#ASLI ENGINEERING

Email Classification at Slack

SWIPE

BY

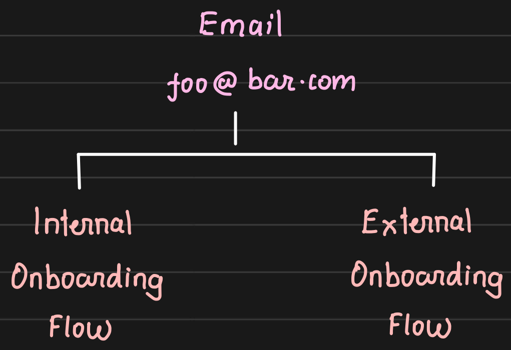
ARPIT BHAYANI

Email Classification @ Slack

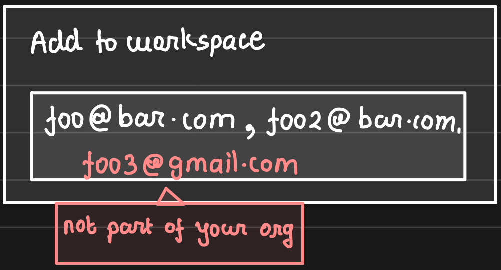
On slack, we can invite people by email

Two kinds of people we can invite

1. Internal → part of the same org
2. External → part of different org



To give a smoother invitation experience Slack classifies the email and than gives preference to that option



Can they just not compare email domains?

No! because email domains can be diff

eg: there are org that assigns emails per region

eg: foo1@bar.in
foo2@bar.us
} foo1 and foo2 are employees

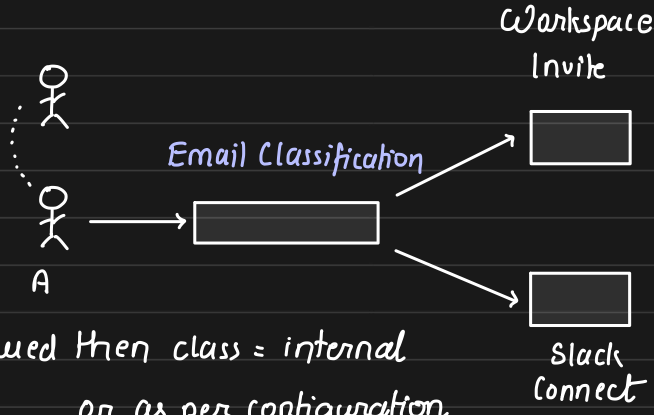
eg: some org provide diff email to contractual employees, vendors and interns

eg: foo3@bar.external
foo4@bar.temp

ARPIT BHAYANI

Email Classification Service

Email classificiation is first attempted to be done through heuristics, example



1. Settings Context

   if only certain domains allowed then class = internal or as per configuration

2. Inviter context

   if inviter's domain = invitee's domain then inviter's class applied

3. Team Context

   Above two are simple settings driven, but Team's Context requires database query and some logic to determine the class.

   This is a challenge because Slack workspace can have million members

Team Domain Context

Idea: Keep track of all domains part of a workspace and use that aggregated count to classify.

ARPIT BHAYANI

Table 'domains'

team_id	domain	count	date_update	role
A	bar com	2	———	admin
A	bar.com	68	———	member
A	bar.in	30	———	member
A	gmail.com	3	———	member

Total count of users grouped by role matching the same domain

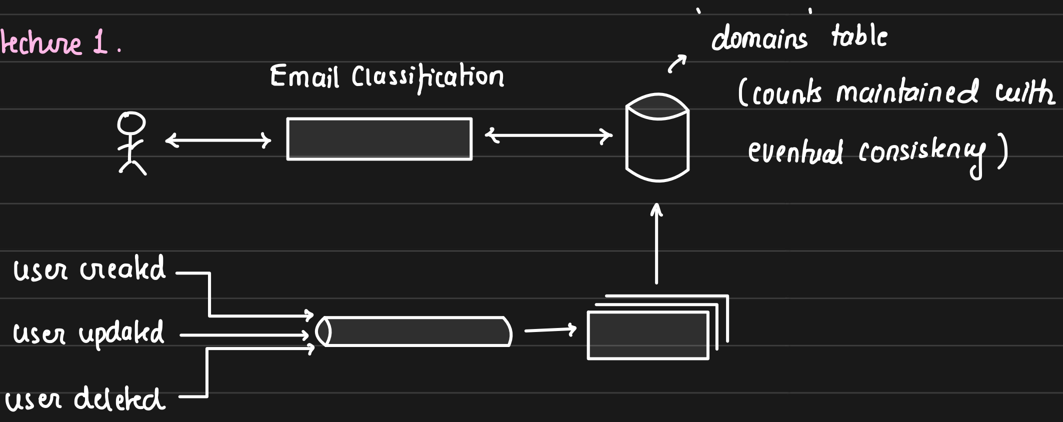
* admin having '@bar.com' is a bigger indication that it is an internal domain!!

Threshold : 10% Domain to be considered as internal if there are at least 10% or more employees in organisation with given domain

Hence, foo1 @ bar.com → internal   foo2 @ bar.in → internal

foo3 @ gmail.com → external

Architecture 1.



ARPIT BHAYANI

Implementation Details

We call UPSERT instead of INSERT to do relative add^n / subtract^n

eg: user creation: UPSERT count = count + 1 where
team_id = T and domain = bar.com and
role = member;

Why upsert: row-level lock, relative operations

eg: user updation: UPSERT count = count − 1 where
team_id = T and domain = bar.com and
role = member ;

UPSERT count = count + 1 where
team_id = T and domain = bar.com and
role = admin ;

* No matter how many queries are fired, because upserts take row level lock we can be sure that the system will remain eventually consistent

Challenge: Message can be processed twice
↳ numbers can drift and hence we need a 'healer'

The system will auto-heal whenever it sees a drift

1. when an email address is added for the 1st time
2. when user upgrades their plan
3. periodically

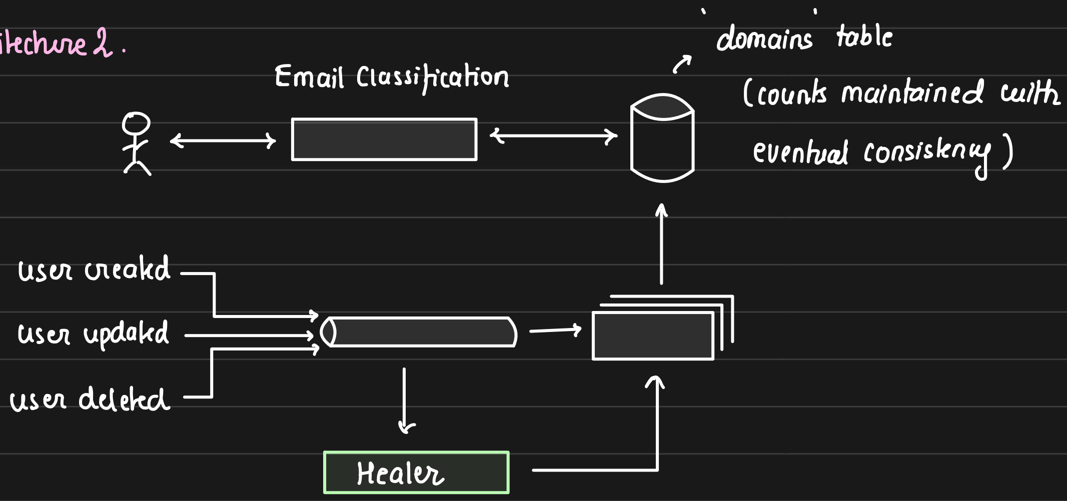
Naive Healing : recompute the count and update the table

↳ what about the events | updates that happened while healer ran?

Better approach: mark the datetime when healer starts, note the existing count, compute the actual count, trigger upert to correct drift (until that datetime)

eg: + N | - N (ensures no mutations are lost)

Architecture 2.



ARPIT BHAYANI