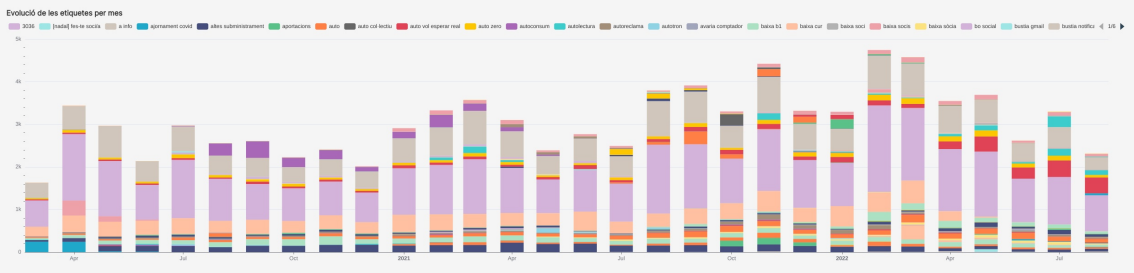


Figura 9.23: Evolució mensual de les etiquetes

Com es pot veure a la figura 9.21 les etiquetes que més es tarden a contestar són les relacionades amb reclamacions, cosa que s'esperava, però mirant a la figura 9.22 el percentatge que hi ha de cada tag veiem com la majoria dels correus etiquetats són de gestió factura, encaixant amb la saturació de factura. A més observem a la figura 9.23 que aquesta majoria es manté al llarg del temps. També cal destacar les tags de formulari docs i newbeacon, que s'apliquen automàticament als correus que provenen de la web, newbeacon pels correus que provenen de la lupa de la pàgina principal i formulari docs dels que provenen del botó de contacte del centre d'ajuda. Per tant sabem que la gent utilitza el triple el fomulari de contacte que la lupa.

Com es comenta al inici, realment quan parlem de correus estem parlant del que helpscout anomena converses, si a partir d'un correu hi ha varies respostes aquestes són els seus threads. A continuació analitzo tot el tema dels threads

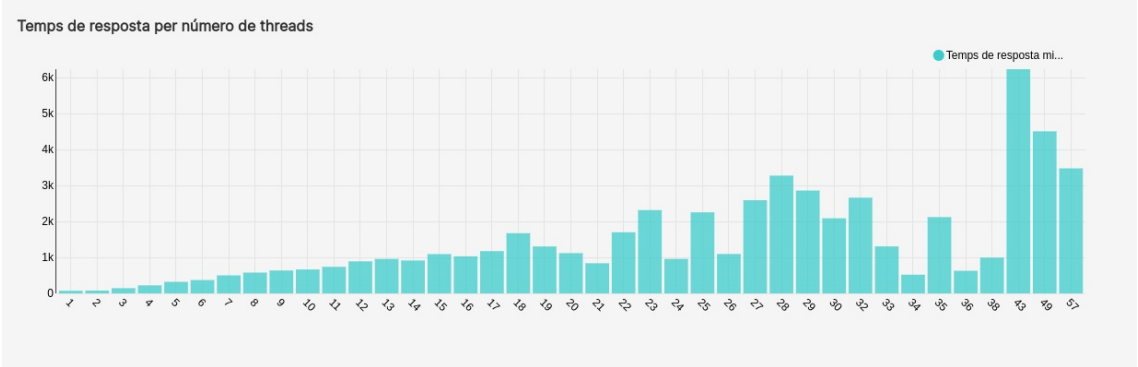


Figura 9.24: Temps de resposta per número de threads

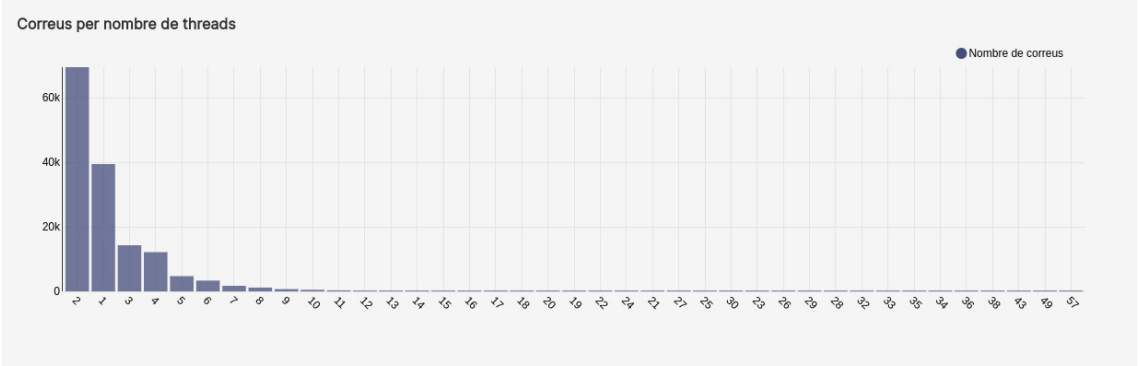


Figura 9.25: Distribució dels correus per nombre de threads

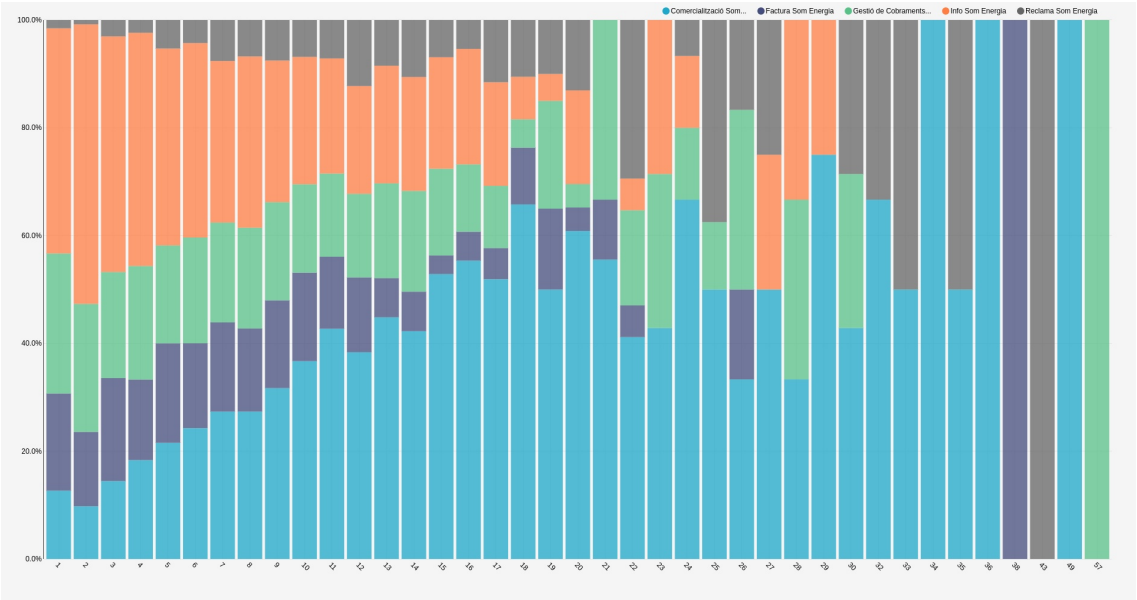


Figura 9.26: Percentatge de threads per mailbox

Analitzant el temps de resposta, veiem a la figura 9.24 que com més threads té una conversa més es tarda a contestar, però al obtenir resultats molt elevats quan hi ha molts threads, també em va sorgir el dubte de si realment era significatiu el nombre de converses que es tardaven tant a contestar, i al mirar a la figura 9.25 quantes converses hi ha per nombre de threads està clar que el més rellevant són les converses amb 2 threads. A la figura 9.26 analitzant per mailbox es veu com a info es tanquen la majoria de correus amb pocs threads i en canvi comercialització té la majoria dels correus amb molts threads.

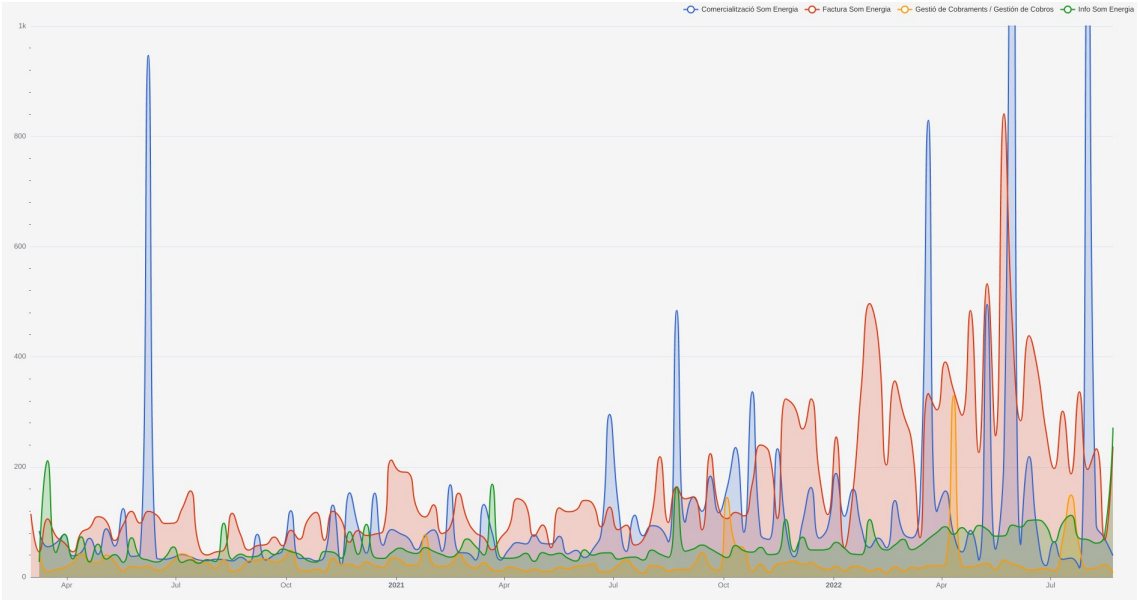


Figura 9.27: Temps de resposta per converses amb dos threads

Al mirar la figura 9.27 veiem com en el temps de resposta per les converses amb 2 threads no hi ha sorpreses, factura és la bústia que tarda més a contestar i cada cop més, en canvi les altres es mantenen relativament estables. Els pics de comercialització els atribueixo més a una mala gestió de la bústia tancat moltes converses antigues de cop que no a una lenta atenció al client.

Figura 9.28: Evolució del nombre de threads per conversa a bústies atc

Veient l'evolució en el temps del nombre de threads de cada conversa a la figura 9.28, el percentatge de converses que es tanquen fàcilment sempre és elevat i el percentage de converses que presenten un nombre elevat de threads són pocs, així doncs es manté una bona feina per part de l'equip per poder resoldre els problemes dels socis amb un sol correu.

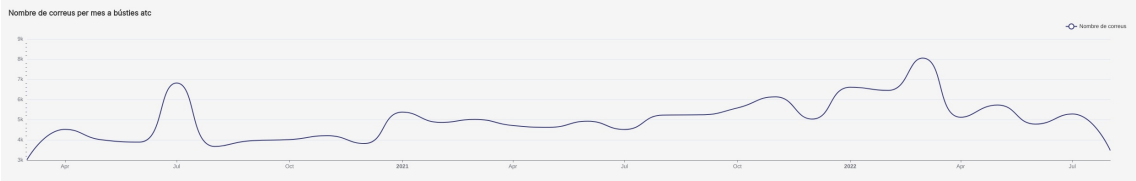


Figura 9.29: Evolució mensual del nombre de correus a bústies atc

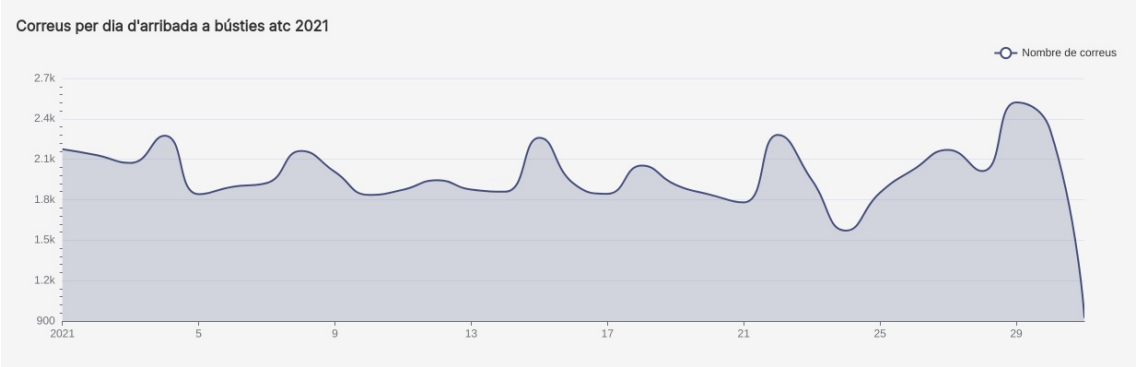


Figura 9.30: Distribució dels correus per dia d'arribada l'any 2021

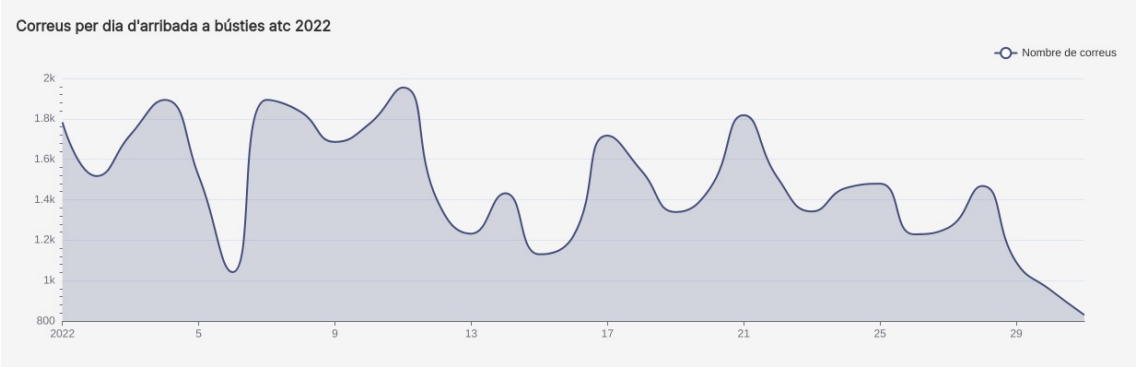


Figura 9.31: Distribució dels correus per dia d'arribada l'any 2022

A la figura 9.29 veiem el pic del juliol de 2020 que correspon al pic de cobraments pel covid i a finals d'any i inicis dels següents també augmenta, probablement degut al canvi de tarifes, al 2021 veiem com els últims mesos de l'any van ser els que van tenir més càrrega de treball i del que portem de 2022 els primers mesos van ser els més durs, degut a com es van disparar els preus al mercat elèctric.

Pel que fa a l'estacionalitat dins del mes, el 2021 (figura 9.30) no hi ha grans diferencies entre inici, mitjans i finals de mes. A la figura 9.31 del que portem de 2022 si que sembla que potser hi ha més tendència als principis de mes.

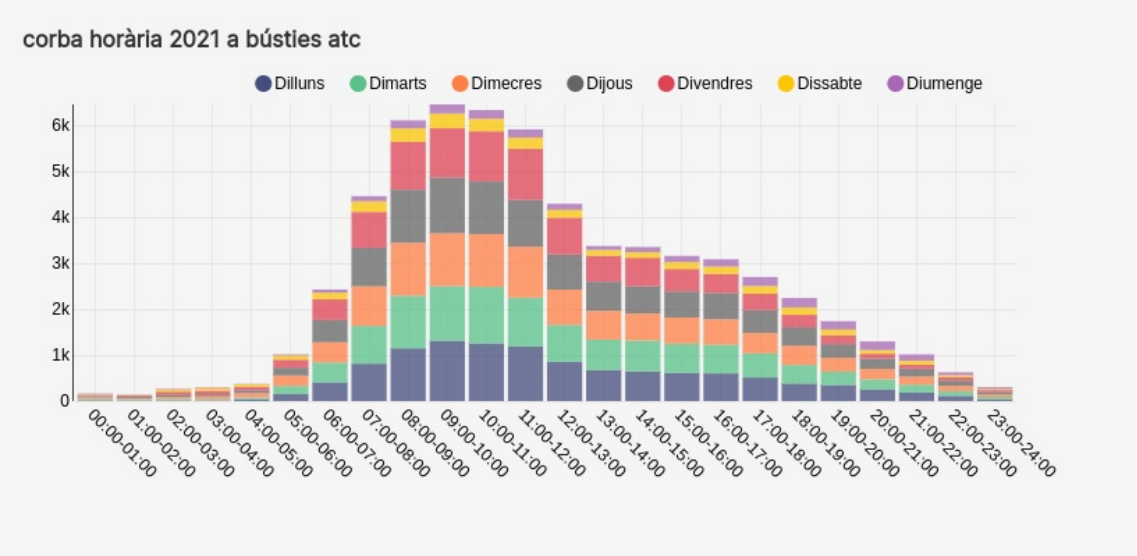


Figura 9.32: Corba horària l'any 2021 a les bústies atc

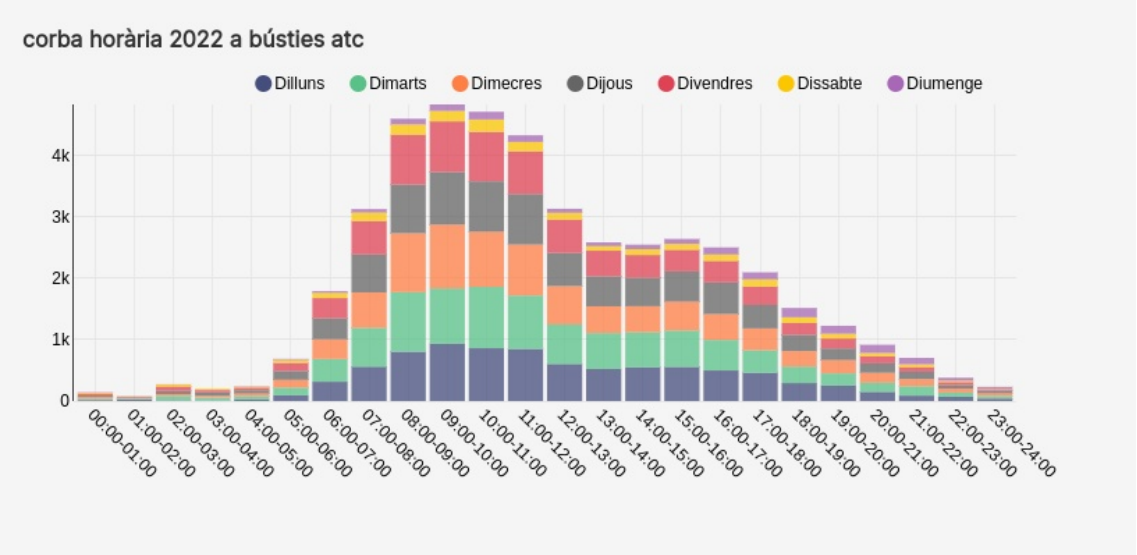


Figura 9.33: Corba horària l'any 2022 a les bústies atc

Si comparem els correus per la seva hora d'arribada, a les figures 9.32 i 9.33 observem com la corba horària no canvia al llarg dels anys, rebent el gruix dels correus pel matí, i en menor mesura per la tarda. Tampoc s'aprecien diferencies a nivell del dia de la setmana que es reben. És important però tenir en ment que les hores que es mostren són UTC.

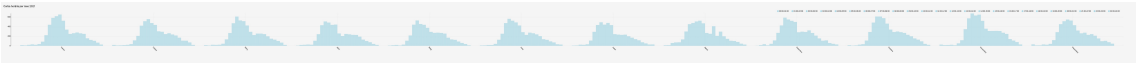


Figura 9.34: Corba horària l'any 2021 per mes a les bústies atc

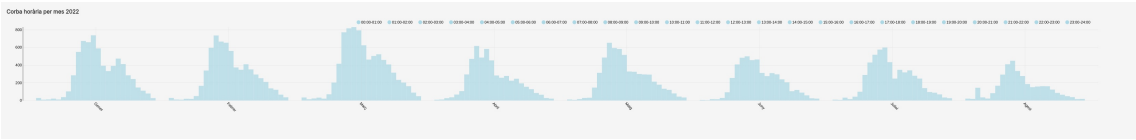


Figura 9.35: Corba horària l'any 2022 per mes a les bústies atc

Comparant mes a mes, tot i que alguns mesos es reben més correus que d'altres i alguns mesos el pic matinal és més marcat tots segueixen la mateixa tendència de que es reben més correus pel matí, possiblement pel fet de que l'horàri d'atenció al client és també pel matí.

9.2.1.4 Comunicació

Quan es prenien els requeriments del projecte durant una reunió amb en Francesc Casadellà, expert en comunicació, que col·labora amb Som Energia des dels seus inicis i actualment pertany al equip de comunicació, va sortir la hipòtesi de que l'enviament de la factura que fem mensualment és la comunicació més important.

Figura 9.36: Respostes a la factura mensual

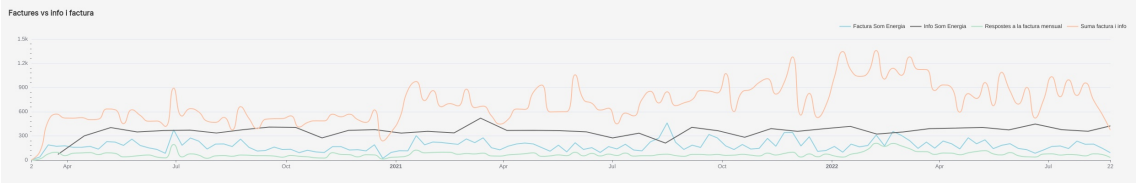


Figura 9.37: Respostes a la factura i correus a info i factura

Les respostes a les factures es tracten principalment a la bústia d'info per qüestions generals i a la bústia de facturació en cas de tractar-se de temes més

complexos que requreixin de la intervenció d'un especialista en facturació. Es pot apreciar a la figura 9.37 com els pics en respostes a les factures que enviem sovint també signifiquen un pic a la bústia de facturació, a la d'info i a la suma de les dues.

Una altra de les necessitats que té l'equip de comunicació és tenir un històric per poder avaluar la seva feina i poder preveure quin serà l'impacte de les seves comunicacions, als gràfics a continuació només es té en compte la bústia d'info ja que és on es reben les respostes a les comunicacions massives.

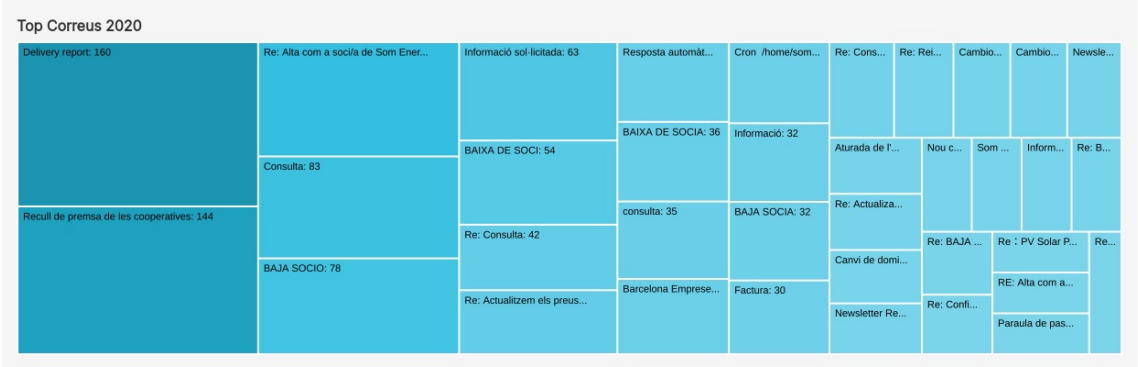


Figura 9.38: Correus més populars l'any 2020

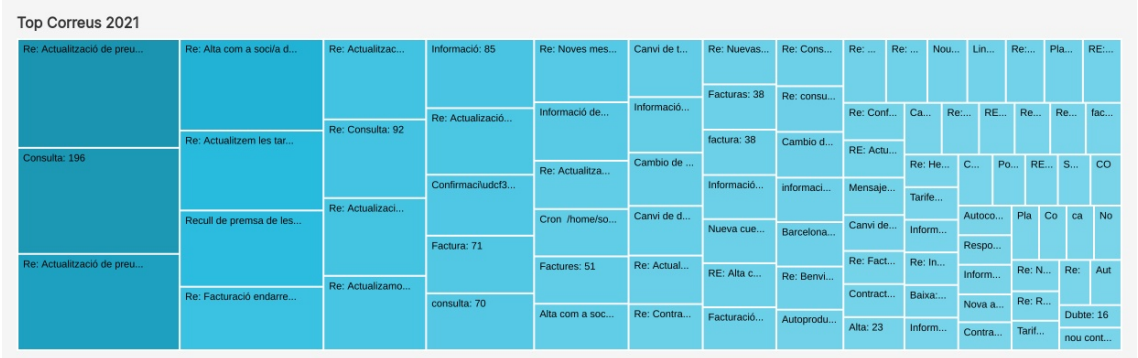


Figura 9.39: Correus més populars l'any 2021

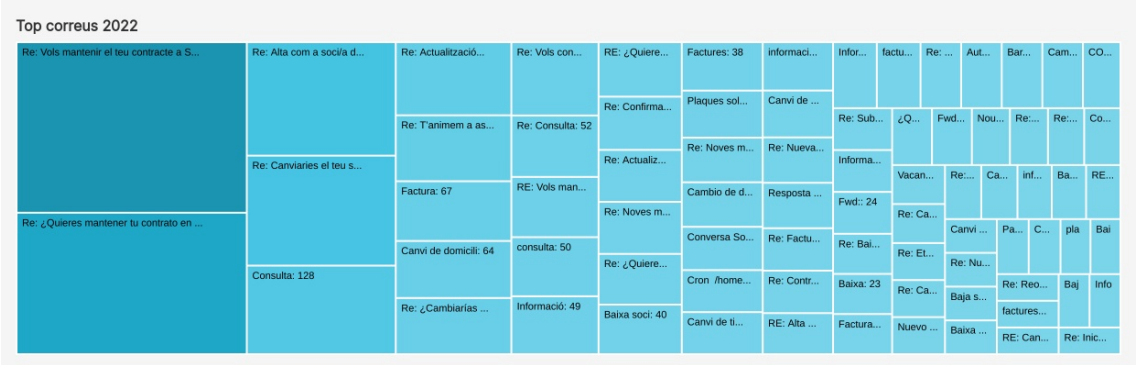


Figura 9.40: Correus més populars l'any 2022

Mirant els correus més rebuts cada any, a la figura 9.38 per l'any 2020 no hi ha res destacable. Al 2021 a la figura 9.39 sí que veiem com els diferents canvis de preus que hi va haver-hi durant l'any van tenir un gran impacte, i el que portem de 2022 (figura 9.40) per sort el correu més contestat és el correu que va enviar Som Energia a les persones que tenien un contracte tot i no ser sòcies animant-les a associar-se si no volien passar el seu contracte a la comercialitzadora de referència. Quasi 700 respostes és un gran èxit per la quantitat de correus d'aquest tipus que es van enviar. Aquests 3 gràfics però només són representatius pels enviaments massius, ja que les respostes tindran el mateix assumpte. També s'ha de tenir en compte que algun s'han utilitzat adreces no reply per fer algun enviament, que tot i ser contrari als valors de Som Energia en algun moment s'ha cregut aportú per evitar saturar l'atenció al client.

Durant l'elaboració del projecte m'he adonat d'alguns correus que eren respostes automàtiques, aquests es produeixen quan la gent està de vacances i no fa cas al correu. Es pot veure com clarament l'agost no és un bon moment per fer comunicacions i també en menor mesura el nadal.

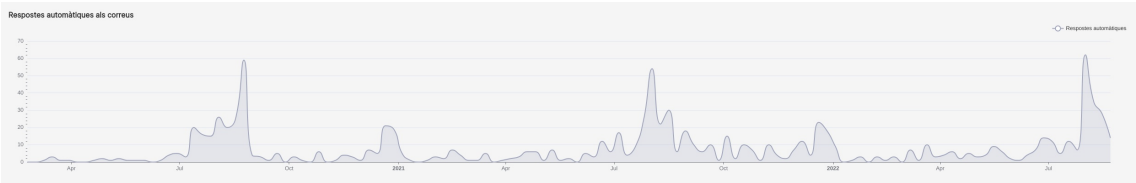


Figura 9.41: Respostes automàtiques als correus

Finalment, a casa tenim la llum contractada amb Som Energia i recentment vam participar en una compra col·lectiva, aquesta no va anar tant bé com esperàvem i després d'una trucada se li va demanar a la meva mare que envies un correu explicant la situació. Llavors tenia curiositat des d'un punt de vista personal sobre si això era habitual o un cas aïllat

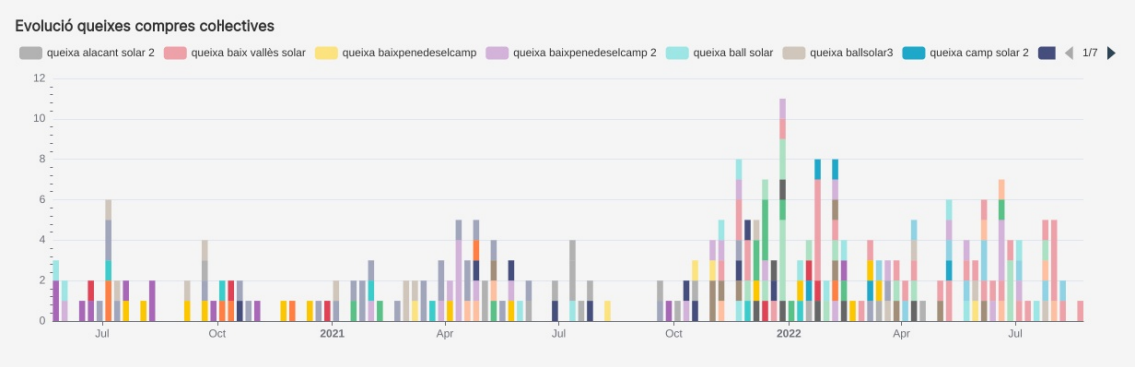


Figura 9.42: Evolució de les queixes per les compres col·lectives

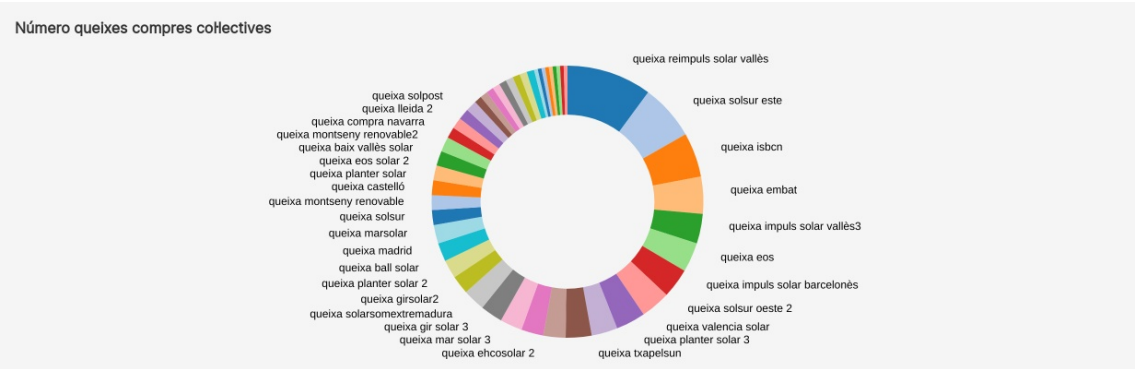


Figura 9.43: Queixes per cada compra col·lectiva

Després d'analitzar els correus de la bústia d'autoproducció veiem clarament a la figura 9.43 com hi ha poques queixes pel gran nombre de compres col·lectives que es fan, en el cas de la que vaig participar només hi va haver 6 correus de queixa de 100 persones que hi van participar. En aquest gràfics però no es té en compte la mida de la compra col·lectiva, per exemple la compra col·lectiva que presenta més queixes és també una on més persones han participat i tot i que l'evolució en el temps, com mostra la figura 9.42 cada cop hi ha més queixes, el nombre de compres col·lectives també ha augmentat.

CAPÍTOL 10

Conclusions

10.1 Assoliment dels objectius del PFG

S'han assolit els objectius del PFG perquè s'ha construït la part més important de la infraestructura de dades de Som Energia i ja s'està utilitzant per altres projectes a part del PFG.

L'equip de comunicació i també el d'atenció i suport han obtingut una informació que fins ara no tenien i els ajudaran a millorar a partir d'ara, a més de valorar molt positivament els resultats.

10.2 Assoliments dels requisits

A la presentació dels resultats a l'equip d'atenció i suport i el de comunicació van sorprendre positivament els resultats, ja que no esperaven obtenir tanta informació i tant detallada.

10.3 Desviació de la planificació original

S'han fet més sprints dels planificats originalment. Això ha suposat s'ha fet molta més feina en infraestructura del que en un principi s'havia planejat, però també deixar sense fer altres fonts de dades tot i que al haver prioritzat fer una infraestructura sòlida i reutilitzable integrar més fonts en un futur serà molt més fàcil.

10.4 Crítica dels resultats

El resultat final visible pels usuaris ha estat el dashboard, que ha estat rebut molt positivament, superant les seves expectatives. S'han complert tots els requeriments inicials basats en els correus.

Per la part d'infraestructura tot i que s'esperava que Airflow s'adaptés més fàcilment a les nostres necessitats, l'infraestructura resultant també ha tingut una rebuda molt positiva per part dels companys, que ja l'estan utilitzant. El

resultat final també ha implicat millores molt important comparat amb només les eines que oferia Airflow.

L'anàlisi dels correus ha servit molt a l'equip de comunicació per poder millorar quan enviar comunicacions massives per no saturar l'atenció al client, i ara també tenen una eina on consultar ràpidament l'impacte de la seva feina.

També ha ajudat a l'equip d'atenció i suport a comprendre millor la seva feina i poder prendre decisions basades en dades per afrontar millor les pròximes crisis que sembla que esperen al sector elèctric.

Fins ara no existia cap eina per poder avaluar com estava funcionant l'atenció al client i amb aquest projecte ja es pot saber si realment es tarden més a contestar el correus que anteriorment i quan es tarda depenent del tipus de correu.

CAPÍTOL 11

Treball futur

11.1 Treball en curs

Actualment hi ha dues pull requests obertes:

- Migrar les mailboxes als nous estàndars de SQLAlchemy ORM i actualitzar-se automàticament amb un DAG, ja que tot i que les bústies de correus no canviïn de habitualment, actualment no s'estan actualitzant.
- Detectar automàticament mitjançant intel·ligència artificial l'idioma dels correus, el que permetria obtenir resultats sobre l'us de les llengües a Som Energia.

Aquestes pull requests està previst mergejar-les un cop algun company les hagi revisat.

11.2 Treball futur

11.2.1 DockerSwarm o Kubernetes

Es van fer proves amb el DockerSwarmOperator sense èxit, en un futur també es podria provar d'utilitzar kubernetes pels orquestar els contenidors, a més Portainer té suport per kubernetes i docker swarm.

11.2.2 Trucades

Les trucades són juntament amb el correu electrònic els mitjans principals que utilitzen els socis per comunicar-se amb la cooperativa. Cada dia una part molt important de la jornada laboral de l'equip tècnic es destina a fer atenció telefònica.

11.2.3 Xarxes socials

Tot i que ja s'havia començat a treballar el tema es va abandonar ja que es va considerar poc prioritari, encara que les xarxes socials no és on més activitat té Som Energia també és una font de dades que s'acabarà integrant

11.2.4 Sendgrid

Amb les dades que ens proporciona Sengrid podem saber quan una persona obre un correu, per tant es podria millorar molt l'anàlisi de l'impacte de cada comunicació que es fa per correu.

Permetria conèixer a part de les respostes, que ja les sabem, quan s'obre el correu o si es cliquen els links.

11.2.5 Mailchimp

A part de Sengrid també es realitzen comunicacions a través de Mailchimp, per tant també es podria afegir com a font de dades en per obtenir el mateix que amb Sengrid però pels correus que s'envien per Mailchimp.

Codi

El codi del projecte es pot trobar en el respositori públic somenergia-kpis. Al estar barrejat amb més codi l'adjunto també com a Annex.

```
from airflow.operators.python_operator import
 ↪  BranchPythonOperator
from airflow import DAG
import os.path

def checkIfRepoAlreadyCloned():
    if
     ↪  os.path.exists('/opt/airflow/repos/somenergia-kpis/.git'):
        return 'git_pull_task'
    return 'git_clone'

def build_check_repo_task(dag: DAG) ->
 ↪  BranchPythonOperator:
    check_repo_task = BranchPythonOperator(
        task_id='check_repo_task',
        python_callable=checkIfRepoAlreadyCloned,
        do_xcom_push=False,
        dag=dag,
    )

    return check_repo_task
```

Tasca per comprovar si existeix el repositori

```
from airflow.operators.python_operator import
 ↪  BranchPythonOperator
from airflow import DAG
import paramiko
import io
```

```
def pull_repo_ssh(repo_server_url, repo_server_key,
 ↪  task_name):
    p = paramiko.SSHClient()
    p.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    keyfile = io.StringIO(repo_server_key)
    mykey = paramiko.RSAKey.from_private_key(keyfile)
    p.connect(repo_server_url, port=2200,
     ↪  username="airflow", pkey=mykey)
    stdin, stdout, stderr = p.exec_command("git -C
     ↪  /opt/airflow/repos/somenergia-kpis pull")
    txt_stderr = stderr.readlines()
    txt_stderr = "".join(txt_stderr)
    print (f"Stderr de git pull ha retornat {txt_stderr}")
    # si stderr té més de 0 \n és que hi ha canvis al fer
     ↪  pull
    #Your configuration specifies to merge with the ref
     ↪  'refs/heads/main' from the remote, but no such ref
     ↪  was fetched.
    #Apareix quan fem molts git pull a la vegada
    return "image_remove" if txt_stderr.count('\n')>0 and
     ↪  not 'no such ref was fetched' in txt_stderr else
     ↪  task_name

def build_branch_pull_ssh_task(dag: DAG, task_name) ->
 ↪  BranchPythonOperator:
    branch_pull_ssh_task = BranchPythonOperator(
        task_id='git_pull_task',
        python_callable=pull_repo_ssh,
        op_kwargs={ "repo_server_url": "{{
         ↪  var.value.repo_server_url }}",
                    "repo_server_key": "{{
                     ↪  var.value.repo_server_key }}",
                    "task_name": task_name},
        do_xcom_push=False,
        dag=dag,
    )

    return branch_pull_ssh_task
```

Tasca per fer un git pull

```
from airflow.operators.python_operator import
 ↪  PythonOperator
from airflow import DAG
import paramiko
import io

def cloneRepoSSH(repo_server_url,repo_server_key):
    p = paramiko.SSHClient()
    p.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    keyfile = io.StringIO(repo_server_key)
    mykey = paramiko.RSAKey.from_private_key(keyfile)
    p.connect(repo_server_url, port=2200,
     ↪  username="airflow", pkey=mykey)
    p.exec_command("git clone
     ↪  https://github.com/Som-Energia/somenergia-kpis.git
     ↪  /opt/airflow/repos/somenergia-kpis")

def build_git_clone_ssh_task(dag: DAG) -> PythonOperator:
    git_clone_ssh_task = PythonOperator(
        task_id='git_clone_ssh_task',
        python_callable=cloneRepoSSH,
        op_kwargs={ "repo_server_url" : "{{
         ↪  var.value.repo_server_url }}",
                    "repo_server_key": "{{
                     ↪  var.value.repo_server_key }}" },
        dag=dag,
    )

    return git_clone_ssh_task
```

Tasca per fer un git clone

```
from airflow import DAG
from airflow.operators.python import PythonOperator
import requests

def _process(portainer_key, post_url):
```

```
    r = requests.post(url=post_url, headers={'x-api-key' :
     ↪  portainer_key, 'Content-Type':'application/json'},
     ↪  params={'t':'somenergia-kpis-requirements:latest',
     ↪  'remote':
     ↪  'https://github.com/Som-Energia/somenergia-kpis.git#main',
     ↪  'nocache': 'true'}, data='{}', verify=False)

def build_image_build_task(dag: DAG) -> PythonOperator:

    task_image_build = PythonOperator(
             task_id='image_build',
             python_callable=_process,
             op_kwargs={'portainer_key':"{{
              ↪  var.value.portainer_api_key }}",
                        'post_url':"{{
                         ↪  var.value.docker_build_url
                         ↪  }}"},
             trigger_rule='one_success',
             dag=dag,
             )

    return task_image_build
```

Tasca per contstruïr una imatge

```
from airflow import DAG
from airflow.operators.python import PythonOperator
import requests

def _process(portainer_key,remove_url):
    r = requests.delete(url=remove_url,
     ↪  headers={'x-api-key' : portainer_key,
     ↪  'Content-Type':'application/json'}, data='{}',
     ↪  verify=False)

def build_remove_image_task(dag: DAG) -> PythonOperator:

    task_remove_image = PythonOperator(
             task_id='image_remove',
```

```
            python_callable=_process,
            op_kwargs={'portainer_key':"{{
             ↪  var.value.portainer_api_key }}",
                       'remove_url': "{{
                        ↪  var.value.docker_remove_url
                        ↪  }}"},
            dag=dag,
            trigger_rule='one_success',
            )

    return task_remove_image
```

Tasca per contstruïr una imatge

```
from airflow import DAG
from airflow.providers.docker.operators.docker import
 ↪  DockerOperator
from kpis_tasks.t_branch_pull_ssh import
 ↪  build_branch_pull_ssh_task
from kpis_tasks.t_git_clone_ssh import
 ↪  build_git_clone_ssh_task
from kpis_tasks.t_check_repo import build_check_repo_task
from kpis_tasks.t_image_build import build_image_build_task
from kpis_tasks.t_remove_image import
 ↪  build_remove_image_task
from docker.types import Mount, DriverConfig
from datetime import datetime, timedelta
from airflow.models import Variable

my_email = Variable.get("fail_email")
addr = Variable.get("repo_server_url")

args= {
  'email': my_email,
  'email_on_failure': True,
  'email_on_retry': False,
  'retries': 5,
  'retry_delay': timedelta(minutes=5),
}
```

```
nfs_config = {
    'type': 'nfs',
    'o': f'addr={addr},nfsvers=4',
    'device': ':/opt/airflow/repos'
}

driver_config = DriverConfig(name='local',
 ↪  options=nfs_config)
mount_nfs = Mount(source="local", target="/repos",
 ↪  type="volume", driver_config=driver_config)

with DAG(dag_id='hs_get_conversations_dag',
 ↪  start_date=datetime(2020,3,20),
 ↪  schedule_interval='@hourly', catchup=True,
 ↪  tags=["Helpscout", "Extract"], default_args=args) as
 ↪  dag:

    task_branch_pull_ssh =
     ↪  build_branch_pull_ssh_task(dag=dag,
     ↪  task_name='hs_get_conversations')
    task_git_clone = build_git_clone_ssh_task(dag=dag)
    task_check_repo = build_check_repo_task(dag=dag)
    task_image_build = build_image_build_task(dag=dag)
    task_remove_image= build_remove_image_task(dag=dag)

    get_conversations_task = DockerOperator(
        api_version='auto',
        task_id='hs_get_conversations',
        image='somenergia-kpis-requirements:latest',
        working_dir='/repos/somenergia-kpis',
        command='python3 -m
         ↪  datasources.helpscout.hs_get_conversations "{{
         ↪  data_interval_start }}" "{{ data_interval_end
         ↪  }}" \
                "{{ var.value.puppis_prod_db }}" "{{
 ↪  var.value.helpscout_api_id }}" "{{
 ↪  var.value.helpscout_api_secret }}"',
        docker_url=Variable.get("moll_url"),
        mounts=[mount_nfs],
        mount_tmp_dir=False,
```

```
        auto_remove=True,
        retrieve_output=True,
        trigger_rule='none_failed',
    )

    task_check_repo >> task_git_clone
    task_check_repo >> task_branch_pull_ssh
    task_git_clone >> task_image_build
    task_branch_pull_ssh >> get_conversations_task
    task_branch_pull_ssh >> task_remove_image
    task_remove_image >> task_image_build >>
     ↪  get_conversations_task
```

DAG per obtenir les converses

```
from airflow import DAG
from airflow.providers.docker.operators.docker import
 ↪  DockerOperator
from kpis_tasks.t_branch_pull_ssh import
 ↪  build_branch_pull_ssh_task
from kpis_tasks.t_git_clone_ssh import
 ↪  build_git_clone_ssh_task
from kpis_tasks.t_check_repo import build_check_repo_task
from kpis_tasks.t_image_build import build_image_build_task
from kpis_tasks.t_remove_image import
 ↪  build_remove_image_task
from docker.types import Mount, DriverConfig
from datetime import datetime, timedelta
from airflow.models import Variable

my_email = Variable.get("fail_email")
addr = Variable.get("repo_server_url")

args= {
  'email': my_email,
  'email_on_failure': True,
  'email_on_retry': False,
  'retries': 5,
  'retry_delay': timedelta(minutes=5),
```

```
}

nfs_config = {
    'type': 'nfs',
    'o': f'addr={addr},nfsvers=4',
    'device': ':/opt/airflow/repos'
}

driver_config = DriverConfig(name='local',
 ↪  options=nfs_config)
mount_nfs = Mount(source="local", target="/repos",
 ↪  type="volume", driver_config=driver_config)

with DAG(dag_id='hs_get_tags_dag',
 ↪  start_date=datetime(2022,6, 15),
 ↪  schedule_interval='@hourly', catchup=True,
 ↪  tags=["Helpscout", "Extract"], default_args=args) as
 ↪  dag:

    task_branch_pull_ssh =
     ↪  build_branch_pull_ssh_task(dag=dag,
     ↪  task_name='hs_get_tags')
    task_git_clone = build_git_clone_ssh_task(dag=dag)
    task_check_repo = build_check_repo_task(dag=dag)
    task_image_build = build_image_build_task(dag=dag)
    task_remove_image= build_remove_image_task(dag=dag)

    get_tags_task = DockerOperator(
        api_version='auto',
        task_id='hs_get_tags',
        image='somenergia-kpis-requirements:latest',
        working_dir='/repos/somenergia-kpis',
        command='python3 -m
         ↪  datasources.helpscout.hs_get_tags "{{
         ↪  data_interval_start }}" "{{ data_interval_end
         ↪  }}" \
                "{{ var.value.puppis_prod_db}}" "{{
 ↪  var.value.helpscout_api_id}}" "{{
 ↪  var.value.helpscout_api_secret}}"',
        docker_url=Variable.get("moll_url"),
        mounts=[mount_nfs],
```

```
        mount_tmp_dir=False,
        auto_remove=True,
        retrieve_output=True,
        trigger_rule='none_failed',
    )

    task_check_repo >> task_git_clone
    task_check_repo >> task_branch_pull_ssh
    task_git_clone >> task_image_build
    task_branch_pull_ssh >> get_tags_task
    task_branch_pull_ssh >> task_remove_image
    task_remove_image >> task_image_build >> get_tags_task
```

DAG per obtenir les etiquetes

```
from airflow import DAG
from airflow.providers.docker.operators.docker import
 ↪  DockerOperator
from airflow.sensors.external_task import
 ↪  ExternalTaskSensor
from docker.types import Mount, DriverConfig
from datetime import datetime, timedelta
from airflow.models import Variable


my_email = Variable.get("fail_email")
addr = Variable.get("repo_server_url")

args= {
  'email': my_email,
  'email_on_failure': True,
  'email_on_retry': False,
  'retries': 5,
  'retry_delay': timedelta(minutes=5),
}

nfs_config = {
    'type': 'nfs',
    'o': f'addr={addr},nfsvers=4',
```

```
    'device': ':/opt/airflow/repos'
}

driver_config = DriverConfig(name='local',
 ↪  options=nfs_config)
mount_nfs = Mount(source="local", target="/repos",
 ↪  type="volume", driver_config=driver_config)

with DAG(dag_id='hs_transform_conversations_dag',
 ↪  start_date=datetime(2020,3,20),
 ↪  schedule_interval='@hourly', catchup=True,
 ↪  tags=["Helpscout", "Transform"], default_args=args) as
 ↪  dag:

    sensor_hs_cru = ExternalTaskSensor(
        task_id="sensor_hs_cru",
        external_dag_id='hs_get_conversations_dag',
        external_task_id='hs_get_conversations',
        allowed_states=['success'],
        failed_states=['failed', 'skipped'],
    )

    transform_conversations_task = DockerOperator(
        api_version='auto',
        task_id='hs_transform_conversations',
        image='somenergia-kpis-requirements:latest',
        working_dir='/repos/somenergia-kpis',
        command='python3 -m
         ↪  pipelines.hs_transform_conversations "{{
         ↪  data_interval_start }}" "{{ data_interval_end
         ↪  }}" \
                "{{ var.value.puppis_prod_db }}"',
        docker_url=Variable.get("moll_url"),
        mounts=[mount_nfs],
        mount_tmp_dir=False,
        auto_remove=True,
        retrieve_output=True,
        trigger_rule='none_failed',
    )

    sensor_hs_cru >> transform_conversations_task
```

DAG per transformar les converses

```
from datetime import timedelta
from helpscout.client import HelpScout
from sqlalchemy.dialects.postgresql import JSONB
from sqlalchemy import Table, Column, Integer, MetaData,
 ↪  DateTime
from sqlalchemy import create_engine
import sys
import pendulum

def create_HS_engine(engine, hs_app_id, hs_app_secret):
    hs = HelpScout(app_id=hs_app_id,
     ↪  app_secret=hs_app_secret,
     ↪  sleep_on_rate_limit_exceeded=True)
    return hs, engine

def create_table(engine):
    table_name = 'hs_conversation'
    meta = MetaData(engine)
    conv_table = Table(
        table_name,
        meta,
        Column('id', Integer, primary_key=True),
        Column('data', JSONB),
        Column('task_data_interval_start', DateTime),
        Column('task_data_interval_end', DateTime)
    )
    conv_table.create(engine, checkfirst=True)

    return conv_table

def get_conversations(hs, engine, conv_table, inici, fi,
 ↪  status):

    params = f"query=(modifiedAt:[{inici} TO
     ↪  {fi}])&status={status}"
```

```
    print(f"Let's get conversations with params: {params}")

    conversations = hs.conversations.get(params=params)

    print(f"Let's insert conversations")

    for c in conversations:
        statement =
         ↪  conv_table.insert().values(data=c.__dict__,task_data_interval_st
         ↪  task_data_interval_end=pendulum.parse(fi))
        engine.execute(statement)

    return True

def update_hs_conversations(verbose=2, dry_run=False,
 ↪  inici=None, fi=None):
    args = sys.argv[1:]
    data_interval_start=pendulum.parse(args[0])
    data_interval_end=pendulum.parse(args[1])
    #fem de la setmana anterior
    data_interval_start = data_interval_start -
     ↪  timedelta(days=7)
    data_interval_end = data_interval_end -
     ↪  timedelta(days=7)
    #tornem a passar a string
    data_interval_start =
     ↪  data_interval_start.strftime("%Y-%m-%dT%H:%M:%SZ")
    data_interval_end =
     ↪  data_interval_end.strftime("%Y-%m-%dT%H:%M:%SZ")

    engine = create_engine(args[2])
    hs_app_id = args[3]
    hs_app_secret = args[4]

    hs, engine = create_HS_engine(engine, hs_app_id,
     ↪  hs_app_secret)
    conv_table = create_table(engine)

    params = {
        'inici': data_interval_start,
        'fi': data_interval_end,
```

```
        'status':'closed',
    }

    return get_conversations(hs, engine, conv_table,
     ↪  **params)

if __name__ == '__main__':
    update_hs_conversations()
```

Script per obtenir les converses

```
from helpscout.client import HelpScout
import pandas as pd
from sqlalchemy import create_engine
import sys
import pendulum
from sqlalchemy.orm import Session

from classes.models import HS_tag

def create_HS_engine(engine, hs_app_id, hs_app_secret):
    hs = HelpScout(app_id=hs_app_id,
     ↪  app_secret=hs_app_secret,
     ↪  sleep_on_rate_limit_exceeded=True)
    return hs, engine

def update_tags(engine, hs_app_id, hs_app_secret, dis,
 ↪  die):

    hs, engine = create_HS_engine(engine, hs_app_id,
     ↪  hs_app_secret)
    #Importem totes les tags de Som Energia a HelpScout
    tags = hs.tags.get()

    #per tema idempotencia nomes les mes noves de l'ultima
     ↪  execucio
    tags_insert = [HS_tag(id=t.id, name=t.name) for t in
     ↪  tags if dis < pendulum.parse(t.createdAt) and
     ↪  pendulum.parse(t.createdAt) <= die]
```

```
    print(f"insertem {len(tags_insert)} tags")
    with Session(engine) as session:
        with session.begin():
            session.add_all(tags_insert)

if __name__ == '__main__':
    args = sys.argv[1:]
    dis = pendulum.parse(args[0])
    die = pendulum.parse(args[1])
    engine = create_engine(args[2])
    hs_app_id = args[3]
    hs_app_secret = args[4]

    update_tags(engine, hs_app_id, hs_app_secret, dis, die)
```

Script per obtenir les etiquetes

```
from datetime import timedelta
from sqlalchemy import create_engine, text
import sys
import pendulum
from sqlalchemy.orm import Session

from classes.models import HS_tag, HS_clean_conversation


def hs_clean_conversation_from_dict(data, dict_tags, start,
 ↪  end):
    tags = [dict_tags[t['id']] for t in data['tags']]
    customerWaitingSince_time =
     ↪  pendulum.parse(data['customerWaitingSince']['time'])
     ↪  if 'time' in data['customerWaitingSince'] else None
    return HS_clean_conversation(
        number=data['number'], id_helpscout=data['id'],
         ↪  threads=data['threads'], type=data['type'],
         ↪  folder_id=data['folderId'],
        status=data['status'], state=data['state'],
         ↪  subject=data.get('subject',''),
         ↪  mailbox_id=data['mailboxId'],
         ↪  created_at=pendulum.parse(data['createdAt']) ,
```

```
        closed_by=data['closedBy'],
         ↪  closed_at=pendulum.parse(data.get('closedAt',end)),
         ↪  user_updated_at=pendulum.parse(data['userUpdatedAt']),
        cc=data['cc'], bcc=data['bcc'],
         ↪  created_by_id=data['createdBy']['id'],
         ↪  created_by_email=data['createdBy']['email'],
         ↪  closed_by_user_email=data['closedByUser']['email'],

         ↪  customer_waiting_since_time=customerWaitingSince_time,
         ↪  source_type=data['source']['type'],
         ↪  source_via=data['source']['via'],
         ↪  primary_customer_id=data['primaryCustomer']['id'],

         ↪  primary_customer_email=data['primaryCustomer'].get('email','')
         ↪  assignee_id=data.get('assignee',{'id':0})['id'],
         ↪  assignee_email=data.get('assignee',{'email':''})['email'],
        tags=tags,
        task_data_interval_start=pendulum.parse(start),
         ↪  task_data_interval_end=pendulum.parse(end)
    )

def move_conversations(engine, inici, fi):

    print(f"Let's get conversations")


    textual_sql = text("SELECT * from hs_conversation where
     ↪  cast(data->'status' as text) = '\"closed\"' and
     ↪  cast(data->'createdBy'->'email' as text) not like
     ↪  '%@somenergia.coop' \
                        and data->'closedByUser'->'email'
 ↪  != '\"none@nowhere.com\"' and task_data_interval_start
 ↪  = :dis and task_data_interval_end = :die")
    result = engine.execute(textual_sql,dis=inici, die=fi)
     ↪  #dis i die han de tenir un timedelta d'una hora

    print(f"Let's insert {result.rowcount} conversations")

    with Session(engine) as session:
        with session.begin():
            tags = session.query(HS_tag)
```

```
            dict_tags = {t.id: t for t in tags}
            conversations_insert =
             ↪  [hs_clean_conversation_from_dict(e.data,
             ↪  dict_tags, inici, fi) for e in result]
            session.add_all(conversations_insert)

    return True

def transform_hs_conversations(verbose=2, dry_run=False,
 ↪  inici=None, fi=None):
    args = sys.argv[1:]
    data_interval_start=pendulum.parse(args[0])
    data_interval_end=pendulum.parse(args[1])
    #fem de la setmana anterior
    data_interval_start = data_interval_start -
     ↪  timedelta(days=7)
    data_interval_end = data_interval_end -
     ↪  timedelta(days=7)
    #tornem a passar a string
    data_interval_start =
     ↪  data_interval_start.strftime("%Y-%m-%dT%H:%M:%SZ")
    data_interval_end =
     ↪  data_interval_end.strftime("%Y-%m-%dT%H:%M:%SZ")

    engine = create_engine(args[2])

    move_table = None

    params = {
        'inici': data_interval_start,
        'fi': data_interval_end,
    }

    return move_conversations(engine, **params)

if __name__ == '__main__':
    transform_hs_conversations()
```

Script per transformar les converses

```
import unittest

from sqlalchemy_utils import (
    assert_nullable,
    assert_non_nullable,
)
from sqlalchemy.orm import sessionmaker
from sqlalchemy import Column, Integer, DateTime, String,
 ↪  ForeignKey, create_engine
from sqlalchemy.orm import relationship

import pendulum

from .hs_transform_conversations import
 ↪  hs_clean_conversation_from_dict


from classes.models import (
    Base,
    HS_clean_conversation,
    HS_tag,
    Conversation_tag
)


class HelpscoutTransformTest(unittest.TestCase):

    engine = create_engine('sqlite:///:memory:')
    Session = sessionmaker(bind=engine)
    session = Session()

    def setUp(self):
        Base.metadata.create_all(self.engine)
        self.tag = HS_tag(id=1, name='test_tag')
        self.session.add(self.tag)
        self.conv = HS_clean_conversation(id=1, number=2,
         ↪  id_helpscout=3,
         ↪  customer_waiting_since_time=None,
         ↪  tags=[self.tag])
        self.session.add(self.conv)
```

```
        self.session.commit()

    def tearDown(self):
        Base.metadata.drop_all(self.engine)

    def base_hs_data(self):
        sample_time = '1970-01-01 00:00:00'
        data = {
            'number': 1000,
            'id': 1000,
            'threads': 1000,
            'type': 'lolo',
            'folderId': 1000,
            'status': 'blabla',
            'state': 'blabla',
            'subject': 'blabla',
            'mailboxId': 1,
            'createdAt': sample_time,
            'closedBy': 'blabla',
            'closedAt': sample_time,
            'userUpdatedAt': sample_time,
            'cc': '',
            'bcc': '',
            'createdBy': {
                'id': 1000,
                'email': 'blabla@example.com',
            },
            'closedByUser': {
                'email': 'blabla@example.com',
            },
            'customerWaitingSince': {
                'time': sample_time
            },
            'source': {
                'type': 'blabla',
                'via': 'blabla'
            },
            'primaryCustomer': {
                'id': 1000,
                'email': 'blabla@example.com'
            },
```

```
            'assignee': {
                'id': 1000,
                'email': ''
            },
            'tags': [{
                'id': 1000
            }]
        }

        dict_tags = {1000 : 'blabla'}
        return data, dict_tags

    def base_hs_conv(self):
        sample_time = '1970-01-01T00:00:00+00:00'
        return {
            'number': 1000,
            'id_helpscout': 1000,
            'threads': 1000,
            'type': 'lolo',
            'folderId': 1000,
            'status': 'blabla',
            'state': 'blabla',
            'subject': 'blabla',
            'mailboxId': 1,
            'createdAt': sample_time,
            'closedBy': 'blabla',
            'closedAt': sample_time,
            'userUpdatedAt': sample_time,
            'cc': '',
            'bcc': '',
            'createdBy_id': 1000,
            'createdBy_email': 'blabla@example.com',
            'closedByUser_email': 'blabla@example.com',
            'customerWaitingSince_time': sample_time,
            'source_type': 'blabla',
            'source_via': 'blabla',
            'primaryCustomer_id': 1000,
            'primaryCustomer_email': 'blabla@example.com',
            'assignee_id': 1000,
            'assignee_email': '',
            'task_data_interval_start': sample_time,
```

```
            'task_data_interval_end': sample_time
        }

    def base_hs_conv_transformed(self):
        sample_time = '1970-01-01T00:00:00+00:00'
        return {
            'number': 1000,
            'id_helpscout': 1000,
            'threads': 1000,
            'type': 'lolo',
            'folder_id': 1000,
            'status': 'blabla',
            'state': 'blabla',
            'subject': 'blabla',
            'mailbox_id': 1,
            'created_at': sample_time,
            'closed_by': 'blabla',
            'closed_at': sample_time,
            'user_updated_at': sample_time,
            'cc': '',
            'bcc': '',
            'created_by_id': 1000,
            'created_by_email': 'blabla@example.com',
            'closed_by_user_email': 'blabla@example.com',
            'customer_waiting_since_time': sample_time,
            'source_type': 'blabla',
            'source_via': 'blabla',
            'primary_customer_id': 1000,
            'primary_customer_email': 'blabla@example.com',
            'assignee_id': 1000,
            'assignee_email': '',
            'task_data_interval_start': sample_time,
            'task_data_interval_end': sample_time
        }

    def object_to_dict(self, obj):
        return {
            c.name: getattr(obj, c.name).isoformat()
            if isinstance(getattr(obj, c.name),
             ↪  pendulum.DateTime)
            else getattr(obj, c.name)
```

```
            for c in obj.__table__.columns
        }

    def test__query_HS_clean_conversation(self):
        expected = [self.conv]
        result =
         ↪  self.session.query(HS_clean_conversation).all()
        self.assertEqual(result, expected)

    def test__query_HS_tag(self):
        expected = [self.tag]
        result = self.session.query(HS_tag).all()
        self.assertEqual(result, expected)

    def test__conv_id_not_nullable(self):
        result =
         ↪  self.session.query(HS_clean_conversation).all()
        assert_non_nullable(result[0], 'id')

    def test__tag_id_not_nullable(self):
        result = self.session.query(HS_tag).all()
        assert_non_nullable(result[0], 'id')

    def test__conv_number_nullable(self):
        result =
         ↪  self.session.query(HS_clean_conversation).all()
        assert_nullable(result[0], 'number')

    def test__tag_number_nullable(self):
        result = self.session.query(HS_tag).all()
        assert_nullable(result[0], 'name')

    def test__hs_clean_conversation_from_dict__base(self):
        self.maxDiff = None
        data, dict_tags = self.base_hs_data()
        start = '2022-01-01'
        end = '2022-01-01'
        hsconv = hs_clean_conversation_from_dict(data,
         ↪  dict_tags, start, end)

        expected = self.base_hs_conv_transformed()
```

```
        expected['task_data_interval_start'] =
         ↪  pendulum.parse(start).isoformat()
        expected['task_data_interval_end'] =
         ↪  pendulum.parse(end).isoformat()
        expected['id'] = None
        self.assertDictEqual(self.object_to_dict(hsconv),
         ↪  expected)

    def
     ↪  test__hs_clean_conversation_from_dict__customerWaitingSince_NoneTime
        self.maxDiff = None
        data, dict_tags = self.base_hs_data()
        data['customerWaitingSince'] = {}
        start = '2022-01-01'
        end = '2022-01-01'
        hsconv = hs_clean_conversation_from_dict(data,
         ↪  dict_tags, start, end)

        expected = self.base_hs_conv_transformed()
        expected['task_data_interval_start'] =
         ↪  pendulum.parse(start).isoformat()
        expected['task_data_interval_end'] =
         ↪  pendulum.parse(end).isoformat()
        expected['id'] = None
        expected['customer_waiting_since_time'] = None
        self.assertDictEqual(self.object_to_dict(hsconv),
         ↪  expected)
```

Tests

```
from sqlalchemy import Column, Integer, DateTime, String,
 ↪  ForeignKey, create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship
Base = declarative_base()

class HS_clean_conversation(Base):
    __tablename__ = 'hs_clean_conversation'
    id = Column(Integer, primary_key=True)
```

```
    number = Column(Integer)
    id_helpscout = Column(Integer)
    threads = Column(Integer)
    type = Column(String)
    folder_id = Column(Integer)
    status = Column(String)
    state = Column(String)
    subject = Column(String)
    mailbox_id = Column(Integer)
    created_at = Column(DateTime)
    closed_by = Column(Integer)
    closed_at = Column(DateTime)
    user_updated_at = Column(DateTime)
    cc = Column(String)
    bcc = Column(String)
    created_by_id = Column(Integer)
    created_by_email = Column(String)
    closed_by_user_email = Column(String)
    customer_waiting_since_time = Column(DateTime)
    source_type = Column(String)
    source_via = Column(String)
    primary_customer_id = Column(Integer)
    primary_customer_email = Column(String)
    assignee_id = Column(Integer)
    assignee_email = Column(String)
    task_data_interval_start = Column(DateTime, index=True)
    task_data_interval_end = Column(DateTime, index=True)
    tags = relationship('HS_tag',
     ↪  secondary='hs_conversation_tag')

class HS_tag(Base):
   __tablename__ = 'hs_tag'
   id = Column(Integer, primary_key = True)
   name = Column(String)
   clean_conversations =
    ↪  relationship(HS_clean_conversation,
    ↪  secondary='hs_conversation_tag')

class Conversation_tag(Base):
    __tablename__ = 'hs_conversation_tag'
```

```
    clean_conversation_id = Column(Integer,
     ↪  ForeignKey('hs_clean_conversation.id'), primary_key
     ↪  = True)
    tag_id = Column(Integer, ForeignKey('hs_tag.id'),
     ↪  primary_key = True)

#Per crear les taules
#engine = create_engine()
#Base.metadata.create_all(engine)
```

Classes del ORM

```
click>=8.0.3
python-helpscout-v2>=2.0.0
SQLAlchemy>=1.4.27
lxml>=4.7.1
pandas>=1.3.4
python-helpscout-v2>=2.0.0
pandas>=1.3.4
numpy>=1.21.4
openpyxl>=3.0.9
pendulum>=2.1.2
psycopg2-binary>=2.9.3
pandera>=0.11.0
SQLAlchemy-Utils>=0.38.2
```

Requeriments del projecte

```
FROM python:3.8-slim-buster

RUN apt-get update
RUN apt-get install -y wget
RUN wget
 ↪  https://raw.githubusercontent.com/Som-Energia/somenergia-kpis/main/requi

RUN pip3 install -r requirements.txt
```

Dockerfile

Bibliografia

[Airflow 022] Apache Airflow. *DAG*, (Accessed: June 2022). Available at https://airflow.apache.org/docs/apache-airflow/stable/concepts/dags.html. (Cited on page 11.)

[Apache 022] Apache. *Apache Airflow*, (Accessed: May 2022). Available at https://github.com/apache/airflow. (Cited on page 15.)

[Astronomer 022a] Astronomer. *DAG writing best practices in Apache Airflow*, (Accessed: June 2022). Available at https://www.astronomer.io/guides/dag-best-practices/. (Cited on page 23.)

[Astronomer 022b] Astronomer. *PythonVirtualenvOperator*, (Accessed: June 2022). Available at https://registry.astronomer.io/providers/apache-airflow/modules/pythonvirtualenvoperator. (Cited on page 23.)

[Foundation 022a] Free Software Foundation. *GNU Wget*, (Accessed: August 2022). Available at https://www.gnu.org/software/wget/. (Cited on page 17.)

[Foundation 022b] Python Software Foundation. *Python FAQ*, (Accessed: May 2022). Available at https://docs.python.org/3/faq/general.html. (Cited on page 11.)

[Foundation 022c] The Apache Software Foundation. *Introduction | Superset*, (Accessed: August 2022). Available at https://superset.apache.org/docs/intro/. (Cited on page 17.)

[Group 022] The PostgreSQL Global Development Group. *postgres*, (Accessed: June 2022). Available at https://www.postgresql.org/about/. (Cited on page 15.)

[Inc. 022a] Docker Inc. *Overview of Docker Compose*, (Accessed: August 2022). Available at https://docs.docker.com/compose/. (Cited on page 17.)

[Inc. 022b] Docker Inc. *Overview of Docker Compose*, (Accessed: August 2022). Available at https://docs.docker.com/compose/. (Cited on page 17.)

[Joiner 022] Britt Joiner. *How to use Trello for Scrum (and better teamwork)*, (Accessed: August 2022). Available at https://blog.trello.com/how-to-scrum-and-trello-for-teams-at-work?hsLang=en. (Cited on page 18.)

[Muro 022] José Antonio Muro. *¿Qué es un ORM?*, (Accessed: July 2022). Available at https://www2.deloitte.com/es/es/pages/technology/articles/que-es-orm.html. (Cited on page 11.)

[OCU 022] OCU. *El 22% de los usuarios ha tenido algún problema con su compañía de electricidad en el último año*, (Accessed: July 2022). Available at https://www.ocu.org/organizacion/prensa/notas-de-prensa/2019/encuestaelectricidad291019. (Cited on page 2.)

[RedHat 022] RedHat. *What is Docker?*, (Accessed: August 2022). Available at https://www.redhat.com/en/topics/containers/what-is-docker. (Cited on page 16.)

[school 022] IEBS school. *¿Qué es Data Lake y para qué sirve?*, (Accessed: August 2022). Available at https://www.iebschool.com/blog/data-lake-big-data/. (Cited on page 12.)

[TimescaleDocs 022] TimescaleDocs. *TimescaleDB Overview*, (Accessed: August 2022). Available at https://docs.timescale.com/timescaledb/latest/overview/#supercharged-postgresql. (Cited on page 15.)

[Wikipedia 022a] Wikipedia. *Extract,Transform,Load*, (Accessed: August 2022). Available at https://en.wikipedia.org/wiki/Extract,_transform,_load. (Cited on page 12.)

[Wikipedia 022b] Wikipedia. *Apache Airflow*, (Accessed: May 2022). Available at https://en.wikipedia.org/wiki/Apache_Airflow. (Cited on page 15.)