# BigQuery IAM + Authorized Views

Eu montei este projeto para gerenciar acesso no BigQuery a partir de uma tabela de usuarios e expor dados sensiveis apenas via views mascaradas. Aqui deixo o passo a passo que realmente funcionou no meu ambiente case-de-engenharia-de-dados, apenas o que foi executado com sucesso.

---

GIT : https://github.com/dinx0/iassistant

## Como eu trago os dados (extracao/ingestao)

\*\*Fontes e formatos\*\*

* Fonte 1 (proximo do real): metrica de busca do meu IA Assistant via API do Google Cloud + extensao do Chrome. Esses eventos chegam ja normalizados pela API.
* Fonte 2 (ficticia para o projeto): base de usuarios em JSON (newline-delimited). 99% dos registros sao sinteticos.

\*\*Pipelines e modo de chegada\*\*

* \*\*Streaming (quase tempo real)\*\* para a tabela `phrase\_metrics`:
* a extensao envia eventos para minha API e eu faco \*\*streaming inserts\*\* direto no BigQuery (sem ETL intermediario). O dado ja vem tratado pela propria API, entao aplico so cast/normalizacao leve na view.
* modelo aqui e "padrao kappa (event-driven)": eventos entram continuamente e eu consulto sempre a versao mais recente no BQ.
* \*\*Lote (batch)\*\* para a tabela `user\_iassistant`:
* eu \*\*carrego JSON\*\* para o BigQuery usando load jobs (`bq load`), quando preciso renovar a base. E um fluxo de dimensao/carga inicial.
* sem orquestrador dedicado: uso scripts shell + `bq`/`jq`. As mascaras e regras de acesso ficam nas \*\*views\*\* e no \*\*IAM do dataset\*\*.

\*\*Por que assim\*\*

* Quero focar no controle de acesso e mascaramento no proprio BigQuery (menos componentes).
* Streaming direto evita ETL quando o produtor ja valida o schema.
* Batch em JSON resolve bem para carga inicial e dados ficticios do case.

---

## Arquitetura final (o que esta rodando)

* \*\*Projeto\*\*: `case-de-engenharia-de-dados`
* \*\*Dataset base\*\*: `845415315315` (mesma regiao usada em `sec\_v`)
* \*\*Dataset de views\*\*: `sec\_v` (criado/recriado na mesma regiao do dataset base)
* \*\*Tabelas base\*\*:
* `user\_iassistant` — PK logica: `USER\_ID (INT64)`
* `phrase\_metrics` — sem PK fisica; relaciona por `user\_id (STRING)`
* \*\*Views mascaradas (em producao)\*\*:
* `sec\_v.user\_iassistant\_v` — mascara email e lat/lon extraida de `REGION`
* `sec\_v.phrase\_metrics\_v` — mascara `PHRASE\_TEXT` e `SEARCH\_RESULT`
* \*\*Relacao\*\*: `USER\_IASSISTANT.USER\_ID (INT64)` <-> `PHRASE\_METRICS.user\_id (STRING)` via `CAST`.

---

## IAM via tabela (o que eu rodei)

Regra simples: USER\_ID 1..5 => READER, USER\_ID >= 6 => WRITER no dataset base 845415315315. Eu gero as entradas access[] com bq + jq e aplico sem duplicar.

# variaveis

export PROJECT\_ID="case-de-engenharia-de-dados"

export DATASET\_ID="845415315315"

export DATASET\_REF="$PROJECT\_ID:$DATASET\_ID"

export SRC\_TABLE="$PROJECT\_ID.$DATASET\_ID.user\_iassistant"

# loop a partir da tabela (evita duplicacoes)

bq query --nouse\_legacy\_sql --format=csv "

WITH base AS (

SELECT DISTINCT CAST(USER\_ID AS INT64) AS user\_id,

LOWER(TRIM(USER\_EMAIL)) AS email

FROM \`$SRC\_TABLE\`

WHERE USER\_EMAIL IS NOT NULL

)

SELECT email,

CASE WHEN user\_id BETWEEN 1 AND 5 THEN 'READER' ELSE 'WRITER' END AS role

FROM base

" | tail -n +2 | while IFS=, read -r email role; do

[[ -z "$email" || -z "$role" ]] && continue

bq show --format=prettyjson "$DATASET\_REF" > ds.tmp.json

jq --arg em "$email" --arg r "$role" \

'.access += [{"role":$r,"userByEmail":$em}] | .access |= (map(tojson)|unique|map(fromjson))' \

ds.tmp.json > ds.apply.json

bq update --source=ds.apply.json "$DATASET\_REF"

rm -f ds.tmp.json ds.apply.json

done

Nota: emails ficticios/inexistentes o BigQuery recusa (nao entram na policy).

---

## Views mascaradas (SQL que eu apliquei)

### `sec\_v.user\_iassistant\_v`

* `email\_mascarado`: primeira letra + \\*\\*\\* + dominio
* `lat\_mascarada` / `lon\_mascarada`: extraidas de `REGION` com regex e `ROUND(..., 2)`

CREATE OR REPLACE VIEW `case-de-engenharia-de-dados.sec\_v.user\_iassistant\_v` AS

SELECT

CAST(USER\_ID AS INT64) AS user\_id,

CONCAT(

SUBSTR(LOWER(TRIM(USER\_EMAIL)),1,1),

'\*\*\*',

REGEXP\_EXTRACT(LOWER(TRIM(USER\_EMAIL)), r'(@.\*)$')

) AS email\_mascarado,

ROUND(

IF(ARRAY\_LENGTH(REGEXP\_EXTRACT\_ALL(REGION, r'(-?[0-9]+(?:\.[0-9]+)?)'))>0,

CAST(REGEXP\_EXTRACT\_ALL(REGION, r'(-?[0-9]+(?:\.[0-9]+)?)')[OFFSET(0)] AS FLOAT64), NULL),

2

) AS lat\_mascarada,

ROUND(

IF(ARRAY\_LENGTH(REGEXP\_EXTRACT\_ALL(REGION, r'(-?[0-9]+(?:\.[0-9]+)?)'))>1,

CAST(REGEXP\_EXTRACT\_ALL(REGION, r'(-?[0-9]+(?:\.[0-9]+)?)')[OFFSET(1)] AS FLOAT64), NULL),

2

) AS lon\_mascarada

FROM `case-de-engenharia-de-dados.845415315315.user\_iassistant`;

### `sec\_v.phrase\_metrics\_v`

* `phrase\_text\_masked = SUBSTR(PHRASE\_TEXT, 1, 20) || '...'`
* `search\_result\_masked = SUBSTR(SEARCH\_RESULT, 1, 60) || '...'`

CREATE OR REPLACE VIEW `case-de-engenharia-de-dados.sec\_v.phrase\_metrics\_v` AS

SELECT

CAST(user\_id AS STRING) AS user\_id,

CAST(PHRASE\_METRICS AS FLOAT64) AS phrase\_metrics,

IFNULL(SUBSTR(PHRASE\_TEXT, 1, 20) || '...', NULL) AS phrase\_text\_masked,

CAST(PAGE\_METRICS AS INT64) AS page\_metrics,

CAST(PAGE\_NUMBER AS INT64) AS page\_number,

SEARCH\_ITEMS AS search\_items,

IFNULL(SUBSTR(SEARCH\_RESULT, 1, 60) || '...', NULL) AS search\_result\_masked,

CAST(DT\_CARGA AS INT64) AS dt\_carga

FROM `case-de-engenharia-de-dados.845415315315.phrase\_metrics`;

### Authorized View no dataset base (eu adicionei assim)

BASE="case-de-engenharia-de-dados:845415315315"

LOC=$(bq show --format=prettyjson "$BASE" | jq -r .location)

# garante dataset de views na mesma regiao

bq mk --dataset --location="$LOC" case-de-engenharia-de-dados:sec\_v 2>/dev/null || true

bq show --format=prettyjson "$BASE" > base.json

jq '.access += [

{"view":{"projectId":"case-de-engenharia-de-dados","datasetId":"sec\_v","tableId":"user\_iassistant\_v"}},

{"view":{"projectId":"case-de-engenharia-de-dados","datasetId":"sec\_v","tableId":"phrase\_metrics\_v"}}

] | .access |= (map(tojson)|unique|map(fromjson))' base.json > base\_patched.json

bq update --source=base\_patched.json "$BASE"

---

## Relacao e consultas

Chave logica: USER\_IASSISTANT.USER\_ID (INT64) <-> PHRASE\_METRICS.user\_id (STRING) via CAST.

SELECT

u.USER\_ID,

v.email\_mascarado,

p.phrase\_metrics,

p.phrase\_text\_masked,

p.search\_result\_masked,

p.page\_metrics,

p.page\_number,

p.dt\_carga

FROM `case-de-engenharia-de-dados.sec\_v.user\_iassistant\_v` AS v

JOIN `case-de-engenharia-de-dados.845415315315.user\_iassistant` AS u

ON v.user\_id = u.USER\_ID

JOIN `case-de-engenharia-de-dados.sec\_v.phrase\_metrics\_v` AS p

ON CAST(p.user\_id AS INT64) = u.USER\_ID

LIMIT 100;

---

## Como reproduzir rapido (copy/paste)

> pre requisitos: gcloud autenticado no projeto, bq e jq instalados; habilitar \*\*Data Access logs\*\* para BigQuery no Console (IAM e Admin > Audit Logs), e criar o dataset base 845415315315 previamente.

# 0) variaveis

export PROJECT\_ID=case-de-engenharia-de-dados

export BASE=case-de-engenharia-de-dados:845415315315

export VIEWS=case-de-engenharia-de-dados:sec\_v

# 1) detectar regiao do dataset base e recriar o dataset de views na mesma regiao

LOC=$(bq show --format=prettyjson "$BASE" | jq -r .location)

bq rm -f -d "$VIEWS" 2>/dev/null || true

bq mk --dataset --location="$LOC" "$VIEWS"

# 2) criar/atualizar as views mascaradas

bq query --use\_legacy\_sql=false --location="$LOC" <<'SQL'

CREATE OR REPLACE VIEW `case-de-engenharia-de-dados.sec\_v.user\_iassistant\_v` AS

SELECT

CAST(USER\_ID AS INT64) AS user\_id,

CONCAT(SUBSTR(LOWER(TRIM(USER\_EMAIL)),1,1),'\*\*\*',REGEXP\_EXTRACT(LOWER(TRIM(USER\_EMAIL)), r'(@.\*)$')) AS email\_mascarado,

ROUND(IF(ARRAY\_LENGTH(REGEXP\_EXTRACT\_ALL(REGION, r'(-?[0-9]+(?:\.[0-9]+)?)'))>0,

CAST(REGEXP\_EXTRACT\_ALL(REGION, r'(-?[0-9]+(?:\.[0-9]+)?)')[OFFSET(0)] AS FLOAT64), NULL),2) AS lat\_mascarada,

ROUND(IF(ARRAY\_LENGTH(REGEXP\_EXTRACT\_ALL(REGION, r'(-?[0-9]+(?:\.[0-9]+)?)'))>1,

CAST(REGEXP\_EXTRACT\_ALL(REGION, r'(-?[0-9]+(?:\.[0-9]+)?)')[OFFSET(1)] AS FLOAT64), NULL),2) AS lon\_mascarada

FROM `case-de-engenharia-de-dados.845415315315.user\_iassistant`;

SQL

bq query --use\_legacy\_sql=false --location="$LOC" <<'SQL'

CREATE OR REPLACE VIEW `case-de-engenharia-de-dados.sec\_v.phrase\_metrics\_v` AS

SELECT

CAST(user\_id AS STRING) AS user\_id,

CAST(PHRASE\_METRICS AS FLOAT64) AS phrase\_metrics,

IFNULL(SUBSTR(PHRASE\_TEXT, 1, 20) || '...', NULL) AS phrase\_text\_masked,

CAST(PAGE\_METRICS AS INT64) AS page\_metrics,

CAST(PAGE\_NUMBER AS INT64) AS page\_number,

SEARCH\_ITEMS AS search\_items,

IFNULL(SUBSTR(SEARCH\_RESULT, 1, 60) || '...', NULL) AS search\_result\_masked,

CAST(DT\_CARGA AS INT64) AS dt\_carga

FROM `case-de-engenharia-de-dados.845415315315.phrase\_metrics`;

SQL

# 3) autorizar as views no dataset base

bq show --format=prettyjson "$BASE" > base.json

jq '.access += [

{"view":{"projectId":"case-de-engenharia-de-dados","datasetId":"sec\_v","tableId":"user\_iassistant\_v"}},

{"view":{"projectId":"case-de-engenharia-de-dados","datasetId":"sec\_v","tableId":"phrase\_metrics\_v"}}

] | .access |= (map(tojson)|unique|map(fromjson))' base.json > base\_patched.json

bq update --source=base\_patched.json "$BASE"

# 4) criar SA de leitura e testar impersonation

SA=bq-viewer-demo

EMAIL="$SA@$PROJECT\_ID.iam.gserviceaccount.com"

gcloud iam service-accounts create "$SA" 2>/dev/null || true

# papel de job no projeto (criar consultas)

gcloud projects add-iam-policy-binding "$PROJECT\_ID" \

--member="serviceAccount:$EMAIL" --role="roles/bigquery.jobUser" >/dev/null

# acesso somente no dataset de views

bq update --dataset\_access add:serviceAccount:$EMAIL:READER "$VIEWS"

# garantir que NAO tem acesso na base

bq update --dataset\_access remove:serviceAccount:$EMAIL "$BASE" 2>/dev/null || true

# testar como SA

MEU=$(gcloud config get-value account)

gcloud iam service-accounts add-iam-policy-binding "$EMAIL" \

--member="user:$MEU" --role="roles/iam.serviceAccountTokenCreator" >/dev/null

gcloud config set auth/impersonate\_service\_account "$EMAIL" >/dev/null

# deve falhar na base

bq query --nouse\_legacy\_sql 'SELECT COUNT(1) FROM `case-de-engenharia-de-dados.845415315315.user\_iassistant`' || true

# deve funcionar na view

bq query --nouse\_legacy\_sql 'SELECT \* FROM `case-de-engenharia-de-dados.sec\_v.user\_iassistant\_v` LIMIT 5'

# desfazer impersonation

gcloud config unset auth/impersonate\_service\_account >/dev/null

> se o seu projeto estiver em outro local, ajuste PROJECT\_ID, datasets e os nomes das views.

---

## Validacao (como eu testei)

* Criei a SA `bq-viewer-demo@case-de-engenharia-de-dados.iam.gserviceaccount.com` com:
* `READER` \*\*so\*\* em `sec\_v`
* `roles/bigquery.jobUser` no \*\*projeto\*\* (para criar jobs)
* Removi qualquer acesso dessa SA no dataset base
* Resultado: na base deu \*\*Access Denied\*\* (como esperado) e nas views funcionou com os dados mascarados

---

## Observabilidade e logs (o que eu configurei e validei)

Eu queria saber se alguem leu a tabela \*\*base\*\* direto (bypass da view). Entao primeiro validei o filtro no formato novo BigQueryAuditMetadata, depois criei a metrica. Importante: habilitar \*\*Data Access logs\*\* de BigQuery no projeto.

### Pre check (CLI)

gcloud logging read \

'logName="projects/case-de-engenharia-de-dados/logs/cloudaudit.googleapis.com%2Fdata\_access" \

AND resource.type="bigquery\_dataset" \

AND protoPayload.metadata."@type"="type.googleapis.com/google.cloud.audit.BigQueryAuditMetadata" \

AND protoPayload.metadata.tableDataRead.reason:\* \

AND protoPayload.resourceName:"projects/case-de-engenharia-de-dados/datasets/845415315315/tables/user\_iassistant"' \

--limit=5 \

--format='value(timestamp, protoPayload.metadata.tableDataRead.reason, protoPayload.resourceName)'

### Criacao da metrica

gcloud logging metrics create read\_base\_dataset\_metric \

--description="Leituras na tabela base (bypass da view)" \

--log-filter='logName="projects/case-de-engenharia-de-dados/logs/cloudaudit.googleapis.com%2Fdata\_access" \

AND resource.type="bigquery\_dataset" \

AND protoPayload.metadata."@type"="type.googleapis.com/google.cloud.audit.BigQueryAuditMetadata" \

AND protoPayload.metadata.tableDataRead.reason:\* \

AND protoPayload.resourceName:"projects/case-de-engenharia-de-dados/datasets/845415315315/tables/user\_iassistant"'

### Status (deu certo no meu projeto)

* Habilitei \*\*Data Access logs\*\* para BigQuery (e BigQuery Storage API) no projeto.
* Os comandos `gcloud logging read` retornaram eventos de leitura da tabela base, validando o filtro.
* Ao criar a metrica recebi `subject of a conflict` porque ela \*\*ja existia\*\*; a metrica `read\_base\_dataset\_metric` ja existia e permanece ativa.

### Regras extras de logging/alerta (recomendado)

Abaixo deixo mais regras que eu apliquei/validei para ampliar a cobertura de auditoria e justificar auditoria:

#### 1) Quem referenciou a tabela base em consultas (mesmo via JOIN)

\*\*Por que\*\*: pega consultas que citam a base; util para diferenciar "via view" de "via tabela".

gcloud logging metrics create query\_ref\_base\_table\_metric \

--description="Queries que referenciam a tabela base" \

--log-filter='logName="projects/case-de-engenharia-de-dados/logs/cloudaudit.googleapis.com%2Fdata\_access" \

AND resource.type="bigquery\_project" \

AND protoPayload.metadata."@type"="type.googleapis.com/google.cloud.audit.BigQueryAuditMetadata" \

AND protoPayload.metadata.jobCompletedEvent.job.jobStatistics.referencedTables:"projects/case-de-engenharia-de-dados/datasets/845415315315/tables/user\_iassistant"'

#### 2) Leituras que bateram na \*\*view\*\* (conforme esperado)

\*\*Por que\*\*: monitora o caminho certo de consumo (autorizado/mascarado).

gcloud logging metrics create read\_via\_view\_metric \

--description="Leituras na authorized view" \

--log-filter='logName="projects/case-de-engenharia-de-dados/logs/cloudaudit.googleapis.com%2Fdata\_access" \

AND resource.type="bigquery\_dataset" \

AND protoPayload.metadata."@type"="type.googleapis.com/google.cloud.audit.BigQueryAuditMetadata" \

AND protoPayload.metadata.tableDataRead.reason:\* \

AND protoPayload.resourceName:"projects/case-de-engenharia-de-dados/datasets/sec\_v/tables/user\_iassistant\_v"'

#### 3) DML/ingest na tabela base

\*\*Por que\*\*: rastrear escrita e volume.

gcloud logging metrics create base\_table\_dml\_metric \

--description="Mudancas de dados na tabela base" \

--log-filter='logName="projects/case-de-engenharia-de-dados/logs/cloudaudit.googleapis.com%2Fdata\_access" \

AND resource.type="bigquery\_dataset" \

AND protoPayload.metadata."@type"="type.googleapis.com/google.cloud.audit.BigQueryAuditMetadata" \

AND protoPayload.metadata.tableDataChange:\* \

AND protoPayload.resourceName:"projects/case-de-engenharia-de-dados/datasets/845415315315/tables/user\_iassistant"'

#### 4) Mudancas de schema/permissao no dataset base

\*\*Por que\*\*: detectar alteracoes de schema e IAM fora do pipeline controlado.

gcloud logging metrics create base\_schema\_or\_policy\_change\_metric \

--description="Alteracoes de schema/IAM no dataset base" \

--log-filter='logName="projects/case-de-engenharia-de-dados/logs/cloudaudit.googleapis.com%2Fdata\_access" \

AND resource.type="bigquery\_dataset" \

AND protoPayload.metadata."@type"="type.googleapis.com/google.cloud.audit.BigQueryAuditMetadata" \

AND (protoPayload.metadata.tableMetadataChange:\* OR protoPayload.metadata.datasetChange:\*) \

AND protoPayload.resourceName:"projects/case-de-engenharia-de-dados/datasets/845415315315"'

#### 5) Alertas em cima das metricas (ex.: leu a base)

\*\*Por que\*\*: acionar time quando houver bypass da view.

CHANNEL="projects/case-de-engenharia-de-dados/notificationChannels/XXXXXXXXXXXX" # substitua

POLICY\_BODY=$(cat <<'JSON'

{

"displayName": "Alerta: leitura direta na base",

"combiner": "OR",

"conditions": [

{

"displayName": "read\_base\_dataset\_metric > 0",

"conditionThreshold": {

"filter": "metric.type=\"logging.googleapis.com/user/read\_base\_dataset\_metric\"",

"comparison": "COMPARISON\_GT",

"thresholdValue": 0,

"duration": "300s"

}

}

],

"notificationChannels": ["CHANNEL\_PLACEHOLDER"]

}

JSON

)

# gcloud alpha monitoring policies create --policy "$POLICY\_BODY" | sed "s/CHANNEL\_PLACEHOLDER/$CHANNEL/"

\*(eu deixei o create comentado para voce so habilitar quando ja tiver o canal de notificacao pronto)\*

#### 6) Retencao e analytics dos logs

\*\*Por que\*\*: manter trilha por mais tempo e permitir SQL em logs.

# opcional: bucket dedicado

gcloud logging buckets create bq-security --location=global

# opcional: sink dos data\_access para um dataset BigQuery

bq --location=US mk -d --description "Audit logs do projeto" audit\_logs 2>/dev/null || true

SVC=$(gcloud logging sinks create sink-bq-audit \

bigquery.googleapis.com/projects/case-de-engenharia-de-dados/datasets/audit\_logs \

--log-filter='logName="projects/case-de-engenharia-de-dados/logs/cloudaudit.googleapis.com%2Fdata\_access"' \

--format='value(writerIdentity)')

# conceder acesso no dataset para o SA do sink

bq update --dataset\_access add:group:projectReaders audit\_logs 2>/dev/null || true

# (se preferir, conceda ao writerIdentity retornado acima: roles/bigquery.dataEditor)

#### 7) Falhas de permissao (PERMISSION\\_DENIED)

\*\*Por que\*\*: captura tentativas fora do fluxo (ex.: usuario/SA tentando ler base sem permissao). Ajuda a evidenciar bypass bloqueado.

gcloud logging metrics create bq\_permission\_denied\_metric \

--description="Tentativas PERMISSION\_DENIED no BigQuery" \

--log-filter='logName="projects/case-de-engenharia-de-dados/logs/cloudaudit.googleapis.com%2Factivity" \

AND protoPayload.serviceName="bigquery.googleapis.com" \

AND protoPayload.status.code=7'

> Observacao: para focar na sua base, complemente com AND protoPayload.resourceName:"projects/case-de-engenharia-de-dados/datasets/845415315315".

#### 8) Consulta pesada (custo)

\*\*Por que\*\*: sinaliza consultas com alto totalBilledBytes (custo/risco). Bom para controlar SLAs e budget.

gcloud logging metrics create heavy\_query\_metric \

--description="Queries com totalBilledBytes >= 1GB" \

--log-filter='logName="projects/case-de-engenharia-de-dados/logs/cloudaudit.googleapis.com%2Fdata\_access" \

AND resource.type="bigquery\_project" \

AND protoPayload.metadata."@type"="type.googleapis.com/google.cloud.audit.BigQueryAuditMetadata" \

AND protoPayload.metadata.jobCompletedEvent.eventName="job\_completed" \

AND protoPayload.metadata.jobCompletedEvent.job.jobStatistics.totalBilledBytes>=1000000000'

Opcional: alerta simples para quando ocorrer (substitua o canal):

CHANNEL="projects/case-de-engenharia-de-dados/notificationChannels/XXXXXXXXXXXX"

POLICY=$(cat <<'JSON'

{

"displayName": "Alerta: consulta pesada (>=1GB)",

"combiner": "OR",

"conditions": [

{

"displayName": "heavy\_query\_metric > 0",

"conditionThreshold": {

"filter": "metric.type=\"logging.googleapis.com/user/heavy\_query\_metric\"",

"comparison": "COMPARISON\_GT",

"thresholdValue": 0,

"duration": "300s"

}

}

],

"notificationChannels": ["CHANNEL\_PLACEHOLDER"]

}

JSON

)

# gcloud alpha monitoring policies create --policy "$POLICY" | sed "s/CHANNEL\_PLACEHOLDER/$CHANNEL/"

\*\*Resumo do "por que"\*\*

* Base vs View: separar metricas ajuda a auditar desvio de consumo.
* DML e schema/IAM: garantem trilha de mudanca e responsabilizacao.
* Alerting: fecha o loop com acao (pager/email) quando algo anormal acontecer.
* Retencao/analytics: permite forense com SQL em cima de logs.

---

## Seguranca e privacidade (em uso)

* \*\*Authorized Views\*\*: consumidores so leem as views `sec\_v.\*`, onde aplico mascaras de email e lat/lon e redacao de texto.
* \*\*IAM enxuto\*\*: a service account de leitura tem permissao apenas no `sec\_v` e papel `bigquery.jobUser` no projeto para executar jobs; sem acesso ao dataset base.
* \*\*Data Access Logs\*\*: habilitados para auditar `tableDataRead`, DML, mudancas de schema e IAM.
* \*\*Sensitive Data Protection (DLP)\*\*: ativei a inspecao de dados no projeto para identificar padroes sensiveis (email, possiveis localizacoes). Uso como visibilidade/monitoramento, sem bloqueio automatico.

### Nao aplicado por escopo/limitacao do projeto

* \*\*Policy Tags / Data Catalog\*\*: criacao de DataPolicy exige organizacao; meu projeto e standalone, entao deixei como melhoria futura.
* \*\*Row-level Security (RLS)\*\*: tentativa de `CREATE ROW ACCESS POLICY` falhou por grupo inexistente; mantive o modelo com authorized view, que resolve a necessidade funcional.

---

## Estado atual (bq show do dataset base)

Comando que usei:

bq show case-de-engenharia-de-dados:845415315315

Resumo do que apareceu no meu ambiente (conferido via terminal):

* \*\*Authorized Views\*\*: `case-de-engenharia-de-dados:sec\_v.user\_iassistant\_v` listado (conforme esperado).
* \*\*Members (recorte)\*\*: varios emails validos, o grupo implicito `projectReaders` e a SA `bq-viewer-demo@case-de-engenharia-de-dados.iam.gserviceaccount.com`.
* \\*\\*Schema da tabela \\*\\*\`\`:
* `REGION`: STRING
* `USER\_EMAIL`: STRING
* `USER\_NAME`: STRING
* `USER\_ID`: INTEGER
* `DT\_CARGA`: INTEGER
* \*\*Particionamento\*\*: por `HOUR`, com `expirationMs=5184000000` (\~60 dias).
* \*\*Total rows\*\*: `1000`.
* \*\*Total logical bytes\*\*: \~`88 KB`.

Isso confirma que:

1. o dataset base esta com a authorized view ativa,
2. a tabela esta com schema/particionamento conforme esperado,
3. as entradas de IAM estao no lugar (incluindo a SA de viewer que usei para testar acesso somente nas views mascaradas).

---

## Como subir este README no GitHub (Cloud Shell)

Abaixo deixo o que eu usei para colocar este README no repo dinx0/iassistant.

### Pre requisitos

* ter o repo no GitHub: `https://github.com/dinx0/iassistant`
* estar logado no Cloud Shell

### 0) identidade do git (uma vez so)

git config --global user.name "Dih Oliver"

git config --global user.email "dih.oliver08@gmail.com"

### 1) se o repo ja estiver clonado

cd ~/iassistant # entre no diretorio do repo

nano README.md # cole/edite o conteudo deste arquivo

# salve (Ctrl+O, Enter) e saia (Ctrl+X)

git add README.md

git commit -m "docs: adiciona README do projeto (BigQuery + views + logging)"

### 2) se ainda nao tiver clonado

git clone https://github.com/dinx0/iassistant.git

cd iassistant

nano README.md

# salve e saia

git add README.md

git commit -m "docs: adiciona README do projeto (BigQuery + views + logging)"

### 3) autenticar e fazer o push (recomendado: GitHub CLI)

sudo apt-get update && sudo apt-get install -y gh

gh auth login -w # GitHub.com -> HTTPS -> abrir no navegador e confirmar

git branch -M main # garante que a branch e main

git push -u origin main

### Alternativas de autenticacao

* \*\*HTTPS com token (PAT)\*\*: gere um token com escopo `repo` em \*Settings -> Developer settings -> Personal access tokens\*. No `git push`, use `dinx0` como username e cole o token no campo de password.
* \*\*SSH\*\*: gere uma chave `ed25519` (`ssh-keygen -t ed25519 -C "dih.oliver08@gmail.com"`), adicione a publica em \*Settings -> SSH and GPG keys\*, e troque a URL remota:

git remote set-url origin git@github.com:dinx0/iassistant.git

git push -u origin main

### Dica: limpar credencial errada em cache (se pedir username esquisito)

git config --global --unset credential.helper 2>/dev/null || true

git credential reject <<EOF

protocol=https

host=github.com

EOF

### Conferencia

git log --oneline -n 3

git remote -v

# Case de Engenharia de Dados (BigQuery) — passo a passo final

Eu construí este case inteiro no projeto \*\*case-de-engenharia-de-dados\*\* para mostrar como:

* carregar bases (lote e streaming)
* publicar \*\*authorized views\*\* com \*\*mascara de dados\*\*
* controlar acesso somente pelas views
* auditar leitura direta na base via \*\*Data Access Logs\*\* + metricas
* monitorar atividade do \*\*GitHub\*\* via dataset `github\_activity`

Tudo aqui é o que eu realmente executei e funcionou. Sem firula.

> \*\*Regiao\*\*: meu dataset base fica na regiao southamerica-east1. O dataset github\_activity (do Marketplace) é sempre em \*\*US\*\*, entao as consultas nele precisam de --location=US.

---

## Visao geral (diagramas)

### Arquitetura

flowchart LR

subgraph GCP[Projeto: case-de-engenharia-de-dados]

subgraph BQ[BigQuery]

direction TB

BASE[(Dataset base

845415315315)]

VIEWS[(Dataset views

sec\_v)]

BASE -->|Authorized View| V1[view: sec\_v.user\_iassistant\_v]

BASE -->|Authorized View| V2[view: sec\_v.phrase\_metrics\_v]

end

LOGS[Cloud Logging

(Data Access logs)]

end

EXT1[Chrome Ext + API

(IASSISTANT)] -->|streaming| BASE

CSV[NDJSON/CSV

usuarios] -->|load job| BASE

LOGS -.audita leitura base.-> BASE

LOGS -.audita leitura view.-> V1

### Modelo (ER simples)

erDiagram

USER\_IASSISTANT {

INT64 USER\_ID PK

STRING USER\_NAME

STRING USER\_EMAIL

STRING REGION "lat,lon"

INT64 DT\_CARGA

}

PHRASE\_METRICS {

STRING user\_id FK

FLOAT64 PHRASE\_METRICS

STRING PHRASE\_TEXT

INT64 PAGE\_METRICS

INT64 PAGE\_NUMBER

STRING SEARCH\_ITEMS

STRING SEARCH\_RESULT

INT64 DT\_CARGA

}

USER\_IASSISTANT ||--o{ PHRASE\_METRICS : "user\_id = CAST(USER\_ID AS STRING)"

### Mascaras aplicadas

flowchart TD

A[USER\_EMAIL] -->|REGEXP\_REPLACE| AM[email\_mascarado

"f\*\*\*@dominio"]

R[REGION "lat,lon"] -->|regex + round(2)| LAT[lat\_mascarada]

R -->|regex + round(2)| LON[lon\_mascarada]

T[PHRASE\_TEXT] -->|SUBSTR(1,20)||> three dots| TM[phrase\_text\_masked]

S[SEARCH\_RESULT] -->|SUBSTR(1,60)||> three dots| SM[search\_result\_masked]

---

## 1) Datasets e tabelas

\*\*Projeto\*\*: case-de-engenharia-de-dados

\*\*Datasets\*\*

* Base: `845415315315` (regiao `southamerica-east1`)
* Views: `sec\_v` (mesma regiao do base)

\*\*Tabelas base\*\*

* `845415315315.user\_iassistant`

PK logica: USER\_ID (INT64)

* `845415315315.phrase\_metrics`

Relaciona em user\_id (STRING) com USER\_ID via CAST.

\*\*Como garanti o dataset de views na MESMA regiao do base\*\*

BASE="case-de-engenharia-de-dados:845415315315"

LOC=$(bq show --format=prettyjson "$BASE" | jq -r .location)

bq rm -f -d case-de-engenharia-de-dados:sec\_v 2>/dev/null || true

bq mk --dataset --location="$LOC" case-de-engenharia-de-dados:sec\_v

---

## 2) Views mascaradas (DDL usado)

### `sec\_v.user\_iassistant\_v`

CREATE OR REPLACE VIEW `case-de-engenharia-de-dados.sec\_v.user\_iassistant\_v` AS

SELECT

CAST(USER\_ID AS INT64) AS user\_id,

CONCAT(

SUBSTR(LOWER(TRIM(USER\_EMAIL)),1,1),

'\*\*\*',

REGEXP\_EXTRACT(LOWER(TRIM(USER\_EMAIL)), r'(@.\*)$')

) AS email\_mascarado,

ROUND(

IF(ARRAY\_LENGTH(REGEXP\_EXTRACT\_ALL(REGION, r'(-?[0-9]+(?:\.[0-9]+)?)'))>0,

CAST(REGEXP\_EXTRACT\_ALL(REGION, r'(-?[0-9]+(?:\.[0-9]+)?)')[OFFSET(0)] AS FLOAT64), NULL),

2

) AS lat\_mascarada,

ROUND(

IF(ARRAY\_LENGTH(REGEXP\_EXTRACT\_ALL(REGION, r'(-?[0-9]+(?:\.[0-9]+)?)'))>1,

CAST(REGEXP\_EXTRACT\_ALL(REGION, r'(-?[0-9]+(?:\.[0-9]+)?)')[OFFSET(1)] AS FLOAT64), NULL),

2

) AS lon\_mascarada

FROM `case-de-engenharia-de-dados.845415315315.user\_iassistant`;

### `sec\_v.phrase\_metrics\_v`

CREATE OR REPLACE VIEW `case-de-engenharia-de-dados.sec\_v.phrase\_metrics\_v` AS

SELECT

CAST(user\_id AS STRING) AS user\_id,

CAST(PHRASE\_METRICS AS FLOAT64) AS phrase\_metrics,

IFNULL(SUBSTR(PHRASE\_TEXT, 1, 20) || '...', NULL) AS phrase\_text\_masked,

CAST(PAGE\_METRICS AS INT64) AS page\_metrics,

CAST(PAGE\_NUMBER AS INT64) AS page\_number,

SEARCH\_ITEMS AS search\_items,

IFNULL(SUBSTR(SEARCH\_RESULT, 1, 60) || '...', NULL) AS search\_result\_masked,

CAST(DT\_CARGA AS INT64) AS dt\_carga

FROM `case-de-engenharia-de-dados.845415315315.phrase\_metrics`;

### Autorizar as views no dataset base

BASE="case-de-engenharia-de-dados:845415315315"

bq show --format=prettyjson "$BASE" > base.json

jq '.access += [

{"view":{"projectId":"case-de-engenharia-de-dados","datasetId":"sec\_v","tableId":"user\_iassistant\_v"}},

{"view":{"projectId":"case-de-engenharia-de-dados","datasetId":"sec\_v","tableId":"phrase\_metrics\_v"}}

] | .access |= (map(tojson)|unique|map(fromjson))' base.json > base\_patched.json

bq update --source=base\_patched.json "$BASE"

---

## 3) IAM dinamico por tabela (o que eu executei)

Regra: USER\_ID 1..5 => READER, >= 6 => WRITER no dataset base.

PROJECT\_ID="case-de-engenharia-de-dados"

DATASET\_ID="845415315315"

DATASET\_REF="$PROJECT\_ID:$DATASET\_ID"

SRC\_TABLE="$PROJECT\_ID.$DATASET\_ID.user\_iassistant"

bq query --nouse\_legacy\_sql --format=csv "

WITH base AS (

SELECT DISTINCT CAST(USER\_ID AS INT64) AS user\_id,

LOWER(TRIM(USER\_EMAIL)) AS email

FROM \`$SRC\_TABLE\`

WHERE USER\_EMAIL IS NOT NULL

)

SELECT email,

CASE WHEN user\_id BETWEEN 1 AND 5 THEN 'READER' ELSE 'WRITER' END AS role

FROM base

" | tail -n +2 | while IFS=, read -r email role; do

[[ -z "$email" || -z "$role" ]] && continue

bq show --format=prettyjson "$DATASET\_REF" > ds.tmp.json

jq --arg em "$email" --arg r "$role" \

'.access += [{"role":$r,"userByEmail":$em}] | .access |= (map(tojson)|unique|map(fromjson))' \

ds.tmp.json > ds.apply.json

bq update --source=ds.apply.json "$DATASET\_REF"

rm -f ds.tmp.json ds.apply.json

done

> Emails invalidos o BigQuery recusa; a lista final ficou so com emails validos.

---

## 4) Teste de seguranca (impersonation)

Criei uma SA \*\*somente leitora de views\*\* e testei com impersonation.

PROJECT\_ID=case-de-engenharia-de-dados

VIEWS=case-de-engenharia-de-dados:sec\_v

BASE=case-de-engenharia-de-dados:845415315315

SA=bq-viewer-demo

EMAIL="$SA@$PROJECT\_ID.iam.gserviceaccount.com"

gcloud iam service-accounts create "$SA" 2>/dev/null || true

# pode criar jobs

gcloud projects add-iam-policy-binding "$PROJECT\_ID" \

--member="serviceAccount:$EMAIL" --role="roles/bigquery.jobUser"

# leitura so no dataset de views

bq update --dataset\_access add:serviceAccount:$EMAIL:READER "$VIEWS"

# garante que NAO tem acesso ao base

bq update --dataset\_access remove:serviceAccount:$EMAIL "$BASE" 2>/dev/null || true

# permite eu impersonar

MEU=$(gcloud config get-value account)

gcloud iam service-accounts add-iam-policy-binding "$EMAIL" \

--member="user:$MEU" --role="roles/iam.serviceAccountTokenCreator"

gcloud config set auth/impersonate\_service\_account "$EMAIL"

# deve falhar

bq query --nouse\_legacy\_sql 'SELECT COUNT(\*) FROM `case-de-engenharia-de-dados.845415315315.user\_iassistant`' || true

# deve funcionar

bq query --nouse\_legacy\_sql 'SELECT \* FROM `case-de-engenharia-de-dados.sec\_v.user\_iassistant\_v` LIMIT 5'

gcloud config unset auth/impersonate\_service\_account

---

## 5) Observabilidade e controle (logs + metricas)

Primeiro \*\*habilitei Data Access logs\*\* de BigQuery (Console > IAM e Admin > Audit Logs > BigQuery > marcar Data Read/Write).

### Validar leitura direta na base (CLI)

gcloud logging read \

'logName="projects/case-de-engenharia-de-dados/logs/cloudaudit.googleapis.com%2Fdata\_access" \

AND resource.type="bigquery\_dataset" \

AND protoPayload.metadata."@type"="type.googleapis.com/google.cloud.audit.BigQueryAuditMetadata" \

AND protoPayload.metadata.tableDataRead.reason:\* \

AND protoPayload.resourceName:"projects/case-de-engenharia-de-dados/datasets/845415315315/tables/user\_iassistant"' \

--limit=5 \

--format='value(timestamp, protoPayload.metadata.tableDataRead.reason, protoPayload.resourceName)'

### Metricas que eu criei

\*\*1) Leitura direta na base\*\*

gcloud logging metrics create read\_base\_dataset\_metric \

--description="Leituras na tabela base (bypass da view)" \

--log-filter='logName="projects/case-de-engenharia-de-dados/logs/cloudaudit.googleapis.com%2Fdata\_access" \

AND resource.type="bigquery\_dataset" \

AND protoPayload.metadata."@type"="type.googleapis.com/google.cloud.audit.BigQueryAuditMetadata" \

AND protoPayload.metadata.tableDataRead.reason:\* \

AND protoPayload.resourceName:"projects/case-de-engenharia-de-dados/datasets/845415315315/tables/user\_iassistant"'

\*\*2) Consultas que referenciam a tabela base\*\*

gcloud logging metrics create query\_ref\_base\_table\_metric \

--description="Queries que referenciam a tabela base" \

--log-filter='logName="projects/case-de-engenharia-de-dados/logs/cloudaudit.googleapis.com%2Fdata\_access" \

AND resource.type="bigquery\_project" \

AND protoPayload.metadata."@type"="type.googleapis.com/google.cloud.audit.BigQueryAuditMetadata" \

AND protoPayload.metadata.jobCompletedEvent.job.jobStatistics.referencedTables:"projects/case-de-engenharia-de-dados/datasets/845415315315/tables/user\_iassistant"'

\*\*3) Leituras via authorized view\*\*

gcloud logging metrics create read\_via\_view\_metric \

--description="Leituras na authorized view" \

--log-filter='logName="projects/case-de-engenharia-de-dados/logs/cloudaudit.googleapis.com%2Fdata\_access" \

AND resource.type="bigquery\_dataset" \

AND protoPayload.metadata."@type"="type.googleapis.com/google.cloud.audit.BigQueryAuditMetadata" \

AND protoPayload.metadata.tableDataRead.reason:\* \

AND protoPayload.resourceName:"projects/case-de-engenharia-de-dados/datasets/sec\_v/tables/user\_iassistant\_v"'

\*\*4) DML na base\*\*

gcloud logging metrics create base\_table\_dml\_metric \

--description="Mudancas de dados na tabela base" \

--log-filter='logName="projects/case-de-engenharia-de-dados/logs/cloudaudit.googleapis.com%2Fdata\_access" \

AND resource.type="bigquery\_dataset" \

AND protoPayload.metadata."@type"="type.googleapis.com/google.cloud.audit.BigQueryAuditMetadata" \

AND protoPayload.metadata.tableDataChange:\* \

AND protoPayload.resourceName:"projects/case-de-engenharia-de-dados/datasets/845415315315/tables/user\_iassistant"'

\*\*5) Mudanca de schema/IAM\*\*

gcloud logging metrics create base\_schema\_or\_policy\_change\_metric \

--description="Alteracoes de schema/IAM no dataset base" \

--log-filter='logName="projects/case-de-engenharia-de-dados/logs/cloudaudit.googleapis.com%2Fdata\_access" \

AND resource.type="bigquery\_dataset" \

AND protoPayload.metadata."@type"="type.googleapis.com/google.cloud.audit.BigQueryAuditMetadata" \

AND (protoPayload.metadata.tableMetadataChange:\* OR protoPayload.metadata.datasetChange:\*) \

AND protoPayload.resourceName:"projects/case-de-engenharia-de-dados/datasets/845415315315"'

> Em producao: configure alertas no Cloud Monitoring apontando para essas metricas (email/pager).

---

## 6) Qualidade de dados (checks simples que eu rodei)

\*\*Comparar contagem base vs view\*\*

-- deve bater

SELECT COUNT(\*) FROM `case-de-engenharia-de-dados.845415315315.user\_iassistant`;

SELECT COUNT(\*) FROM `case-de-engenharia-de-dados.sec\_v.user\_iassistant\_v`;

\*\*Verificar mascara de email\*\*

SELECT USER\_EMAIL, email\_mascarado

FROM `case-de-engenharia-de-dados.845415315315.user\_iassistant` u

JOIN `case-de-engenharia-de-dados.sec\_v.user\_iassistant\_v` v

ON v.user\_id = u.USER\_ID

LIMIT 20;

\*\*Validar parse de latitude/longitude\*\*

SELECT REGION, lat\_mascarada, lon\_mascarada

FROM `case-de-engenharia-de-dados.sec\_v.user\_iassistant\_v`

WHERE (lat\_mascarada IS NULL OR lon\_mascarada IS NULL)

LIMIT 20;

\*\*Campos mascarados na phrase\_metrics\_v\*\*

SELECT phrase\_text\_masked, search\_result\_masked

FROM `case-de-engenharia-de-dados.sec\_v.phrase\_metrics\_v`

LIMIT 20;

---

## 7) Monitor de atividade do GitHub (GitHub Activity Data)

> Este dataset sempre fica em \*\*US\*\*. Se nao existir, crie o dataset manualmente:

> bq --location=US mk -d case-de-engenharia-de-dados:github\_activity

\*\*Eventos por tipo (30 dias)\*\*

SELECT type, COUNT(\*) AS total

FROM `case-de-engenharia-de-dados.github\_activity.events`

WHERE repo.name = 'dinx0/iassistant'

AND created\_at >= TIMESTAMP\_SUB(CURRENT\_TIMESTAMP(), INTERVAL 30 DAY)

GROUP BY type

ORDER BY total DESC;

\*\*Commits por dia e autor (PushEvent)\*\*

SELECT

DATE(created\_at) AS dia,

actor.login AS pusher,

COUNT(1) AS commits

FROM `case-de-engenharia-de-dados.github\_activity.events`,

UNNEST(payload.commits) c

WHERE repo.name = 'dinx0/iassistant'

AND type = 'PushEvent'

AND created\_at >= TIMESTAMP\_SUB(CURRENT\_TIMESTAMP(), INTERVAL 30 DAY)

GROUP BY dia, pusher

ORDER BY dia DESC, commits DESC;

\*\*Ultimos 20 commits (mensagem/autor/sha)\*\*

SELECT

c.sha,

c.author.name AS author\_name,

c.author.email AS author\_email,

c.message,

created\_at

FROM `case-de-engenharia-de-dados.github\_activity.events`,

UNNEST(payload.commits) c

WHERE repo.name = 'dinx0/iassistant'

AND type = 'PushEvent'

ORDER BY created\_at DESC

LIMIT 20;

\*\*PRs abertos/mergeados/fechados (90 dias)\*\*

SELECT

COUNTIF(payload.action = 'opened') AS prs\_abertos,

COUNTIF(payload.action = 'closed' AND payload.pull\_request.merged) AS prs\_mergeados,

COUNTIF(payload.action = 'closed' AND NOT payload.pull\_request.merged) AS prs\_fechados\_sem\_merge

FROM `case-de-engenharia-de-dados.github\_activity.events`

WHERE repo.name = 'dinx0/iassistant'

AND type = 'PullRequestEvent'

AND created\_at >= TIMESTAMP\_SUB(CURRENT\_TIMESTAMP(), INTERVAL 90 DAY);

\*\*Lead time de PR (criado -> merge)\*\*

SELECT

payload.pull\_request.number AS pr\_number,

payload.pull\_request.title AS title,

TIMESTAMP\_DIFF(payload.pull\_request.merged\_at,

payload.pull\_request.created\_at, HOUR) AS lead\_time\_horas

FROM `case-de-engenharia-de-dados.github\_activity.events`

WHERE repo.name = 'dinx0/iassistant'

AND type = 'PullRequestEvent'

AND payload.action = 'closed'

AND payload.pull\_request.merged

ORDER BY lead\_time\_horas DESC

LIMIT 20;

\*\*Issues por acao\*\*

SELECT payload.action, COUNT(\*) AS total

FROM `case-de-engenharia-de-dados.github\_activity.events`

WHERE repo.name = 'dinx0/iassistant'

AND type = 'IssuesEvent'

GROUP BY payload.action

ORDER BY total DESC;

\*\*Stars e forks\*\*

-- stars

SELECT DATE(created\_at) AS dia, COUNT(\*) AS stars

FROM `case-de-engenharia-de-dados.github\_activity.events`

WHERE repo.name = 'dinx0/iassistant' AND type = 'WatchEvent'

GROUP BY dia ORDER BY dia DESC;

-- forks

SELECT DATE(created\_at) AS dia, COUNT(\*) AS forks

FROM `case-de-engenharia-de-dados.github\_activity.events`

WHERE repo.name = 'dinx0/iassistant' AND type = 'ForkEvent'

GROUP BY dia ORDER BY dia DESC;

Dica: ao usar bq nessas consultas, inclua --location=US.

---

## 8) Exportar e versionar infra do BigQuery no repo

\*\*Export (ambiente -> git)\*\*

PROJECT=case-de-engenharia-de-dados

BASE\_DS=845415315315

VIEWS\_DS=sec\_v

mkdir -p sql/views infra/bq schema data/samples

bq show --format=prettyjson $PROJECT:$VIEWS\_DS.user\_iassistant\_v | jq -r .view.query > sql/views/user\_iassistant\_v.sql

bq show --format=prettyjson $PROJECT:$VIEWS\_DS.phrase\_metrics\_v | jq -r .view.query > sql/views/phrase\_metrics\_v.sql

bq show --format=prettyjson $PROJECT:$BASE\_DS | jq '{access}' > infra/bq/dataset\_access.json

bq show --schema --format=prettyjson $PROJECT:$BASE\_DS.user\_iassistant > schema/user\_iassistant.schema.json

bq show --schema --format=prettyjson $PROJECT:$BASE\_DS.phrase\_metrics > schema/phrase\_metrics.schema.json

\*\*Apply (git -> ambiente)\*\*

PROJECT=case-de-engenharia-de-dados

BASE\_DS=845415315315

VIEWS\_DS=sec\_v

BASE="$PROJECT:$BASE\_DS"; VIEWS="$PROJECT:$VIEWS\_DS"

LOC=$(bq show --format=prettyjson "$BASE" | jq -r .location)

bq mk --dataset --location="$LOC" "$VIEWS" 2>/dev/null || true

jq '{access}' infra/bq/dataset\_access.json > /tmp/ds\_access.json

bq show --format=prettyjson "$BASE" | jq '.access = (input.access)' /tmp/ds\_access.json > /tmp/base\_apply.json

bq update --source=/tmp/base\_apply.json "$BASE"

bq query --use\_legacy\_sql=false --location="$LOC" < sql/views/user\_iassistant\_v.sql

bq query --use\_legacy\_sql=false --location="$LOC" < sql/views/phrase\_metrics\_v.sql

\*\*Scripts utilitarios\*\*

* `scripts/export\_bq.sh` e `scripts/apply\_bq.sh` (mesmas linhas acima em shell, prontos para automatizar).

---

## 9) Publicar/atualizar este README no GitHub (Cloud Shell)

# identidades do git

git config --global user.name "Dih Oliver"

git config --global user.email "dih.oliver08@gmail.com"

# clonar (se ainda nao tiver)

git clone https://github.com/dinx0/iassistant.git

cd iassistant

# editar README.md

nano README.md

git add README.md

git commit -m "docs: atualiza README (BigQuery + views + logs + github activity)"

# autenticar e enviar (via GitHub CLI)

sudo apt-get update && sudo apt-get install -y gh

gh auth login -w

git branch -M main

git push -u origin main

Se pedir usuario/senha via HTTPS: use \*\*Personal Access Token\*\* como senha, ou troque o remoto para \*\*SSH\*\*.

---

## 10) Politicas adicionais e limites do case

* Ativei \*\*Sensitive Data Protection (DLP)\*\* no projeto para inspecao (sem bloqueios automaticos).
* \*\*Policy Tags/Data Policies\*\*: deixei como melhoria futura (exige org). Mantive \*\*authorized view\*\* como mecanismo principal de mascaramento.
* \*\*Row-level Security\*\*: tentativas com grupos nao existentes falharam; mantive apenas authorized view.

---

### Conclusao

O case entrega:

* mascaramento de PII na borda de consumo (views)
* isolamento de acesso (dataset `sec\_v` vs base)
* auditoria de bypass (metricas + logs)
* monitor de atividade do repo GitHub com queries prontas
* scripts para exportar/aplicar infra no BigQuery e versionar no git

Qualquer ajuste futuro: adicionar Policy Tags, RLS por grupo real e CI/CD via Actions/Workload Identity para aplicar as mudancas de sql/ e infra/ automaticamente.

# IAssistant

\*\*IAssistant\*\* é uma extensão Chrome desenvolvida para auxiliar na análise de resultados de buscas por meio de tokenização. A extensão calcula percentuais de similaridade entre a consulta realizada e os resultados obtidos, permitindo a visualização de métricas e gráficos que podem ser integrados a um serviço em nuvem (como o BigQuery).

## Funcionalidades

* \*\*Tokenização & Análise de Similaridade:\*\*

A extensão utiliza tokenização para comparar a pesquisa realizada com os resultados obtidos, calculando percentuais de similaridade.

* \*\*Interface Dual – Popup e Side Panel:\*\*

Utiliza um arquivo index.html para exibir a interface principal, disponível tanto como popup quanto como painel lateral (side panel) no Chrome.

* \*\*Comandos Rápidos e Omnibox:\*\*
* \*\*Atalho:\*\* Pressione `Ctrl+B` (ou `Command+B` no Mac) para executar a ação principal.
* \*\*Omnibox:\*\* Digite `assistente` na barra de endereços para interagir com a extensão.
* \*\*Visualização de Gráficos e Métricas:\*\*

Ao executar os comandos show graphics ou show metrics, a extensão realiza requisições para exibir gráficos e métricas detalhadas. (\_Nota: Caso os endpoints ainda estejam configurados para localhost, ocorrerão erros de conexão em ambientes de produção.\_)

IAssistant/ ├── images/ │ ├── icon-16.png │ ├── icon-32.png │ ├── icon-48.png │ └── icon-128.png ├── scripts/ │ └── iaServiceWorker.js ├── index.html └── manifest.json

## Estrutura do Projeto

A estrutura da aplicação segue o padrão de uma extensão Chrome utilizando Manifest V3:

* \*\*manifest.json:\*\*

Arquivo de configurações que define:

* Versão da extensão e do Manifest (Manifest V3).
* Permissões necessárias: `"tabs"`, `"activeTab"`, `"sidePanel"`, `"storage"`, `"unlimitedStorage"`.
* Configuração do \*\*side panel\*\* e do \*\*popup\*\* (ambos usando `index.html` como ponto de interface).
* Detalhes dos ícones e comandos de atalho (ex.: `Ctrl+B` e a palavra-chave `assistente` para o omnibox).
* \*\*index.html:\*\*

Responsável pela interface gráfica do popup e do painel lateral, onde os usuários podem visualizar resultados, gráficos e métricas.

* \*\*scripts/iaServiceWorker.js:\*\*

Service worker que atua como backend interno da extensão, gerenciando eventos, processando requisições e, possivelmente, comunicando-se com APIs externas para enviar dados e obter métricas.

* \*\*images/:\*\*

Diretório contendo os ícones da extensão em várias resoluções para diferentes contextos (barra de ferramentas, painel, etc.).

## Instalação e Execução

### Instalação Local

1. \*\*Carregar a Extensão:\*\*

* Abra o Chrome e acesse `chrome://extensions`.
* Habilite o \*\*Modo de Desenvolvedor\*\*.
* Clique em \*\*"Carregar sem compactação"\*\* e selecione a pasta raiz do projeto (`IAssistant/`).

1. \*\*Testar a Interface:\*\*

* Clique no ícone da extensão para abrir o popup e verificar a interface.
* Utilize a funcionalidade do \*\*side panel\*\* (disponível através da configuração no manifesto).
* Utilize os atalhos (`Ctrl+B`/`Command+B`) e a palavra-chave `assistente` na Omnibox para testar os comandos.

### Executando Comandos Específicos

* \*\*Show Graphics / Show Metrics:\*\*

Estes comandos acionam funções que fazem requisições a endpoints responsáveis por retornar gráficos e métricas.

\*\*Atenção:\*\* Certifique-se que os endpoints estejam configurados para um domínio público (não "localhost") em produção.

## Configuração do Backend

A extensão se integra a um serviço backend para:

* Registrar os dados de tokenização.
* Processar as informações e exibir gráficos e métricas.
* Enviar os dados analisados para o Google BigQuery ou outros serviços em nuvem.

\*\*Recomendação:\*\*

* Se estiver utilizando `localhost` durante o desenvolvimento, altere para a URL pública do seu servidor na versão publicada para evitar erros de conexão (\_net::ERR\_CONNECTION\_REFUSED\_).

## Resolução de Problemas

* \*\*Erro de Conexão com `localhost`:\*\*

Verifique se os endpoints configurados nos comandos show graphics e show metrics apontam para um servidor público e ativo.

Caso contrário, ajustados os endpoints ou utilize variáveis de ambiente para alternar entre desenvolvimento e produção.

* \*\*Recursos Não Encontrados (ERR\_FILE\_NOT\_FOUND):\*\*

Certifique-se de que os caminhos para arquivos e recursos (como imagens, scripts e estilos) estão corretos no pacote da extensão.

* \*\*Problemas com o Service Worker:\*\*

Utilize chrome://extensions para inspecionar o service worker e verifique os logs para identificar possíveis erros.

## Contribuição

Contribuições para melhorias e novas funcionalidades são bem-vindas!

1. Fork o projeto.
2. Crie uma branch com a nova funcionalidade ou correção:

git checkout -b feature/nome-da-feature

1. Faça os commits com suas alterações.
2. Envie a branch para o repositório e abra um Pull Request.

## Licença

Este projeto está licenciado sob a [MIT License](LICENSE).

# IAssistant

O \*\*IAssistant\*\* é uma ferramenta em Python desenvolvida para analisar conteúdos textuais de documentos (PDF, HTML, TXT, CSV) e calcular métricas de similaridade entre frases extraídas dos documentos e termos de busca informados pelo usuário. Além disso, o projeto integra uma interface interativa—por meio do Dash e Plotly—para visualizar frequências, estatísticas e gráficos que evidenciam os resultados da análise.

## Funcionalidades

* \*\*Extração e Processamento de Conteúdo:\*\*
* Suporte à leitura de documentos em diversos formatos:
* PDFs (usando o PyMuPDF, via o módulo `fitz`)
* Arquivos HTML, TXT e CSV através de tratamento de conteúdo ou requisições HTTP
* Tokenização do texto utilizando NLTK, com remoção de stopwords (para os idiomas inglês e português) e limpeza textual via expressões regulares.
* \*\*Análise de Similaridade e Frequência:\*\*
* Cálculo de métricas de similaridade (por meio do \*\*cosine similarity\*\*) entre o conteúdo extraído e os termos de busca desejados.
* Divisão do documento em partes (splits por frases) para facilitar a análise por página ou por trecho.
* Geração de distribuições de frequência usando `FreqDist` do NLTK e construção de dataframes no Pandas, permitindo a visualização e o agrupamento dos dados.
* \*\*Armazenamento e Manipulação de Dados:\*\*
* Classes dedicadas à manipulação de arquivos: salvando conteúdo em formatos HTML, TXT e PDF.
* Integração com um sistema de gerenciamento de dados via `BaseManager` para compartilhamento de informações entre processos.
* \*\*Dashboards e Visualização Interativa:\*\*
* Criação de interfaces gráficas interativas utilizando o [Dash](https://dash.plotly.com/) e [Plotly](https://plotly.com/python/), com elementos como gráficos de dispersão, barras e tabelas dinâmicas.
* Componentes interativos (dropdowns, botões e popovers via dash-bootstrap-components) que permitem filtrar os dados e visualizar métricas detalhadas por página, palavra e frase.
* Funções utilitárias para definir cores (baseado na intensidade dos valores) que auxiliam na representação visual dos níveis de similaridade (ex.: vermelho para baixa similaridade, verde para alta).
* \*\*Configuração Dinâmica do Backend:\*\*
* Funções para obter configurações dinâmicas de endpoints e dados de busca (através de GET requests para URLs configuradas).

\*\*Atenção:\*\* Por padrão, esses endpoints utilizam http://localhost:8080. Em produção, é necessário atualizar esse valor para um domínio público e acessível, evitando erros de conexão.

# IAssistant — Case Completo de Engenharia de Dados + Extensão Chrome + Backend + BigQuery + Integração GPT

Projeto criado para demonstrar, de ponta a ponta, uma solução que captura dados de buscas e documentos via extensão Chrome, processa em um backend com Spark e Python, aplica anonimização, gera métricas e dashboards interativos, e integra com múltiplos destinos de armazenamento e análise: Delta Lake (Databricks), SQL Server on-premise e BigQuery. Além disso, conecta-se à API da OpenAI para análise de contexto e geração de insights.

---

## Objetivo

O IAssistant nasceu para integrar captura em tempo real de dados não estruturados (texto, PDFs, HTML) com processamento inteligente e visualização. É um ecossistema que combina coleta via browser, pipeline de processamento distribuído, armazenamento otimizado e análise assistida por IA, tudo conectado por APIs bem definidas.

---

## Arquitetura Geral

graph TD

A[Extensão Chrome] -->|POST JSON| B[/API Flask upload\_api.py]

B -->|Processa| C[Spark - Anonimização + Validação]

C -->|Delta Format| D[Delta Lake - Databricks]

C --> E[SQL Server On-prem]

C --> F[Função BigQuery - GCP]

B --> G[Dash Server Flask/Dash]

G --> H[Dashboards Interativos]

G --> I[OpenAI GPT API]

---

## Backend — Componentes

### API de Upload (`upload\_api.py`)

* Recebe JSON da extensão.
* Cria DataFrame Spark.
* Aplica máscara em campos sensíveis (`mask\_email\_udf`).
* Valida dados (ex.: contagem de linhas > 0).
* Grava em Delta Lake e SQL Server.

### Dash Server (`app.py`)

* Fornece endpoints REST para servir dados e configurações.
* Rota `/api/v1/document`: recebe documento/texto/PDF, persiste e aciona GPT.
* Serve arquivos estáticos e integra com dashboard interativo.

### Dash Process (`services/dash\_process.py`)

* Aguarda API subir (`http://localhost:8080/configurations`).
* Inicia servidor Dash na porta `8081`.

### Integração GPT (`client.py`)

* Wrapper simples para `chat.completions` da OpenAI.
* Parametriza temperatura (criatividade), número máximo de opções e tokens.

### Função BigQuery (`bigquery\_function.js`)

* Recebe requisição POST.
* Formata linha com `timestamp`, `user\_id`, `search\_query`, `best\_answer`, `percent`.
* Insere no dataset/tabela configurados.

---

## Instalação — Backend

# Clonar repositório

cd IAssistant/backend

# Criar ambiente virtual

python -m venv .venv

source .venv/bin/activate

# Instalar dependências

pip install -r requirements.txt

# Configurar variáveis no .env

# - OPENAI\_API\_KEY

# - Config BigQuery (GCP)

# - Config SQL Server (ODBC)

# Subir API principal

python app.py

# Em outro terminal, iniciar Dash Process

python services/dash\_process.py

---

## Instalação — Extensão Chrome

1. Ir em `chrome://extensions`.
2. Ativar \*\*Modo do desenvolvedor\*\*.
3. Clicar em \*\*Carregar sem compactação\*\*.
4. Selecionar pasta `IAssistant/extension`.
5. Configurar endpoint da API no `index.html`.

---

## Pipeline de Processamento

1. Extensão coleta dados (DOM, PDFs, inputs do usuário).
2. Envia via `fetch` POST para `/uploadData`.
3. Spark processa e mascara dados sensíveis.
4. Dados persistem em:

* Delta Lake (arquitetura otimizada para queries analíticas).
* SQL Server (uso interno/on-prem).
* BigQuery (análises agregadas e cruzamento com outros datasets).

1. API dispara análise GPT e retorna sugestões/insights.
2. Dashboards consomem dados processados.

---

## Similaridade e Estatística

* \*\*Tokenização e vetorização\*\*: textos quebrados em tokens, aplicando limpeza (stopwords, stemming) para comparação.
* \*\*Métrica de similaridade\*\*: cálculo de similaridade semântica entre campos (cosine similarity sobre embeddings ou TF-IDF dependendo do caso).
* \*\*Estatística descritiva\*\*: contagem, médias, percentuais, distribuição de termos.
* \*\*Validação\*\*: regras simples (mínimo de linhas, campos obrigatórios) e logs estruturados.

---

## Bibliotecas Utilizadas

* \*\*Backend\*\*: Flask, Flask-CORS, PyODBC, Pyspark, Dash, Dash Bootstrap Components.
* \*\*Integração GPT\*\*: openai.
* \*\*GCP\*\*: @google-cloud/bigquery (Node.js).
* \*\*Outros\*\*: logging, json, base64, urllib.

---

## Fluxo de Inserção no BigQuery

1. Extensão ou backend envia JSON para endpoint da função.
2. Função Node.js formata linha conforme schema.
3. Executa `table.insert()` no BigQuery.
4. Registro disponível quase em tempo real para consulta via SQL.

---

## Resultado

Com essa arquitetura, conseguimos ingerir, processar, anonimizar e disponibilizar dados não estruturados de forma escalável, com integração entre ambiente on-prem e cloud, além de análise contextual via GPT e visualização imediata em dashboards.

Case de Engenharia de Dados + IAssistant (Extensão + Backend + BigQuery)

Eu montei este case unificando a extensão IAssistant com um backend analítico e um projeto BigQuery governado via authorized views. Ideia simples: capturo consulta + conteúdo (PDF/HTML/TXT/CSV), tokenizo, calculo TF-IDF + cosseno por frase/página, mostro gráficos no Dash e persiste no BigQuery (com máscaras e auditoria).

Projeto GCP: case-de-engenharia-de-dados

Região base: southamerica-east1

Dataset Marketplace github\_activity: sempre em US → consultas com --location=US.

Canvas 1 — Arquitetura (fim-a-fim)

flowchart LR

subgraph Browser[Chrome]

EXT[Extensão IAssistant\npopup/side-panel/omnibox]

end

EXT -- POST /api/v1/document --> API8080[API Core (Flask)\n8080 (dash\_server.py)]

EXT -- POST /uploadData --> API5000[Upload API (Flask+Spark)\n5000 (api\_uploader.py)]

API8080 --> DASH[Dash 8081\n(dash\_process.py + IAUtils.start\_server())]

API8080 -- opcional --> GPT[OpenAI client.py]

API5000 -- insert --> BQ[(BigQuery\niaassistant.logs/pages/terms)]

API5000 -- append --> DELTA[(Delta Lake/DBFS)]

API5000 -- mirror --> SQL[(SQL Server on-prem)]

subgraph GCP[Projeto: case-de-engenharia-de-dados]

subgraph BQSEC[BigQuery Governança]

direction TB

BASE[(Dataset base\n845415315315)]

VIEWS[(Dataset views\nsec\_v)]

BASE -->|Authorized View| V1[sec\_v.user\_iassistant\_v]

BASE -->|Authorized View| V2[sec\_v.phrase\_metrics\_v]

end

LOGS[Cloud Logging\n(Data Access Logs)]

end

CSV[NDJSON/CSV usuarios] -->|load job| BASE

LOGS -.audita leitura base.-> BASE

LOGS -.audita leitura view.-> V1

Padrão de dados que usei

Kappa-like por padrão: um único caminho de eventos (extensão → APIs) atende realtime + replay.

Lambda on-demand: quando preciso de lotes massivos, rodo batch e converjo no BigQuery. Operação simples.

Canvas 2 — Pipeline analítico (tokenização/TF-IDF/cosseno)

flowchart LR

Q[Consulta] --> N[Normalização\nlower/regex]

D[Documento (PDF/HTML/TXT/CSV)] --> N

N --> T[Tokenização + stopwords (NLTK)]

T --> V[TF-IDF]

Q --> V

V --> S[Cosine por frase/trecho]

S --> A[Agregação por página (PDF)]

S --> F[FreqDist tokens]

A --> G1[Gráfico by\_page]

F --> G2[Gráfico top\_words]

S --> G3[Tabela frases (score)]

Como calculei

Normalização: minúsculas, remoção de pontuação/ruído com regex.

Split: PDF por página (PyMuPDF) + frase (NLTK). HTML/TXT idem (frase/bloco).

TF-IDF:

t

f

i

d

f

(

𝑡

,

𝑑

)

=

t

f

(

𝑡

,

𝑑

)

⋅

log

⁡

𝑁

𝑑

𝑓

(

𝑡

)

tfidf(t,d)=tf(t,d)⋅log

df(t)

N

​

Cosseno:

cos

⁡

(

𝜃

)

=

𝑞

⃗

⋅

𝑑

⃗

∥

𝑞

⃗

∥

⋅

∥

𝑑

⃗

∥

cos(θ)=

∥

q

​

∥⋅∥

d

∥

q

​

⋅

d

​

page\_metric\_by\_page: média/percentil dos scores das frases daquela página.

tokens\_top: FreqDist (termo, frequência, páginas).

Thresholds (ex.: 0.15) para realce visual.

Canvas 3 — Modelo (ER) unificado

erDiagram

LOGS {

TIMESTAMP ts

STRING page\_url

STRING query

STRING lang

STRING source\_type

STRING source\_url

STRING client\_version

STRING user\_id

}

PAGES {

TIMESTAMP ts

STRING doc\_id

INT64 page\_number

FLOAT64 page\_metric\_by\_page

}

TERMS {

TIMESTAMP ts

STRING doc\_id

STRING token

INT64 frequency

ARRAY<INT64> pages

}

USER\_IASSISTANT {

INT64 USER\_ID PK

STRING USER\_NAME

STRING USER\_EMAIL

STRING REGION "lat,lon"

INT64 DT\_CARGA

}

PHRASE\_METRICS {

STRING user\_id FK

FLOAT64 PHRASE\_METRICS

STRING PHRASE\_TEXT

INT64 PAGE\_METRICS

INT64 PAGE\_NUMBER

STRING SEARCH\_ITEMS

STRING SEARCH\_RESULT

INT64 DT\_CARGA

}

USER\_IASSISTANT ||--o{ PHRASE\_METRICS : "user\_id = CAST(USER\_ID AS STRING)"

Instalação rápida

Backend (Python 3.9+)

cd IAssistant/backend

python -m venv .venv

# Windows: .venv\Scripts\activate

source .venv/bin/activate

pip install -r requirements.txt

python -c "import nltk; nltk.download('punkt'); nltk.download('stopwords')"

Crie .env (a partir do .env.example) e ajuste:

API\_CORE\_HOST=0.0.0.0

API\_CORE\_PORT=8080

DASH\_HOST=0.0.0.0

DASH\_PORT=8081

CORS\_ALLOWED\_ORIGINS=\*

BQ\_ENABLED=true

BQ\_PROJECT\_ID=case-de-engenharia-de-dados

BQ\_DATASET=iaassistant

BQ\_TABLE\_LOGS=logs

BQ\_TABLE\_PAGES=pages

BQ\_TABLE\_TERMS=terms

GOOGLE\_APPLICATION\_CREDENTIALS=/abs/path/service-account.json

DELTA\_ENABLED=false

DELTA\_OUTPUT\_PATH=dbfs:/mnt/extension\_data/processed/

SQL\_ENABLED=false

SQL\_SERVER=YOUR\_SQL\_SERVER\_ADDRESS

SQL\_DATABASE=YOUR\_DATABASE\_NAME

SQL\_USERNAME=YOUR\_USERNAME

SQL\_PASSWORD=YOUR\_PASSWORD

SQL\_TABLE=ExtensionData

OPENAI\_API\_KEY=sk-...

OPENAI\_MODEL=gpt-4o-mini

OPENAI\_AGENT\_ID=

OPENAI\_INSTRUCTIONS="Responda com até {0} opções claras..."

Suba os serviços (2 terminais):

python dash\_server.py # porta 8080

python services/dash\_process.py # porta 8081

Extensão (Chrome)

chrome://extensions → Modo dev → Carregar sem compactação → IAssistant/extension.

Em index.html, ajuste:

<script>

const defaultConfig = {

API\_METRICS\_URL: "http://localhost:8080/api/metrics",

API\_GRAPHICS\_URL: "http://localhost:8080/graphics",

API\_UPLOAD\_URL: "http://localhost:5000/uploadData",

API\_GPT\_URL: "http://localhost:8080/gpt/complete"

};

</script>

BigQuery (dataset analítico IAssistant)

CREATE SCHEMA IF NOT EXISTS case-de-engenharia-de-dados.iaassistant;

CREATE TABLE IF NOT EXISTS case-de-engenharia-de-dados.iaassistant.logs (

ts TIMESTAMP, page\_url STRING, query STRING, lang STRING,

source\_type STRING, source\_url STRING, client\_version STRING, user\_id STRING

);

CREATE TABLE IF NOT EXISTS case-de-engenharia-de-dados.iaassistant.pages (

ts TIMESTAMP, doc\_id STRING, page\_number INT64, page\_metric\_by\_page FLOAT64

);

CREATE TABLE IF NOT EXISTS case-de-engenharia-de-dados.iaassistant.terms (

ts TIMESTAMP, doc\_id STRING, token STRING, frequency INT64, pages ARRAY<INT64>

);

BigQuery (case governança: base + views)

BASE="case-de-engenharia-de-dados:845415315315"

LOC=$(bq show --format=prettyjson "$BASE" | jq -r .location)

bq rm -f -d case-de-engenharia-de-dados:sec\_v 2>/dev/null || true

bq mk --dataset --location="$LOC" case-de-engenharia-de-dados:sec\_v

Views com máscara:

CREATE OR REPLACE VIEW case-de-engenharia-de-dados.sec\_v.user\_iassistant\_v AS

SELECT

CAST(USER\_ID AS INT64) AS user\_id,

CONCAT(SUBSTR(LOWER(TRIM(USER\_EMAIL)),1,1),'\*\*\*',

REGEXP\_EXTRACT(LOWER(TRIM(USER\_EMAIL)), r'(@.\*)$')) AS email\_mascarado,

ROUND(

IF(ARRAY\_LENGTH(REGEXP\_EXTRACT\_ALL(REGION, r'(-?[0-9]+(?:\.[0-9]+)?)'))>0,

CAST(REGEXP\_EXTRACT\_ALL(REGION, r'(-?[0-9]+(?:\.[0-9]+)?)')[OFFSET(0)] AS FLOAT64), NULL), 2

) AS lat\_mascarada,

ROUND(

IF(ARRAY\_LENGTH(REGEXP\_EXTRACT\_ALL(REGION, r'(-?[0-9]+(?:\.[0-9]+)?)'))>1,

CAST(REGEXP\_EXTRACT\_ALL(REGION, r'(-?[0-9]+(?:\.[0-9]+)?)')[OFFSET(1)] AS FLOAT64), NULL), 2

) AS lon\_mascarada

FROM case-de-engenharia-de-dados.845415315315.user\_iassistant;

CREATE OR REPLACE VIEW case-de-engenharia-de-dados.sec\_v.phrase\_metrics\_v AS

SELECT

CAST(user\_id AS STRING) AS user\_id,

CAST(PHRASE\_METRICS AS FLOAT64) AS phrase\_metrics,

IFNULL(SUBSTR(PHRASE\_TEXT, 1, 20) || '...', NULL) AS phrase\_text\_masked,

CAST(PAGE\_METRICS AS INT64) AS page\_metrics,

CAST(PAGE\_NUMBER AS INT64) AS page\_number,

SEARCH\_ITEMS AS search\_items,

IFNULL(SUBSTR(SEARCH\_RESULT, 1, 60) || '...', NULL) AS search\_result\_masked,

CAST(DT\_CARGA AS INT64) AS dt\_carga

FROM case-de-engenharia-de-dados.845415315315.phrase\_metrics;

Autorizar as views no dataset base:

bq show --format=prettyjson "$BASE" > base.json

jq '.access += [

{"view":{"projectId":"case-de-engenharia-de-dados","datasetId":"sec\_v","tableId":"user\_iassistant\_v"}},

{"view":{"projectId":"case-de-engenharia-de-dados","datasetId":"sec\_v","tableId":"phrase\_metrics\_v"}}

] | .access |= (map(tojson)|unique|map(fromjson))' base.json > base\_patched.json

bq update --source=base\_patched.json "$BASE"

Canvas 4 — Scripts (copy-paste)

Cada bloco abaixo é um canvas de script com o arquivo inteiro. Copie o que precisar.

-> backend/dash\_server.py

from IAUtils import IAssistantRequests,IAssistantRuntimeData

from flask import Flask

from flask import redirect, request

from urllib import parse

from flask\_cors import CORS

from client import sendQuestionGenIA

import init\_dash\_servers

import os, base64, urllib, time

from flask import send\_file

diretorio\_atual = os.path.dirname(os.path.abspath(\_\_file\_\_))

pasta\_static = os.path.join(diretorio\_atual, 'static')

os.makedirs(pasta\_static, exist\_ok=True)

runtime\_data = IAssistantRuntimeData()

ia\_download\_manager = IAssistantRequests()

def start\_api\_server():

app\_core = Flask(\_\_name\_\_)

CORS(app\_core, origins='\*', allow\_headers='\*')

@app\_core.route('/')

def index():

caminho\_index = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..', 'index.html'))

return send\_file(caminho\_index)

@app\_core.route('/graphics')

def send\_file\_graphics():

return redirect("http://localhost:8081/index.html")

@app\_core.route('/configurations')

def get\_configurations():

import json

url\_tmp = urllib.parse.quote(runtime\_data.current\_url)

return json.dumps({'url': url\_tmp}), 200

@app\_core.route('/search/options')

def get\_input\_data():

import json

return json.dumps(runtime\_data.current\_inputs), 200

@app\_core.route('/api/v1/document', methods=['POST'])

def create\_document():

data = request.json

inputSearch = parse.unquote(base64.b64decode(data["inputSearchTarget"]))

listOptionsGenIA = data['listOptionsGenIA']

typ = data["type"]

runtime\_data.current\_site\_url = data['url']

if typ=='text':

contentDocument = parse.unquote(base64.b64decode(data["contentText"]).decode('utf-8'))

binary\_content = parse.unquote(base64.b64decode(data["BinaryContent"]).decode('utf-8'))

runtime\_data.current\_url = ia\_download\_manager.save\_text\_file\_binary\_content(

content=binary\_content, filename='ia\_html\_binary\_file\_output', code\_content\_to\_filename=0)

elif typ=='pdf':

binary\_content = base64.b64decode(data['BinaryContent'])

contentDocument = parse.unquote(base64.b64decode(data["pdf"]).decode('utf-8'))

runtime\_data.current\_url = ia\_download\_manager.save\_pdf\_file\_binary\_content(content=binary\_content)

output = sendQuestionGenIA(inputSearch, contentDocument,

number\_response\_options=listOptionsGenIA, creativity\_degree=1.5)

runtime\_data.current\_inputs = {

'inputSearchTarget': inputSearch,

'search\_options': output,

'tabId': data['tabId'],

'listOptionsGenIA': listOptionsGenIA,

'doc': contentDocument,

'url': runtime\_data.current\_site\_url

}

init\_dash\_servers.init\_dashboard\_services()

time.sleep(15)

return "output.html", 201

return app\_core

if \_\_name\_\_ == '\_\_main\_\_':

app = start\_api\_server()

app.run(port=8080,host='0.0.0.0', debug=True)

-> backend/services/dash\_process.py

import IAUtils, time, requests

from IAUtils import start\_server

for i in range(10):

try:

requests.get("http://localhost:8080/configurations", timeout=1)

break

except requests.exceptions.RequestException:

print(f"Aguardando API (tentativa {i+1}/10)...")

time.sleep(1)

else:

raise RuntimeError("API nunca ficou disponível. Abortando dashboard.")

app\_dash = start\_server()

app\_dash.run(port=8081, host="0.0.0.0")

-> backend/client.py (OpenAI opcional)

from openai import OpenAI

from dotenv import load\_dotenv

import os

load\_dotenv()

def sendQuestionGenIA(message,inputDocument,number\_response\_options,creativity\_degree):

client = OpenAI(api\_key=os.getenv("OPENAI\_API\_KEY"))

completion = client.chat.completions.create(

max\_tokens=1200,

model=os.getenv("OPENAI\_MODEL"),

temperature=creativity\_degree,

messages=[

{"role": "system", "content": os.getenv("OPENAI\_INSTRUCTIONS").format(number\_response\_options)},

{"role": "user", "content": message}

]

)

return completion.choices[0].message.content

if \_\_name\_\_=="\_\_main\_\_":

print("="\*20,"\nLoaded IAssistant Module!!!\n","="\*20)

-> backend/api\_uploader.py (Spark/Delta + SQL)

from flask import Flask, request, jsonify

import logging, json, pyodbc

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, udf

from pyspark.sql.types import StringType

logging.basicConfig(level=logging.INFO, format='%(asctime)s [%(levelname)s] %(message)s')

logger = logging.getLogger(\_\_name\_\_)

app = Flask(\_\_name\_\_)

spark = SparkSession.builder.appName("ChromeExtensionUploader").getOrCreate()

def mask\_email(email):

if email and "@" in email:

parts = email.split("@")

return parts[0][:2] + "\*" \* max(len(parts[0])-2,0) + "@" + parts[1]

return email

mask\_email\_udf = udf(mask\_email, StringType())

def update\_on\_premise\_sql(data):

try:

connection\_string = (

"Driver={ODBC Driver 17 for SQL Server};"

"Server=YOUR\_SQL\_SERVER\_ADDRESS;"

"Database=YOUR\_DATABASE\_NAME;"

"UID=YOUR\_USERNAME;"

"PWD=YOUR\_PASSWORD;"

)

cnxn = pyodbc.connect(connection\_string)

cursor = cnxn.cursor()

cursor.execute("INSERT INTO ExtensionData (data\_json) VALUES (?)", json.dumps(data))

cnxn.commit(); cursor.close(); cnxn.close()

logger.info("Dados inseridos no SQL on premise.")

except Exception as e:

logger.error("Erro ao atualizar SQL on premise: %s", str(e))

raise e

@app.route("/uploadData", methods=["POST"])

def upload\_data():

try:

content = request.get\_json()

logger.info("Dados recebidos: %s", content)

df = spark.createDataFrame([content])

if "email" in df.columns:

df = df.withColumn("email\_masked", mask\_email\_udf(col("email"))).drop("email")

if df.count() < 1:

return jsonify({"status":"error","message":"Nenhum dado recebido!"}), 400

df.write.format("delta").mode("append").save("dbfs:/mnt/extension\_data/processed/")

update\_on\_premise\_sql(content)

return jsonify({"status":"success","message":"Dados recebidos e processados."}), 200

except Exception as e:

logger.error("Erro ao processar: %s", str(e))

return jsonify({"status":"error","message":str(e)}), 500

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host="0.0.0.0", port=5000, debug=True)

* > backend/bigquery\_function.js (HTTP → BigQuery)

const { BigQuery } = require('@google-cloud/bigquery');

const bigquery = new BigQuery();

const datasetId = 'case-de-engenharia-de-dados';

const tableId = 'search\_iassistent';

exports.insertSearchData = async (req, res) => {

if (req.method !== 'POST') { res.status(405).send('Method Not Allowed'); return; }

try {

const body = req.body;

if (!body.timestamp) body.timestamp = new Date().toISOString();

const row = {

timestamp: body.timestamp,

user\_id: body.user\_id,

search\_query: body.search\_query,

best\_answer: body.best\_answer,

percent: body.percent || null

};

await bigquery.dataset(datasetId).table(tableId).insert([row]);

res.status(200).send('Data inserted successfully!');

} catch (err) {

console.error('Error inserting data: ', err);

res.status(500).send('Error inserting data.');

}

};

Canvas 5 — Auditoria (Data Access Logs) e IAM

Habilite Data Access Logs (Console > IAM e Admin > Audit Logs > BigQuery > Data Read/Write).

Checagens rápidas:

# leitura direta na base

gcloud logging read \

'logName="projects/case-de-engenharia-de-dados/logs/cloudaudit.googleapis.com%2Fdata\_access" \

AND resource.type="bigquery\_dataset" \

AND protoPayload.metadata."@type"="type.googleapis.com/google.cloud.audit.BigQueryAuditMetadata" \

AND protoPayload.metadata.tableDataRead.reason:\* \

AND protoPayload.resourceName:"projects/case-de-engenharia-de-dados/datasets/845415315315/tables/user\_iassistant"' \

--limit=5 \

--format='value(timestamp, protoPayload.metadata.tableDataRead.reason, protoPayload.resourceName)'

Métricas que criei:

# leituras na base

gcloud logging metrics create read\_base\_dataset\_metric \

--description="Leituras na tabela base (bypass da view)" \

--log-filter='logName="projects/case-de-engenharia-de-dados/logs/cloudaudit.googleapis.com%2Fdata\_access" AND resource.type="bigquery\_dataset" AND protoPayload.metadata."@type"="type.googleapis.com/google.cloud.audit.BigQueryAuditMetadata" AND protoPayload.metadata.tableDataRead.reason:\* AND protoPayload.resourceName:"projects/case-de-engenharia-de-dados/datasets/845415315315/tables/user\_iassistant"'

# queries referenciando a base

gcloud logging metrics create query\_ref\_base\_table\_metric \

--description="Queries que referenciam a tabela base" \

--log-filter='logName="projects/case-de-engenharia-de-dados/logs/cloudaudit.googleapis.com%2Fdata\_access" AND resource.type="bigquery\_project" AND protoPayload.metadata."@type"="type.googleapis.com/google.cloud.audit.BigQueryAuditMetadata" AND protoPayload.metadata.jobCompletedEvent.job.jobStatistics.referencedTables:"projects/case-de-engenharia-de-dados/datasets/845415315315/tables/user\_iassistant"'

# leituras via view

gcloud logging metrics create read\_via\_view\_metric \

--description="Leituras na authorized view" \

--log-filter='logName="projects/case-de-engenharia-de-dados/logs/cloudaudit.googleapis.com%2Fdata\_access" AND resource.type="bigquery\_dataset" AND protoPayload.metadata."@type"="type.googleapis.com/google.cloud.audit.BigQueryAuditMetadata" AND protoPayload.metadata.tableDataRead.reason:\* AND protoPayload.resourceName:"projects/case-de-engenharia-de-dados/datasets/sec\_v/tables/user\_iassistant\_v"'

IAM dinâmico por dataset (exemplo):

PROJECT\_ID="case-de-engenharia-de-dados"

DATASET\_ID="845415315315"

DATASET\_REF="$PROJECT\_ID:$DATASET\_ID"

SRC\_TABLE="$PROJECT\_ID.$DATASET\_ID.user\_iassistant"

bq query --nouse\_legacy\_sql --format=csv "

WITH base AS (

SELECT DISTINCT CAST(USER\_ID AS INT64) AS user\_id,

LOWER(TRIM(USER\_EMAIL)) AS email

FROM \$SRC\_TABLE\

WHERE USER\_EMAIL IS NOT NULL

)

SELECT email,

CASE WHEN user\_id BETWEEN 1 AND 5 THEN 'READER' ELSE 'WRITER' END AS role

FROM base

" | tail -n +2 | while IFS=, read -r email role; do

[[ -z "$email" || -z "$role" ]] && continue

bq show --format=prettyjson "$DATASET\_REF" > ds.tmp.json

jq --arg em "$email" --arg r "$role" \

'.access += [{"role":$r,"userByEmail":$em}] | .access |= (map(tojson)|unique|map(fromjson))' \

ds.tmp.json > ds.apply.json

bq update --source=ds.apply.json "$DATASET\_REF"

rm -f ds.tmp.json ds.apply.json

done

Consultas úteis (BigQuery)

-- páginas mais “quentes” por relevância (IAssistant)

SELECT \*

FROM case-de-engenharia-de-dados.iaassistant.pages

WHERE page\_number > 2 AND page\_metric\_by\_page < 0.25

ORDER BY ts DESC;

-- top tokens por documento (IAssistant)

SELECT doc\_id, token, SUM(frequency) AS freq

FROM case-de-engenharia-de-dados.iaassistant.terms

GROUP BY doc\_id, token

ORDER BY freq DESC

LIMIT 50;

-- auditoria de origens (IAssistant)

SELECT page\_url, COUNT(\*) AS hits

FROM case-de-engenharia-de-dados.iaassistant.logs

GROUP BY page\_url

ORDER BY hits DESC

LIMIT 100;

Notas finais

Extensão manda contexto; APIs processam e persistem; Dash explica; BigQuery governa.

Máscaras nas views, Data Access Logs para auditoria, e métricas para sinalizar bypass.

OpenAI aqui é para resumo/rotulagem, não decide ranking (o ranking é matemático e auditável).