# Data Pipeline Maintenance

## Day 2 Lecture

*Maintain Data Pipelines like Airbnb and Netflix*

**Transcript:**

for it okay there's essentially four things here
1:07:28
that I think are going to be your big signals at the highest level of we are
1:07:34
we have Tech de or too much Tech that in data engineering we have uh pipelines breaking we have uh Cloud bills then we
1:07:41
have large Cloud bills that are maybe out of control Cloud bills we also have multiple sources of truth it's another
1:07:48
interesting Tech de problem and then you have unclear data sets that are aren't documented so each one of these has
1:07:55
their own kind of like remedies to kind of fix we're going to go over kind of
1:08:01
some interesting remedies that we can go over for each one of these buckets so that we can know how to maybe fix some
1:08:08
of the stuff okay say you have a painful pipeline you have a pipeline that's
1:08:14
breaking and say it's break every on call so you have an on call and then it's like every Saturday and Sunday it
1:08:20
breaks and it ruins the on calls weekend and then it's like then you just you just d being on call because you're like
1:08:26
I have to give up my weekend and I don't want to do that cuz like a lot of times like if you're on call and it's the weekend and then you have to work on the
1:08:32
weekend and then you get no break and then you have to go immediately into work after working on the weekend you're
1:08:37
like data engineering is freaking me out right now man I don't want to do it and so uh that's where uh troubleshooting
1:08:44
these problematic children and these problem pipelines can be very important so the only thing you know I like this

1:08:51
the only thing better than optimized is deprecated I love that saying I uh
1:08:56
because it's like that's how you get the most efficiency right is like if you just delete the pipeline you save the
1:09:01
most in the cloud right if you just the pipeline doesn't exist um obviously that's not always the case of what you
1:09:07
want but um so sometimes optim deprecated is the right play uh I want
1:09:13
to talk real quick about an example that I had at Netflix that was interesting so
1:09:19
when I was working at Netflix I worked in detection right and so detection they had this like Legacy system for detection that was based on it was a
1:09:26
batch processing Legacy system it was like an hourly batch processing Legacy system and uh it ran and I mean there
1:09:33
was like it was like a suite of like 50 pipelines that were all running to like detect anomalies in the cloud environment and uh the security team was
1:09:40
like this isn't good because it detects anomalies an hour late and uh we were like yeah that kind of makes sense that
1:09:46
makes sense why that's not good and then like we were like okay we're going to we're going to move some of them to streaming we're going to move uh
1:09:51
whatever we can to streaming and uh but then we real that like even before we
1:09:57
moved everything over to streaming we recognized that they weren't even using the data anyways so we were like why are
1:10:03
we paying this maintenance cost every single day on these like 50 plus
1:10:08
pipelines for data that's not even being used data that's not providing any value like why are we even doing that so
1:10:15
um uh like so what we did was we got
1:10:21
everyone into into a room and we were like can we get rid of this do we even need this like this is
1:10:28
causing us a lot of headache do we even need to have these pipelines around anymore and uh there was a lot of push

1:10:34
back because there was essentially the like what if like what if we need to run
1:10:39
a query and then uh we we we we we talked with them and we're like you haven't ran a query in the last 60 days
1:10:46
so that what if is like maybe that's not like actually gonna happen but obviously
1:10:52
security is an interesting space right because it's like you you want that you you only need the data when like
1:10:57
something bad happens right so it's like a lot of the times like maybe you don't need the data but um it's kind of the
1:11:03
idea here uh so uh that was something that we ended
1:11:08
up doing there we actually were able to convince them uh it took like like a six-month process of convincing them to
1:11:14
delete these pipelines but we were able to do it and that made our lives a lot better because then our uh the
1:11:19
maintenance burden for the on call dropped like it went from like on call being terrible to on call like almost
1:11:26
being nothing and so a lot of times that can be the case and there was like because the thing was those pipelines were already as optimized as they could
1:11:32
be there was nothing else we could do to make those pipelines better it was just that there was a lot of them and that like they were doing a lot of work and
1:11:39
that like once like you kind of have the uh um tragedy Not Tragedy of commments
1:11:44
you have the um Murphy's Law or something like that like where once you have enough of something you're
1:11:49
guaranteed to have one bad bad of that right it's like once you have a room of enough people right you're guaranteed to
1:11:56
have at least one like just guaranteed right because like that's just how people work or there's a
1:12:01
certain percentage even if it's like one% or half a percent like once you have enough people you're guaranteed that there's going to be at least one
1:12:07

in that room and so um same perspective here right where it's like okay well well like maybe we shouldn't
1:12:14
try to spend all of our time and effort optimizing these pipelines but we should actually uh just delete them
1:12:21
and in that case it was actually the right call and so uh like that's one of
1:12:27
the things that you got to remember about especially when you inherit pipelines and they weren't necessarily
1:12:32
pipelines that you built yourself but you inherit a suite of pipelines from another team or another engineer like
1:12:38
question the value and actually ask yourself like is this worth being on the on call rotation is this something that
1:12:45
should actually be troubleshooted over the weekend because maybe it's not and maybe it shouldn't even
1:12:51
exist and if that's the case then delete eting it is great because then your on
1:12:56
call becomes way easier and like I I'm sure some of y'all like are like Zack that sounds like kind of uh like too
1:13:04
good to be true or like because I I think all of us have had nightmares as data Engineers where we were like you
1:13:10
know our life would be a lot easier if we just deleted all the pipelines and then there would be no data and then and then our lives would be solved and then
1:13:16
we wouldn't have to do anything and then all the all the buzzing in our head and all the anxiety in our hearts would go away and uh but like um in in some cases
1:13:24
I've actually had a couple couple cases in my career where that that was the that was the call so um is there
1:13:29
anything that can be done uh with this right besides you know deleting it uh you can migrate to new technologies it's
1:13:37
great uh you can uh maybe Implement better data modeling so you have fewer
1:13:42
pipelines that are running because you have more robust data modeling where like you don't have to have like duplicative models and you kind of have

1:13:49
like one model or one pipeline that produces a more robust model cuz then it's like if you can take say you have a
1:13:55
suite of 100 pipelines and there's a 1% chance that things break then that means that something's going to break every
1:14:01
day but if you have 10 pipelines and there's a 1% chance that things are going to break that means that things
1:14:06
are going to break like a couple times a month and big difference very big
1:14:11
difference like especially if you can keep the failure rate the same and you can make the modeling better then like
1:14:17
uh uh the con the consistency at which your things actually fail because things are going to fail pipelines fail like and like nothing's perfect and don't
1:14:24
ever expect to write a pipeline it's never going to fail like it's going to fail and and and it's just all about like the frequency and the rate and all
1:14:30
that stuff that you want to think about right other things that you might want to consider uh when you have a painful
1:14:35
pipeline is can you bucket bucket the table that can sometimes help quite a bit I I'll talk more about some examples
1:14:42
that I've had in my career where bucketing was like a silver bullet and the other one is sampling sampling is
1:14:47
great um uh a lot of times because a lot of times you just need directionality you don't need all every single record
1:14:54
and if you can just like like get a 1% sample then you just did a 100x reduction in the complexity of that
1:15:00
Pipeline with like one line of code and sometimes that's the solution just
1:15:05
don't process all the data like I know these Solutions seem like in some ways they're not like tackling the scale they
1:15:12
don't sound sophisticated but they're smart that's the part of it that's great is that like if because if you have a
1:15:19

like a bunch of pipelines that are processing a lot of data and they're not providing that much value and they're causing you a lot of headache like
1:15:26
this is a beautiful path forward that can actually work and especially in terms of Maintenance and I've seen this
1:15:32
work many times and uh definitely remember that deleting the pipeline
1:15:37
could potentially be a tool in your tool belt you don't have to just optimize it
1:15:42
so that's a beautiful path forward I love it so uh I'm sure some of y'all have
1:15:50
upgraded from hive to spark that's probably a common thing that y'all have done
1:15:55
uh I've seen massive gains here but there's also like it's interesting
1:16:00
because this promise is not uh linear or not uniform right because a lot of times
1:16:07
like some people are like oh if you migrate from hive to spark and and you uh you'll see a massive performance gain
1:16:14
and that's only in the case of very specific uh types of pipelines and they
1:16:20
have to like essentially use like a very high cardinality group buy or High
1:16:25
cardinality join they have one of those two keywords in your sequel then
1:16:30
migrating from hive to spark you're going to see a massive massive gain like probably between 50 to 90% um more
1:16:38
efficient and uh like that's that's awesome I think one of the things here
1:16:44
that some data engineers make a mistake about is they promise more because they
1:16:49
think that that's going to be the case for every pipeline but some pipelines you're going to see flat
1:16:55
pipelines are going to be uh only 5 or 10% more efficient uh and then but there are some there are some edges where like
1:17:01
it's really really a lot more efficient where the being it because Spark versus Hive are very interesting because Hive
1:17:08
does everything like on dis it doesn't have any it doesn't use Ram at all like
1:17:13

blows my mind that that's what we did for so many years like we're just going to write all our data to dis forever and
1:17:19
but in the same vein though because Hive does everything on disc it's very resilient and like um a lot like might
1:17:25
be really slow because it does everything on dis but it's definitely going to um finish whereas spark might
1:17:31
actually fail like that's one of the other things that I noticed when I did some Hive to spark migrations is it was
1:17:37
spark was more efficient but less reliable we like uh spark is actually
1:17:42
more prone to out of memory exceptions and more prone to skew than Hive is um
1:17:50
so when you do these migrations sometimes like it's a trade-off it's not just like you're you're getting it's not
1:17:56
an it's not like an upgrade it's uh like you're getting a certain set of traits but you're losing other sets of other
1:18:03
set of traits and for the most part the trade-off is worth it and that's why people do the migration but that's what
1:18:10
I'm saying is that like Hive pipelines are technically more reliable they're more reliable but less efficient so
1:18:17
that's a thing to remember and it's and the main reason for that is because they do all the calculations on disk whereas spark uses Ram and uh that can be one
1:18:25
great way to improve the uh kind of efficiency of your pipeline and uh and
1:18:31
the landing time and all sorts of other things like that but not necessarily the reliability right because the reliability is the one side that like
1:18:37
Hive and sparker um like Hive is actually has a slight Edge so that's one thing you can do
1:18:44
upgrade the technology another thing another option here right is maybe uh you want to move to like streaming you
1:18:50
want to move the pipeline to streaming obviously that should be something that should be taken can very like you should
1:18:56

be thinking through that problem very a lot like definitely like they'll just bring in Flink like it you should only
1:19:03
be bringing in Flink after like you've really really thought about it and that's really the only thing that you should do right it's not necessarily
1:19:09
like an upgrade problem that's more of like a I need to solve a new use case problem but in some cases streaming can
1:19:15
actually be more reliable because of the fact that it's just processing data as it comes in as opposed to processing
1:19:21
some big batch of data and then that can make it so that uh the memory footprint of the job can actually be more
1:19:28
consistent and you don't have these big spikes in the memory footprint and uh because when you have a job that has big
1:19:34
spikes in the memory footprint that's when you have jobs that are more likely to run out of memory and uh especially
1:19:40
like in spark right there is that hard limit right you have that 16 gigabytes right where it's like if you have a
1:19:46
partition that ends up being bigger than 16 gigabytes for whatever reason because of skew or whatever other reason then uh
1:19:53
it's going to be more unreliable than um a streaming job that potentially
1:19:58
won't have that same Spike because of the fact that it processes the data as it comes in so that can be another uh
1:20:05
potential thing that could uh increase the Rel reliability of your pipelines so uh technology upgrades pretty
1:20:11
cool so sampling I love sampling uh I I want to talk a little bit about an example uh kind of War story from my
1:20:19
time at Netflix so um one of the things I worked on at Netflix was how to create
1:20:24
a metric that measured the uh the amount of incremental AWS cost from an AB from
1:20:32
an AB test so it's like say you have two uh like a test and control group in an AB test and one of them causes the app

1:20:39
to make more requests to the server and then that server will send more data down but maybe the like it's 10% more
1:20:46
requests so then the server the server load is 10% more or something like that and so like the way that I I was looking
1:20:54
at that originally was like okay I'm just going to look at all the network requests that um that Netflix receives
1:20:59
and it's like that's a very big data set that data set is hundreds of terabytes an hour and um and the metric that we
1:21:08
were like the and if you're processing hundreds of terabytes an hour that's millions and millions of dollars a year
1:21:13
in cost like in actual cost to the company so and our metric that we were making was probably not going to save
1:21:20
that much money so like we realized that like okay like there's an Roi here where it's like we need a sample we need to
1:21:26
like cut the data down and maybe only uh we ended up using I think less than 1% and uh in since we were able to cut down
1:21:32
the volume and because we only need the directionality of the of the sampling and or of the of the data not the whole
1:21:39
like data set for auditing and so that's a good um way to do it uh before you set
1:21:45
up sampling though uh make sure that you don't just like R like just set up whatever sample thing you want to do
1:21:52
this is something where like you do want to consult a scientist about the data shape so that like because it's very
1:21:58
important to get a random sample but for example here is another thing to think about with um like requests I think this
1:22:05
is a beautiful example where like I think as data Engineers this isn't isn't something that we think about that much
1:22:10
but like okay there's two ways that you could sample the uh the the requests
1:22:16

right where it's like you could take a small percentage of all the requests and just do randomly take a percentage of
1:22:22
all the requests or you can take all the requests for a small percentage of
1:22:28
users and then then you get the whole flow for every user but you only take a small number of users or do you just
1:22:34
take a small number of all the requests across the board and those are you could get the same percent you could get the
1:22:41
same like reduction but it's all about like what rows you pick and what rows you filter out and that makes a very big
1:22:48
difference on the data set itself and like what you're trying to solve for like for example in this case uh the
1:22:54
better solution here was to pick users and get the full request history for
1:23:01
individual users not filter to like random random uh requests across the
1:23:06
board because then uh you get the full kind of makeup of like the whole like request flow for each user and that's a
1:23:12
better um aggregation model than like in this other case where it's like you have like one request for this user two
1:23:18
requests for this user and it's like kind of like you don't have their full history so like the analytical capabilities of that way of sampling
1:23:25
versus the other way of sampling is significantly different so that's where like don't just like be like like
1:23:32
because it's easy to sample very very easy to sample like it's a oneliner to put that into sampling but um don't like
1:23:38
uh don't do that like make sure to ask a data scientist like that's my main point there unless you are amazing at
1:23:45
statistics and you already have the data science skills then maybe do it yourself but I know I don't that's one of the
1:23:51
things that I I recognize that I want to defer to a data scientist about because that will make um the whole process a

1:23:58
lot better so remember sampling is another beautiful tool in your tool built if your pipelines become more
1:24:04
unreliable and you don't need the full data set okay uh but like when should you not
1:24:11
sample I think this is another important question uh there's a good example here of like um if you need the entire data
1:24:18
set for auditing purposes or whatever then sampling is not going to work CU you're you might be looking for a needle
1:24:24
in a hay stack and uh if you sample then the needle is gone and like and you're not going to find all the needles that
1:24:30
you are looking for so audit purposes NOP nope
1:24:35
nope okay so that's sampling audits don't work go to the next one bucketing
1:24:42
bucketing is very important I'm going to give a couple examples of when I used bucketing in my career that have were
1:24:49
just mindboggling mindboggling how good they were right so um
1:24:55
uh big thing with bucketing right uh is so when you have a join right so if you imagine you have a join of two data sets
1:25:02
uh you have this big thing called Shuffle right where you can imagine you have like the left and right side of the
1:25:07
joins and then um how it works as the join key that join key gets hashed into
1:25:14
uh a number and then that number gets moduled like which is like you divide and you take the remainder and then that
1:25:21
is the partition that that row gets assigned to and mostly this is done with um like the
1:25:28
modulo the default module in spark is 200 because you want to have 200 partitions and then that's how it works
1:25:35
uh the problem with a lot of this stuff is that you that whole process of slicing all the data up and moving it around and then moving it across and
1:25:41

then matching it up is very very expensive that shuffle operation is very expensive so what you can do instead is
1:25:48
when you write the data sets the the left and right side say you say they both have user ID they're joining on
1:25:53
user ID then uh you can write the left and right side data sets to their own partitions
1:26:00
essentially so instead of writing like one big parquet file you write like maybe
1:26:05
1,24 smaller par parquet files that do that modulo operation already so that
1:26:11
you have them in 1,24 files and then and then the right side is also in 1,024
1:26:17
files so when you do the join you just match up the files right
1:26:23
and then the files just match up and then you don't have to do all this crazy there's no Shuffle Shuffle doesn't exist
1:26:29
if you if you bucket so and that is uh I highly recommend listening to the
1:26:35
speaker series with Sundar uh he's a a senior staff data engineer at Apple and
1:26:40
he talks a lot more in detail about how he uses bucketing to solve hyperscale
1:26:46
problems at Apple at the petabyte scale so bucketing is one of the most important uh optimization techniques
1:26:53
especially for the highest volumes of data because once you have the massive
1:26:58
massive volumes of data that join where it does that shuffle technique that shuffle Shuffle is broken like once
1:27:04
you're over like 10 terabytes Shuffle is Shuffle just breaks down and becomes
1:27:09
very very very expensive so that's where you can get a massive performance gain if you just write both of the left and
1:27:15
right side of the join to bucketed tables because then you can do the join without Shuffle and it's very performant
1:27:23
and so when I was at Facebook this is a big thing I was doing in notifications was I had the same problem where like uh
1:27:30

the notifications table I had like this so one of the tables was called notif Data features which was like um it was a
1:27:37
a like a machine learning table that had for every notification it had the um like a vector of features of all the
1:27:44
different features of that notification like when it should be sent what content type it was the probability that someone
1:27:49
was going to click on it the probability that someone was going to ignore it like all sorts of different um uh like a big
1:27:55
Vector of like hundreds and hundreds and hundreds of things right and then on the other side you have the actual events
1:28:01
you have the oh they clicked on it they sent it right because you need to merge these two data sets because then you
1:28:06
have the features and the facts and if you merge them together then you can actually train the model again because
1:28:12
then you're like oh we predicted that there was a 90% chance that they uh click on it and then they didn't so
1:28:18
that's going to maybe drop the next time they generate a notification that's like that notification it's going to drop it
1:28:24
a little bit cuz that's going to retrain it but anyways that join was actually at the notification level right cuz you're
1:28:30
joining the notification features and the notification facts you join them
1:28:35
together and then like that join was very very large and so what I did in that case was you join on notification
1:28:42
and on user and if you do that then um and it was bucketed on user so then you
1:28:48
just line up all the files that are based on the user ID and bucket it and then you can join those together and
1:28:55
then uh that removes all the shuffling that was in there before and then that made that it made that job uh go from
1:29:02
taking so before that job was taking up 25% of all of notifications compute and

1:29:08
then after I made that optimization it took up 4% so we got a massive massive
1:29:13
increase in uh efficiency and also a ton of compute back and it made it so our Nam space was actually like working
1:29:20
again it wasn't just like on fire all the time and so that was a big win that we were able to do with bucketing so um
1:29:28
I highly recommend just checking out bucketing as well like we're going to do a little bit of bucketing um next week
1:29:33
in the spark week if uh yeah because you're all in the infrastructure so y'all definitely be in the spark week so
1:29:39
uh it's good stuff like I definitely have used it a lot I've used it at um at
1:29:45
uh at Netflix as well and some other very hypers skill use cases uh so that's
1:29:50
kind of like uh another big thing that I would recommend looking into into there are drawbacks though I think that I want
1:29:56
to talk a little bit about how like it's not like um like because just because it
1:30:03
solves these hyperscale problems like I'm not advocating that every table be bucketed and one of the main reasons for
1:30:10
that is that say you bucket everything to 1,024 files and it's like a small
1:30:15
table then when you read it it has to it has to read in 1,024 files and there's
1:30:22
like opening and reading that many files is slope and like it's worth it when the
1:30:27
when there's a lot of data but like if it's not very much data then the cost of like opening and reading all of those
1:30:33
files is is higher and more of a pain in the ass than if you uh just don't bucket
1:30:40
so that's why like this is something that you is kind of more reserved like for tables that are over at least over a
1:30:46
terabyte like I would not recommend bucketing on anything under a terabyte so that's uh that because if it's under
1:30:53

a terabyte you might as well the shuffle and you're going to get the like probably a a better performance uh
1:30:58
characteristics so that's bucketing bucketing can save your life okay so another big thing that can
1:31:06
happen another big signal for Tech debt right is a large Cloud bills
1:31:13
so um one of the things to remember about the cloud is there's essentially
1:31:18
three places that you pay you pay in IO you pay in compute and you pay in storage those are the three places that
1:31:24
you pay um with IO usually being the biggest one and cuz the main reason why
1:31:31
IO is the biggest one is compute is generally fixed right because
1:31:37
you like rent your computers on like a fixed basis and then like you either have enough or you don't have enough and
1:31:42
then if you don't have enough you need to buy more computers and great but like if you don't have enough or I mean if
1:31:47
you have enough then like you're going to pay the same amount anyways and then storage is just generally really cheap
1:31:53
like especially in comparison to iio and compute um and one of the things one of
1:31:59
the ways you can think about that is okay uh move so moving data around and
1:32:06
moving it from point A to point B is going to be one of the most expensive operations that you're going to have in your Cloud bill this is where like uh
1:32:13
one this is probably the one argument that I have for uh the signal table
1:32:19
pattern if yall remember from week three we talked about the difference between write audit publish and the sign sign
1:32:24
table pattern and this is going to be my and this is Facebook's rationale as well
1:32:29
of why they use signal table over the write audit publish is because write audit publish actually does a partition
1:32:36
exchange so that means that every time that you write a table with write audit
1:32:41

publish you actually essentially do two times you do IO twice right you do IO
1:32:47
into staging and then IO from staging the prod right and so that's where you get you you pay your IO cost twice which
1:32:54
is pricey it's not cheap like it's actually one of the it's like the most expensive part of your Cloud bill so um
1:33:01
that's where that's why it's not as cut and dry of like you should use right audit publish or you should use signal
1:33:06
table because of the fact that IO is just so expensive so expensive and uh
1:33:12
that was one of the things that I noticed like because when I was working um uh at Airbnb and I found some massive
1:33:19
wins uh in storage and compute uh for Airbnb when I was like Computing the
1:33:24
savings that I I I saved for them it was like essentially it was like okay I was
1:33:30
able to save like five like 5% of all the savings was in storage and then 10%
1:33:36
was in compute and then 85% was in IO and obviously this depends on the
1:33:42
pipeline as well because the in that case the reason why that was the case for me is because the pipeline I was
1:33:49
working on had a lot of Downstream usage so if I was able to make the the storage footprint of a
1:33:56
table that has a lot of other pipelines reading it then that's actually where
1:34:02
you get the big win the big uh savings is where they read in that smaller table
1:34:07
every day and as opposed to reading in a bigger table every day and then then your the savings you get is multiplied
1:34:13
by the number of Downstream pipelines and that's why you get a massive IO win because you Scale based on the number of
1:34:20
other pipelines that are reading from your data tables and so that's where like that's why IO can be so massive as
1:34:26
like a win but obviously like if you're in the case where you're at the bottom of the um like data engineering tree

1:34:35
where like your pipeline is producing a data set that is like visualized in a in
1:34:40
a dashboard and it's not master data but it's like more of that that that final like metric aggregation layer then um IO
1:34:48
is not going to be your biggest cost it's going to be compute probably it's going to be or it's going to be close between compute and storage probably
1:34:54
compute probably compute and then storage would be your uh highest costs because you don't have that
1:34:59
multiplicative effect of all those other pipelines reading from your pipeline so Cloud bills it's freaking
1:35:08
there's a lot more we're going to talk about here with Cloud bills because I think this is a big area where you can have a big win both in terms of
1:35:14
efficiency and in terms of Maintenance okay so let's talk about uh
1:35:20
why too much IO can happen uh okay so obviously the first one is duplicative
1:35:26
data models where you have uh like when I was at Airbnb there was when I first started there there was seven different
1:35:31
definitions for availability like is available there was seven different ways that people Define that and uh and it's
1:35:38
like why like why are we doing that like and I mean it's like that pain is
1:35:43
actually that pain is not just in IO cost right that pain is also in communication overhead where it's like I
1:35:49
say available you say available but we mean two different things and so like you also like in some ways that's even
1:35:55
the bigger cost is on the communication overhead because employee time is just so expensive so like that's uh but like
1:36:02
IO cost as well because it's like why do we have five different tables for this that where when we should have one and then those pipelines write every day and
1:36:10
they are generating these tables every day and uh so that duplicative data models not enough robust data modeling
1:36:16

big place where you eat an IO you have inefficient pipelines so imagine if you
1:36:21
doing like the monthly active user pipeline and what you do is you just read in the last 30 days of fact data and you do it
1:36:29
that way and you make Zach sad because like I taught you better than that and like I taught you to use cumulative
1:36:35
table design and because you don't need to scan 30 days of data to process this every day you can scan one day of data
1:36:42
and then the his the historical and then one day of data to calculate monthly active users and so that's another place
1:36:49
because in those cases if you're scanning 30 days of data every day that's a lot of IO right you're reading
1:36:54
in 30 partitions every day when you could be reading in one another big place where you can eat a ton of IO cost
1:37:01
and I'm always like so painful so painful ow um you also have uh excessive
1:37:08
back fills where uh maybe you are trying to backfill a data set and like you're
1:37:13
kind of hasty about it and you didn't uh validate all of your pipeline code yet
1:37:18
and you didn't like really thoroughly validate it this is where um airbnb's midest program like they try to avoid
1:37:24
these IO costs by they say first backfill one month of data and then validate that and then if that looks
1:37:30
good that one month of data looks good then backfill the the the whole history of the data set because then uh you
1:37:37
don't pay these IO costs over and over and over again if there's a bug because that's happened to me before i' and I've
1:37:42
been there I've made this mistake myself many times like probably three at least three or four times in my career where like I've backfill seven years of data
1:37:49
and then the analyst come back comes back to me and is like these numbers don't add up and then I'm like wow like
1:37:54

I just gave I just gave Jeff Bezos like $5,000 I'm sorry Jeff or like you're you're welcome Jeff and I'm sorry uh
1:38:00
Brian chesky or um or Reed Reed Hastings or Mark Zuckerberg I guess Mark
1:38:06
Zuckerberg is he owns his own infra so like it's not it's different there but at uh at Netflix and Airbnb they both
1:38:12
like give Jeff Bezos money so anyways uh backfills it's another uh way that you
1:38:17
want to be careful make sure to follow backfill best practices and to be very confident that uh the day that you're
1:38:23
going to back fill is going to be correct so that you don't just do it all
1:38:29
over again cuz because back fills are very expensive because it's like it's like running all of production for seven
1:38:34
years like that's like a lot if if you're depending on how much history you're trying to back fill uh then you
1:38:40
have not sampling uh this the case where like you feel like you have to use the entire data set when you don't actually
1:38:45
have to use all the entire data set you can do it without all of it uh that's a another kind of common problem I've seen
1:38:52
uh also not subpartitioning correctly so subpartitions can be great because especially if your data table has
1:38:59
another low cardinality column where you can that like you can split the data up in a more uh a cleaner way we're going
1:39:06
to talk about that I have another slide here because predicate push down can be amazing in that case because if you have
1:39:12
a subpartition what it allows you to do is it allows you to avoid reading in a
1:39:17
big chunk of data that you don't need to read in and it because it it just ignores that folder it gives you another
1:39:24
layer of folders and it says oh our query doesn't want that folder because it's filtering it out and we don't even
1:39:30

have to read in the data we can just ignore it and so that's where subpartitioning can be very very powerful in uh avoiding large Cloud
1:39:38
bills there is some trade-offs there and like I think that there are like there there are definitely cases where I've
1:39:44
seen sub partitions used incorrectly as well but that's generally speaking this is how I would imagine things would work
1:39:50
with uh large Cloud bills those are going to be your big ways that um uh big
1:39:56
symptoms or or you know big causes of large Cloud bills that you can probably fix let's talk a little bit about why
1:40:03
subpartitions work so imagine you have a table here we're call this table notifications and then this table has
1:40:09
three partition or it has a subpartition on channel so it has a partition on date and a partition on channel and you'll
1:40:16
see okay we have three um folders here we have a date we have one for January 1st January 1st January 1st for email
1:40:22
SMS and push and one of the things you'll see is we have a query down here select star from notifications where
1:40:28
date is January 1st and channel equals SMS so it you'll see when it when this
1:40:33
query runs in spark it will skip this folder and it will not read in this data
1:40:39
set at all so you don't pay the io cost where like if this wasn't subp
1:40:45
partitioned right and say we were only uh partitioned on date then what would happen is we would read in all three of
1:40:51
these data sets and then throw all of them away that didn't match SMS and then that would be extra IO cost that we
1:40:58
wouldn't want to pay this is a great way to avoid that IO cost by like just immediately filtering it out because we
1:41:05
know that uh none of the data in this um none of the data in this folder is going to be valid and so that is definitely
1:41:12

one of the powerful things that you can do with sub partitioning especially if you have a sub partition that is kind of
1:41:18
low cardinality um like I generally think of like if you have a subar partition it should be uh like it's
1:41:25
similar to enums remember how we were talking about enums before where enums are like uh it should be like like 30 or
1:41:32
less if you have like if it's like a we have the number of unique values is like
1:41:37
30 or less and usually it's like five or less but there are some edges where there's a little bit more but generally speaking it should be like 30 or less
1:41:44
values this is where like country is always the one that's like some people are like yeah we should sub partition on country and it's like no dude don't do
1:41:50
it like country is going to be annoying then like then you because if you do country then you're going to have to have like 200 tasks and that's super
1:41:57
annoying so um you want it to be more of a low cardinality field that you and and
1:42:02
one that people use in their filters a lot right because one of the things that was really powerful about this um
1:42:09
subpartition in notifications is um one of these notifications types is a lot
1:42:14
bigger than the others right push is two orders of magnitude bigger than SMS so
1:42:21
it's like if you wanted if you didn't sub partition and you wanted to run a query that looked at SMS you would be
1:42:27
like you would be reading in all the data and then throwing 99% of it away
1:42:32
and paying that IO cost because you read it in you you read it off of S3 and just
1:42:37
threw it away and just wasted it right and that's where subpartitioning can be very powerful in uh giving you more of
1:42:44
an IO efficiency gain that I highly recommend looking into on especially if you have any big fact data that's where
1:42:52
you're going to see some real big gains here so let's talk a little bit about um
1:42:57

how these things are connected actually because IO compute and storage are not uh independent where like if you have
1:43:04
bigger tables that are stored in bigger places and you read in more data like you're going to have more IO and so
1:43:10
storage and IO are obviously connected and compute is also connected to these two right so um like what I was saying
1:43:17
before uh large iio and compute are connected a lot because if you're scanning too much data you're going to be reading in a lot more data and then
1:43:24
you're going to be running a lot more uh that's going to tie up your spark executors for a longer period of time
1:43:30
because that IO time is just like expensive so you want to make sure that
1:43:35
you are not doing that uh sampling is another great option here besides cumulative tables sampling predicate
1:43:40
push down all that stuff um also um make sure that you're like in your udfs you don't have any um N squared algorithms
1:43:48
in that case that's where you have like Loops inside of Loops if you have nested Loops in your algorithms you want to like make sure to kind of avoid those so
1:43:55
there are some small edge cases where that's the only way to solve the problem but like n 99 times out of 100 that I've
1:44:01
seen a nested Loop in production it's it's Tech debt it's not like oh yeah that guy just had the the best option he
1:44:07
he knew exactly what he was doing and he like he thought through everything now he is just trying to solve the problem in the quickest way he thought of so um
1:44:14
IO and storage are correlated um because if you're not leveraging Park the file
1:44:20
format the best then you're going to get a lot more IO cost because uh your storage tables are not compressed enough
1:44:27
and if you can compress your data tables with paret uh then your IO is less
1:44:33
because when you read in the data it's compressed and then it only becomes uncompressed when you are already in the

1:44:38
co you're in the executor and you're already on the machine so then you don't have to worry about uh um you don't have
1:44:45
to worry about the io cost at that point because the io is already done and because it's it's compressed already and
1:44:52
so and then obviously the other one is duplicative data models just like what I said earlier where it's like if you have seven different definitions for the same
1:44:57
thing you're going to get a lot of a lot of extra IO a lot of extra storage a lot of extras it's a lot of so that's that's
1:45:03
why like data modeling is so important right and so so critical okay now this is uh this is one
1:45:11
of the things that I find a little bit more near and dear to my heart because I think this is probably the most
1:45:16
impactful work that you can do as a data engineer and it makes your job easier
1:45:23
and it makes things more efficient and you the maintenance is better it's like
1:45:28
you win in every single way and there's no way that you lose that's the part
1:45:33
that is so crazy about solving this problem here and like so if you can solve these problems this is where
1:45:39
you're going to you're going to really find a lot of impact as a data engineer and this is like solving these problems
1:45:44
is also like where I got really good performance reviews at Airbnb and all sort all sorts of stuff like that as
1:45:50
well and so let's uh let's dig a little bit into this so it's very common
1:45:55
especially as an organization gets bigger and bigger and they have more and more needs for data that uh they don't
1:46:03
have enough data Engineers or they don't have enough data infrastructure and like what ends up happening is like different
1:46:11
aspects of the company are like well I need this data set and I just need it as quickly as I can get it right and then
1:46:16

they don't really care as much about the quality and they don't really care as much about like if other people have already created it they might search
1:46:23
real quick to see and then if it's there great if not they'll go make it themselves and then they'll keep doing
1:46:29
that um so this work is complicated
1:46:34
because of the fact that one when people do these multiple
1:46:40
sources and they have multiple pipelines that are producing the same thing um
1:46:46
they might have different variations of the same definition so usually that
1:46:52
means that you have to convince somebody that their definition is wrong because
1:46:58
um or maybe not wrong but at least kind of not correct and that is going to be
1:47:05
can be tricky sometimes because some sometimes even people who are not data Engineers are kind of married to the data that they created and they're like
1:47:11
no I I this is the most correct way to do it and so you want to get all these
1:47:17
stakeholders in a room so like for example for me when I was like trying to solve this like many many many different
1:47:23
many many many different definitions of availability at Airbnb like I had to get so many people
1:47:30
in a rooms and just talk it out and see like how we could solve this problem and going forward and like both like and the
1:47:37
thing that was crazy about the way that I solved that problem was that we didn't end up picking any of the ones that
1:47:43
already existed we created a new definition and then everyone agreed on
1:47:49
that definition because they were like yeah that's the most correct definition so sometimes that's the other thing
1:47:54
about these multiple sources of Truth is that they can all be wrong or they can all have problems as well and that like
1:48:01
there's actually another path forward that might be uh the most correct or the
1:48:06

most efficient or the best and so just because there's multiple sources of Truth doesn't necessarily mean that one
1:48:13
of them is you know quote unquote the truth that that there's actually uh um
1:48:19
potentially room for improvements on all of them and so a big thing to to look at
1:48:25
here is you want to document all the sources of truth that you can find this is a big thing I remember I had this
1:48:30
spread sheet at Airbnb that had like I was like it had like 30 or 40 tables in
1:48:35
it and like I was like if we can just consolidate all of these tables can go
1:48:41
away and we don't need any of them and we can go from 40 tables to four and it
1:48:47
would be better it' be a lot better and that was the vision I sold my manager on and I'm like it's going to be so much
1:48:52
better and you need to document these things It's Tricky though because of the fact that like one people can name a table
1:48:58
whatever they want so it can be tricky to even necessarily find all the sources of truth that's why like generally
1:49:05
speaking you don't find all the sources of Truth via code search or via like
1:49:10
grap or however technical way you want to do it that can be a good way to get a good like fundamental starp but like you
1:49:18
really do want to talk with all the relevant business bu people all the relevant people who like in all the
1:49:26
areas of the business that might be using this data because then uh you'll you could uncover some really funky
1:49:33
table names things that you didn't even consider someone might have named their table qqxx l llore w and they're like
1:49:40
yep that's availability and it's like okay bro okay it's availability I guess
1:49:45
right and um anyways uh you can't just rely on like searching the codebase to
1:49:51

find these tables you need to talk conversations it's important and these conversations are also important anyways
1:49:58
because it can help you understand like why they're using this data and also
1:50:03
understanding maybe the pain that they're having with this data or the the the lack of completeness or the
1:50:09
difficulty or like all sorts of other things that you also want to pick up right where like if you're part of the combined track uh yesterday we were
1:50:16
talking a lot about being a product manager and this is a very relevant place to be here where like if you're
1:50:23
building a pipeline you want to be a data product manager where like you want to understand end to end how to Delight
1:50:30
your customers where a big part of that is completeness another part is quality
1:50:35
another part is just refresh freshness quality completeness ease of use all of
1:50:41
those things are very important and then also if you take a pipeline off of someone's plate they'll love you forever
1:50:48
because then they're like wow I don't have to manage this Beast anymore and you will just manage it all for me like thank you
1:50:53
so like that's why you also get a lot of claps and a lot of um kudos for doing
1:50:58
this as well because like in in in my case at Airbnb there was like okay there was seven different pipelines at least
1:51:04
of you know generating different availability definitions and then like then we turned it into one that we owned
1:51:09
and then it was like okay all those other pipelines that people were using we don't need anymore and so that was another way to get a lot of Kudos a lot
1:51:16
of respect right is you're deleting work for people and deleting maintenance for
1:51:21
other people it could be tricky because sometimes it's like you're deleting maintenance for them but you're putting more maintenance on yourself so you want
1:51:27

to like have a good balance of all that stuff obviously you don't want to try to be a superhero um and then after you
1:51:33
have all the needs and you have taken in all the pain and you've documented all of the discrepancies and all of the
1:51:40
different sources of truth that's when you want to build a new spec and you want to build out the spec of the new
1:51:47
pipeline that you want to build that's going to consolidate all of these things and be the New Path forward that then
1:51:54
you can talk to all the stakeholders about and be like this is what we're going to go to we're going to migrate to this it's going to save you all this
1:52:00
pain can you agree to it and you want them to agree to it before you build the spec or before you start writing the
1:52:06
code because they might not agree to it and like if they don't agree to it then uh the problem is is then like if they
1:52:13
don't migrate then a lot of your work is for nothing cuz then they're still going to be using their own other definition
1:52:20
and then like all of that hard work of consolid validating definitions is not like is just dead so it's all about
1:52:26
getting that stakeholder Buy in making sure that everyone's on the same page most of the time it's easy sometimes
1:52:32
it's a little bit harder but most of the time people are like grateful that you're doing it so those are the three
1:52:37
steps like and a lot of that's a conversational thing not very not very technical actually it's very um human
1:52:43
very empathetic very product manery right remember I was saying you know
1:52:50
talk with all of the stakeholders code search for similar names like you know you might want to GP for availability or
1:52:57
something like that and then uh you can also like potentially if you're companies's really ahead of the game
1:53:02

they might have lineage so then you can like go up to the source data because a lot of times even if you have different
1:53:08
definitions of something they all read from the same Source or at least some of the same sources so then you can find
1:53:15
like the most raw log data or the most raw like database snapshots and those
1:53:20
tables are going to like the the down stream of those tables are going to be the the definition tables that you're
1:53:26
looking for and that can be another way to find that stuff not very many companies have lineage though lineage is
1:53:32
like kind of a new thing I think that that's going to be in the future I think that's like going to be a good thing that's one of the things that uh you
1:53:38
know have you heard all Microsoft fabric I've heard that fabric gives you that like out of the box so if you just like
1:53:43
use fabric then you get lineage for everything and like that's cool I like lineage I think that like being able to
1:53:48
see where data comes from is very powerful and it allows you to do a lot of this good work as you kind of go
1:53:55
forward with stuff so definitely do that um okay another thing that you want to
1:54:01
do when you're like consolidating all these sources of Truth is understanding like how it got here because there's
1:54:08
more problems here than just the fact that there's seven data tailes like the there's other problems here that are
1:54:14
also important to talk about and also important to bring up um for example one
1:54:20
of the things that can happen in a lot of these cases is you can have organizational um problems where what
1:54:29
actually happens why there's multiple sources of Truth is one team doesn't trust another team to deliver and so
1:54:34
they just go and build it themselves and it's like why is that the solution right
1:54:40

and it's like when like that's where trust is super important where you need to have trust between teams otherwise like you're going to have this you're
1:54:47
going to have Tech debt like this come up all the time you have an ownership problem as well like uh for example the
1:54:53
team that should own this doesn't have the bandwidth to own it so another
1:54:58
team's like okay fine we have a little bit of spare bandwidth so we're going to take it on and that can be another kind of problem but then they aren't the
1:55:05
right team to own it because they actually don't have enough of the business domain to actually do a good job so that's like more usually that's
1:55:12
more of like a staffing problem where they they should probably move one engineer to the other team so that then they can own that that data set which is
1:55:20
very hard to do by the way um um then you have technical technical would be like okay uh the team that should own
1:55:27
this doesn't have the technical skills to own it and then one of the teams that we have does have the technical skills a
1:55:34
lot of these are a combination of organizational and ownership and Technical right te Technical and skills are the same way and obviously there's
1:55:40
going to be a long tail of other reasons why we have duplicative data sets
1:55:46
potentially coming up but the whole thing about this and why you need to have these conversations especially if
1:55:51
you're like to become a data leader is if you can solve the organizational
1:55:58
problem if you can solve the trust problem and the ownership problem then you stop the bleeding you stop this
1:56:05
problem from proliferating and then you can get people to trust each other more and then you can get more Consolidated
1:56:11
data sets that way and then less maintenance more efficient awesome right
1:56:17
it's really good and then like another thing that's super important is like working with these stakeholders
1:56:22

uh to understand what the ownership model should look like and also like how changes should be incorporated like if a
1:56:30
model if the model does need updates later on like how who's going to drive that and who's going to build that out
1:56:35
right and really hammering that down so that people uh in these other teams can trust that this data set will not just
1:56:43
serve them after it's migrated but that it will serve them in the future as well and that could be another great way to
1:56:49
build trust and to build uh just just just to build a lot of like uh Rapport
1:56:56
and to make people really like you as well cuz they're like they're cuz the you're taking something off of their
1:57:01
plate and also guaranteeing to them that it's going to be it will be managed in the future as well that's awesome that's
1:57:08
like you're taking work off you're taking work from them at immediately and future work so like that like that's a
1:57:15
that's a that's a lot of love you know it's a lot of love so remember to spread the love
1:57:20
around I love specs pipeline specs are so great right and that's what I was
1:57:26
saying earlier got to build a spec get all the needs get everyone to sign off and then because once they sign off and
1:57:32
you have like a signature from them or you have like a like they said this was good then that is like when when it comes time for them to actually move and
1:57:38
migrate their Downstream pipelines like you can kind of bully them into doing it because you're like Yo dude you said you were going to do it and because
1:57:45
migrations are so painful they're always so painful and uh and like having a
1:57:50
document like this could be very helpful in uh getting people to actually do
1:57:55
it let's talk about a little bit about the different models for fixing Tech debt of like kind of the team ways of uh
1:58:02

fixing Tech de that I've seen these are essentially the three that I've seen where you have uh the first one is
1:58:08
called like fix as you go it's also called like boy scouting it's also called like a bunch of things like that
1:58:14
but it's like mostly like when you're shipping new features fix the tech debt as you go and it's it's it's an
1:58:21
interesting model we're going to talk about the pros and cons here in a second uh then you have like allocate a portion of time each quarter so a lot of times
1:58:27
this is what Airbnb did was the second one uh it's called like Tech Excellence week which was the last week of the
1:58:32
quarter the last week of every quarter we uh just did not work on new features we only worked on cleaning up Tech debt
1:58:39
and amazing by the way I love it loved it so much and then uh the last one this
1:58:44
is the way that uh it worked at Netflix was uh instead you have the on call person uh focus on Tech de because they
1:58:51
get slacked in the face every day with the tech that because of they have to fix the on call problems and to deal with the maintenance themselves so then
1:58:58
we're like okay on call person just if you're dealing with the maintenance already and you're unblocking things why
1:59:04
don't you not just unblock but also fix things so you get kind of both in that way but these these three methods have
1:59:10
their uh kind of pros and cons all right obviously fix as you go uh the biggest Pro here is uh like it's not very
1:59:19
structured and like you can there isn't much incremental burden like you're not taking away from very much else um I
1:59:26
think the con here for this model is that like it sounds too good to be true
1:59:32
a little bit where it's like oh yeah as you're coding just fix the tech debt it's almost like as you're shipping
1:59:39
features fix the tech debt right it's like almost like as you're flying the plane fix the plane and it's like I
1:59:46

don't know about that fam I don't know if that's like a an actual viable model or not if it is cool if not like I don't
1:59:54
know I haven't ever seen it work in practice all right I I mean that's what Facebook was all about and Facebook
1:59:59
freaking was a mess in in terms of tech that Facebook was by far the biggest of the messes so
2:00:06
um then you have the allocate dedicated time each quarter right uh you actually fix things in big bursts it's great um
2:00:14
one of the problems with this though I found is that like people essentially uh if you do have that dedicated time at
2:00:20
the end of the quarter you don't have the as you go fixes and a lot of times
2:00:26
you get like you get this like Tech that line and then it's like cliff and then build up and Cliff right and it's like a
2:00:33
crazy uh and like so at the end of the like the the second to the last week of the quarter there's a lot more Tech de
2:00:39
in the code base right and then like and you're not like it's almost like it's almost like saying like oh I'm GNA save
2:00:46
up all my teeth brushing and I'm just going to brush one time for uh 90 minutes at the end of the quarter and
2:00:53
like I mean I don't know if that I don't know if that's how it actually works but uh that's um that's generally speaking
2:01:00
that's how I've seen it work uh the other minus here is then uh people during that week they don't have uh time
2:01:08
to help stakeholders or to uh ship other features or other things like that it
2:01:13
comes it comes at a pretty heavy cost for that week I mean you spend a month a year uh working just on Tech debt which
2:01:20
is a significant amount of time that you could be using to spend other things kind of like the opportunity
2:01:25
cost um having the on call person uh do Tech debt uh the cool thing about that
2:01:31

is that they are very aware of what the most urgent Tech debt things are because the those things are slapping them in
2:01:37
the face because they're like wow this is failing this is failing everything's on fire and then they have to get their fire extinguisher out and then they like
2:01:43
try to fix stuff right um I don't know like one of the things that I've noticed
2:01:49
uh with this model was that like one uh some people don't actually do it uh I
2:01:56
think that that's one of the things I like about the dedicated week as opposed to doing on call person do Tech Deb is
2:02:02
the dedicated week is like everybody is doing Tech Deb that week so you can kind of like there's this expectation that
2:02:09
you all ship something that you all clean stuff up and that like you're going to hold each other accountable uh
2:02:14
I found with the on when you have the when you say the on call person needs to do Tech that that they don't actually do
2:02:20
it that like uh I would say only like half the people actually do it and that was actually something that frustrated
2:02:26
me a lot when I worked at Netflix because I was like Yo dude like if this is our model like why aren't we cleaning stuff up why aren't you actually
2:02:31
dedicating the time to that and like a lot of times it would be they would want to spend that time working on new features or working on other stuff right
2:02:38
and working on the things that are going to like get them promoted because a lot of times Tech de stuff is uh not as
2:02:43
visible and so uh that it's it gets deprioritized and that's where that's why I really do like the the dedicated
2:02:50
time each quarter because then it's like well it's not as visible but we're all going to work on it together at the same
2:02:56
time and then it's like because it's a it's kind of a team effort in that way like it takes a village to um you know
2:03:03

tackle Tech de and so those are the three models obviously I biased y'all into saying which one I like the most
2:03:09
but I also think that the one that I like the most is the heaviest it's the one that takes up the most time it's the one that uh the way I think about it is
2:03:16
it's the one that really takes it seriously though it doesn't just treat it like a secondary prior prity or like
2:03:22
something that I'll get around to it if I can right it actually uh treats it as like we're going to block out time for
2:03:28
this every every quarter so uh definitely try it out all three tell me
2:03:33
how it goes like if y'all can figure out how to get the fix as you go to work I'd love it I think that like I want to hear
2:03:40
uh the success stories there because like if that actually works I think that I would like I I I think that that would
2:03:46
that's why I would want to do it but I just don't think that it's it's realistic
2:03:53
okay let's talk about the data migration models and then we will be almost to uh the Run books part of this presentation
2:04:00
so essentially there are two ways to go when you're migrating things and I have
2:04:05
a good story here because Airbnb changed their mind on this which was so crazy so
2:04:13
if you're building a new data model so you know in that example I was talking about earlier where we like built a spec and we got all the stakeholders sign off
2:04:19
and we build a pipeline and we back fild a pipeline and we have a pipeline L running in production and then there is still that time between when you have
2:04:25
that pipeline running in production and when other people migrate and they move they they depend on your data instead of
2:04:31
the old data there's like a there's that time and there's like these parallel
2:04:37
pipelines that both need to run and one of the things about that is it's kind of expensive because then you're
2:04:43
essentially paying 2x uh the cost of that pipeline for the duration of the
2:04:48

migration and that's like painful full and one of the things I noticed is that
2:04:53
like when that happened I uh like so initially the migrations I was working
2:04:59
on at um Airbnb like when we made a new pipeline it would take like six months
2:05:06
for people to migrate and then we just have two pipelines for six months and then like sometimes it was like is this
2:05:12
even better like we're just adding another Pipeline and we're holding on to both for six months and this was actually a very common problem um at
2:05:18
Airbnb in 2021 where they were like uh and eventually leadership came out and they said like y'all are being too
2:05:25
cautious like if things break things break and people need to migrate because
2:05:31
the problem is like everyone is being too cautious about breaking other people's pipelines or making other people's data delayed and uh they didn't
2:05:38
drop the tables or stop the Legacy or the deprecated pipelines soon enough and
2:05:44
so what ended up happening later on they were essentially like you got you give them a month if they don't migrate in a
2:05:50
month you start breaking stuff that's essentially what they change their mind
2:05:56
right so that like so that we don't have this like crazy long period of time where like the migration takes forever
2:06:03
and people are just deprioritizing moving the tables and back filling and all that stuff because it's not fun it's
2:06:08
not fun migrating data is not fun it's literally the least fun part of data engineering all right and it's like the most boring least fun part of data
2:06:15
engineering so like I get why people don't do it and so that's why like uh they only want to do it if they have to
2:06:22
and the way that you force them to do it is you delete the old Pipeline and you make it so that their pipeline just
2:06:28
doesn't run anymore because now their pipeline can't run because the Upstream data doesn't exist and the only way that they can uh

2:06:35
get their pipeline to run again is by migrating and that's why there's this new the new one is called the bull in a
2:06:40
china shop approach where it's like okay once the new pipeline is up and running like might as well migrate things so
2:06:47
like the way I did it when I was looking at like pricing and availability stuff was I was like okay we're going to move
2:06:53
uh any of the ml stuff any of the expensive things that like if they broke
2:06:59
and we're delayed then we're going to migrate those and then once those are migrated everybody else can just break
2:07:05
and then they can migrate as soon as they can right and then they can just get their stuff to migrate because it
2:07:10
doesn't matter that much right because if it's delayed by a week it still doesn't matter that much because their
2:07:15
data set isn't that high impact anyway and it's actually more impactful
2:07:21
uh on the tech debt buildup and like on my own team's well-being to have two
2:07:27
pipelines running in parallel for and number of months so this is something
2:07:32
that like I found very interesting and I was very grateful that Airbnb changed their policy on this because freaking
2:07:37
that first one was so painful where it was like essentially how it worked was like it was like three months where uh
2:07:43
they run in parallel and then after three months you rename you uh you keep them both running in parallel but the
2:07:49
the deprecated table you rename to like table underscore do not use and then you have it run for another couple months
2:07:56
and then uh and then drop it off at that point and I was always like dude like these zombie tables just need to
2:08:01
freaking die man like why aren't these dying right and so definitely uh they both have their benefits and risks

2:08:07
though because um the cautious approach the thing about it is though is that's a very um it's a very accommodating way of
2:08:16
doing data migrations cuz if you just Break Stuff the thing is is like the other team is going to be like hey you just work on my plate like why you doing
2:08:23
that why why you being a dick why why why you putting work on my plate you don't want yeah you don't want that right so um and so but like they learned
2:08:32
that that's actually the better way to go that like it's because it's usually not that much work to migrate so let's
2:08:37
talk a little bit about U um on call proper on call responsibilities uh so
2:08:43
there's a couple things here that I want to talk about uh so one is around uh set
2:08:49
up proper expectations with your stakeholders I think this is the most important part of on call because uh if
2:08:55
you can get this right then your en call Will usually be a breeze um because at
2:09:01
least like when I was working uh at Airbnb originally the um uh the the
2:09:06
expectation was that any on call Bug was going to be troubleshooted in four hours
2:09:11
and I did that for six months and I was like I'm not doing this like I'm not
2:09:17
four hours is stupid because then it's like if it fails at midnight I have to troubleshoot by 4 a.m. like no like I'm
2:09:23
not doing this and I changed it from 4 hours and I was like you know what we're doing we're doing 24 hours that's what
2:09:29
we're doing and like they there was initially push back because they were like why why are you making it worse why
2:09:36
like why can't you do that I'm like cuz I'm like cuz four hours freaking sucks like and like I don't want to have to wake up in the middle of the night and
2:09:42
like it's bad for my mental health and bad for my well-being and it makes me hate my job and like and the thing but
2:09:48

also I I told them that but also um another angle from it is that like it makes no impact on the downstream
2:09:55
there's actually no impact really on the downstream unless the data is more than a day delay so uh I'm like that's I
2:10:02
think good enough and uh that's where you can have some really good conversations with your stakeholders and the data engineer before you might have
2:10:09
been crazy and done things in a way that like wasn't sustainable for you and you might want to reset those expectations
2:10:16
and don't think like the expectations the on call expectations going into the job are the are what they have to be
2:10:22
forever like you can it is within your power to make changes to that um the other big thing document
2:10:29
every failure and Bug uh you'll usually get an email from airflow or an email from your job orchestrator if something
2:10:35
breaks uh make sure to document it and put it in some sort of Google doc so that you can keep track of all the
2:10:40
failures so that you can find the problem children so that like uh your on call can get smoother and smoother and
2:10:47
smoother and obviously you have like on call handoff which is where where uh that's where usually on call lasts a
2:10:54
week and then you'll pass it to another person and then that person will and then you have to give them all the context of like anything that is
2:11:00
currently broken or things that broke over that week and then that's usually like a 20 30 minute meeting to sync up
2:11:05
with people to get them to be ready for their on call rotation okay run books Let's talk about run books a little bit
2:11:11
uh and we're going to we'll have a little bit of time to maybe create a little bit of a run book so uh complex
2:11:17
pipelines need run books not every pipeline needs run books uh you only need run books if like there's pipelines
2:11:24

that have like a lot of inputs or they have outputs that are used by a lot of teams or they have a lot of complicated
2:11:30
logic or a lot of data quality checks or like you know things like that those are going to be the big things that you need
2:11:36
in the Run book and I think one of the things that's important here to differentiate is what's the difference
2:11:42
between a run book and a spec because they are similar uh and like some some
2:11:48
people actually say that they're the same and the Run book it should be be in the spec and that like the Run book is just like an evolving part of the spec
2:11:56
uh so that's a big thing to remember um so the big things that you want to put in the Run book primary and secondary
2:12:03
owners this is whether this goes in the Run book or the spec is like here or
2:12:08
there usually if that's in the spec it can be in the Run book too and that's going to be like essentially the primary
2:12:14
owner is like who you tap if if you're on call and a pipeline breaks and you
2:12:20
don't know how to tr shoot it the primary owner is who you tap and if they're on vacation you tap the secondary owner and if they're on
2:12:26
vacation like you freak out uh and like if a lot of times there isn't a secondary owner and that can be a
2:12:33
problem as well uh because I I found that big Tech doesn't do a good job at uh allocating two Engineers for every
2:12:39
pipeline but uh that can be a thing um another very important part of this is
2:12:45
listing all the Upstream owners because one of the most common things that happens in data pipelines is the
2:12:50
Upstream data quality checks fail and if you have Upstream data quality checks fail then you need to talk with those
2:12:56
teams and be like yo your data sucks fix it and uh you want this to be teams not
2:13:02
individuals the main reason for that is that uh Team ownership fluctuates less

2:13:07
than individual ownership because um reorgs happen less often than people
2:13:14
leaving the company coming and uh people entering and leaving the company so it's better to actually have uh the contact
2:13:20
point be a team than to have the contact point being an individual because then at least like that because that person
2:13:25
might leave the company and uh then you're screwed because you have no idea who like they're going to go to so uh
2:13:32
that's a big thing that I would IM imagine and like but even still you can have reorgs that happen and uh that team
2:13:38
identifier might also be kind of bogus so you want to be careful there um another big thing here is common issues
2:13:44
so this is usually part of like the common issues is something that that
2:13:49
gets filled out over time that usually the on call person should be taking some
2:13:54
of like if they're if they're copying and looking at all those stack traces and failures that are happening they
2:13:59
should describe what happens and then the common way to troubleshoot it right and the common process to troubleshoot
2:14:05
it assuming that they can't fix it more long term right they can't like it's like just kind of a fundamental nature
2:14:11
of the data as opposed to um uh something that is like a bug right where
2:14:17
there are pipelines that are like that like where you have like just like especially like I know like like scraping pipelines if you're like
2:14:23
scraping the web or whatever you're going to have a lot of common issues that like just happen because you don't have control of the source data um then
2:14:30
you have critical Downstream owners these are the people that you need to talk to these people are going to be the
2:14:36
people who like especially you don't need to like document all your Downstream owners like you just need to

2:14:42
document the ones that matter the most right for me it was like Smart Pricing was the big one that was like the ml
2:14:47
model Downstream where it's like I knew if the pricing data was delayed I had to talk with Smart Pricing quickly just so that they were aware that like the
2:14:54
pipeline was going to uh be kicking back up and that I was troubleshooting it and figuring out what was going on last but not least is slas and
2:15:01
agreements so slas here's service level agreement which is kind of duplicative here slas and agreements they're both
2:15:07
agreements and uh in this case this is like usually this is a number of hours sometimes a number of days number of
2:15:13
hours or days uh after midnight for that day when the data is expected to arrive
2:15:21
and this is the agreement essentially between you and your stakeholders that says okay the DAT is not late until it's
2:15:28
been this many hours after midnight and that's going to be uh how you can kind of prevent a lot of your stakeholder
2:15:36
questions from getting out of hand is by uh because they're not going to be like hey why is it late and then you're like
2:15:41
it's actually not late yet it's going to be late in like five hours so don't quit pinging me right now so that's a thing
2:15:48
to remember as you're kind of like going through this stuff uh so these are the kind of the critical aspects of a run book and if you can
2:15:56
have all these things in one dock then you're going to be in a good spot uh there is stuff that's a pain right um
2:16:03
but here's an example I think that I think will kind of give you a better kind of view of what I mean by how run
2:16:10
books can be helpful so you have over here on the left you have uh um kind of
2:16:17
your Upstream owners right so you have like a owned by Joe mama a monetization
2:16:22

team he owns coupons and credits and then Henry on the finance team owns revenue and then Karen on the customer
2:16:29
service team owns the customer service data then you have your pipeline and then these are your output data sets and
2:16:35
then these are your Downstream consumers that you want to talk to if this stuff breaks right and some of the times like
2:16:42
depending on how mature your data infrastructure is some of the stuff can
2:16:47
be completely automated so that these people will actually be aware of a delay or aware of a breakage uh without you
2:16:56
having to contact them which is great but like if it's a very long delay they're going to ping you and be like
2:17:02
why is the data late like why haven't you troubleshooted this right and so you want to be able to have a good
2:17:07
conversation and a good communication with that person as well so that you can have a kind of a a picture in your mind
2:17:15
of okay this is the person that uh I need to talk to and generally speaking
2:17:20
when you're building these pipelines this is just another part of this process that I find to be very valuable
2:17:26
which is not quite as technical but is a a good part of this process as well is
2:17:32
you should have a regularly uh recurring meeting with everybody on both sides of
2:17:39
this so for example in this case I would probably have three uh maybe quarterly
2:17:44
or monthly or quarterly one-on ones with Joe mama Henry and Karen and then ALS
2:17:50
also with experimentation and pricing teams customer service team and efficiency team I would have uh at least
2:17:58
a at least a quarterly if not a monthly one-on-one with each of those teams so that uh we can just be all on the same
2:18:05
page and then also just to understand like where they're trying to go and if we can improve some of these data sets

2:18:11
to make them better so that like you get like it's great because then in those
2:18:19
cases you just get way better connection with all of your stakeholders and you're just more integrated in the business
2:18:25
overall and you get a nice benefit from this of like hey we are um uh the the
2:18:32
your stakeholders will be nicer to you when there's problems because like like you actually showed that you care about
2:18:39
investing in the relationship over the long run so it makes your life easier because then you're like oh
2:18:46
like um because I I ran into this problem a long time especially early in my career when I was on call was that
2:18:53
like I would I would have stakeholders Downstream who like I were just like
2:18:59
unnamed right or like I just didn't even have a face to put on this stakeholder because there was just like a lot of
2:19:04
stakeholders or a lot of people right and then it was just like I don't even know I'm I'm I'm making Unknown People
2:19:09
angry by delaying their data sets and uh and so like and I was always that kind
2:19:14
of stressed me out and so I found that like you definitely want to do relationship building with all of your
2:19:20
up upstream and all of your Downstream consumers as a data engineer because then like you're going to just catch
2:19:25
things so much better and it solves the other problems as well it solves those uh duplicative data modeling problems it
2:19:32
solves like so much of the other kind of pieces of the puzzle so we going go create a new Doc here and then uh call

## Lab 1

2:19:39
this uh runbook for exactly Inc growth
2:19:46
pipeline obviously I'm a startup so like some I don't really have stakeholders but we're going to pretend that I do

2:19:51
have stakeholders so uh so in this case uh what we want to say here is uh you usually want like a title right we'll
2:19:57
call this on call runbook for exactly Inc growth
2:20:02
pipeline okay so we have like our Upstream data sets and this is going to be like uh
2:20:10
events we say like website events and then we have uh user exports then we
2:20:16
have Downstream um uh consumers we have we have experimentation platform and we have uh
2:20:25
dashboards say both like that right uh user database exports so these are going
2:20:31
to be our two kind of uh different ways of doing things right and so we this is
2:20:38
our we remember saying I was listing my upstream and downstream data sets right and then in here we have like common
2:20:45
anomalies right um or like uh sometimes
2:20:50
uh referer is null too much uh this is fixed Downstream but we
2:21:00
are alerted about it because it messes with the
2:21:05
metrics right something like that because you can also have like data quality checks that are non-blocking
2:21:12
right but they alert you because like they want to let you know that like you might need to talk with the dashboard users or the experimentation platform
2:21:19
and then you have like your user database uh you know export might fail um uh to be extracted on a given
2:21:27
day when this happens uh export as soon
2:21:33
as possible and just use tomorrow's or no use yesterday's that's
2:21:39
actually what it is when this happens use just use uh
2:21:44
yesterday's export for today something like that right depending on like how
2:21:49
reliable your infrastructure is obviously this isn't item potent and I hate it but like that might be what your

2:21:54
run book is as a company right and so this is the idea right so you have like your common issues like and you probably
2:22:01
want to call this like common issues right then this is like bolded and like
2:22:08
big right and then like obviously you have a primary owner this is going to be
2:22:13
Zach secondary secondary owner is Lulu right
2:22:18
and then we have uh well there was there was some more pieces of this right we have the um run books right so uh
2:22:25
primary Upstream owners uh common issues uh critical dat and then slas and
2:22:31
agreements right so uh let's put that in here so we can say um
2:22:38
slas this is bigger then uh in this case
2:22:47
uh this is the data should land um 4 hours after a UTC midnight or
2:22:55
something like that right so obviously these run books you're probably like Zach this is like the most ugly run book
2:23:01
you I've ever seen you write in your entire career and you're kind of right about that but uh the idea here right is
2:23:08
now if someone goes to this Pipeline and then they're like H like I see this like refer anomaly and then it's like boom
2:23:16
done right but then it's like okay and then you want to question like okay if that's the case maybe we need to remove
2:23:22
that check and just and make it so it's like a higher um signal to noise ratio where like the actual alerts are that
2:23:28
way but it's it's kind of hard to do that over the long run like of just actually uh like fixing all your data
2:23:34
quality checks and like some of them are going to be noisy sometimes but that's kind of the idea right and this is
2:23:41
obviously the the I I expect a lot more out of y'all because you're going to like hopefully create a kind of a
2:23:47

pipeline spec that is um if if you're in the combined group you're lucky because you already created a spec that has a
2:23:54
pipeline but like if not if you're like in the infrastructure group then you need to also create just like a pipeline
2:24:01
that describes things with inputs and outputs and then uh it would it can also describe potential common issues and
2:24:08
slas so that you this this is going to be your homework for uh this week and it's done in groups though so you don't
2:24:14
have to like it's it's a lot I'm I'm actually asking for quite a bit especially if you're uh if you didn't do
2:24:19
the uh Analytics homework in week three then uh this is going to be uh a significant amount of work congrats on
2:24:25
getting to the end of the day two lecture if you like this Channel and like this boot camp make sure to like
2:24:31
comment and subscribe and share this content with your friends I'm so happy that you're taking this time to invest
2:24:36
in your knowledge and getting better at data see you at the next one [Music]