Real-Time Pipelines with Flink and Kafka

Day 1 Lecture

Transcript:

0:00
real-time data processing is one of the hottest things in data engineering right now in this course we're going to be
0:05
covering Apache Flink we're going to be covering Apache Kafka we're going to be covering how to use these things to
0:10
process data in real time you're going to be able to connect to an actual Apache Kafka cluster in the cloud that
0:16
is attached to the data expert Academy you'll be able to process click stream data in the labs and you'll be learning
0:23
a lot about the complex windowing functions that are available in Flint such as count windows lighting windows
0:31
and session windows that should be a really fun time as well and uh I really hope you enjoy this many hour course if
0:38
you want to learn more about how to do this in the cloud check out the data expert Academy in the link in the description below I can get you 20%
0:45
[Music] off well what is a streaming pipeline
0:53
this is a great question there's a lot of words that we're going to talk about today uh there's streaming There's real
0:59
time there 's near real time and then there's micro batch we're going to talk about each one of those and get very

Lecture 1

1:06
clear on the definition of these words because as a data engineer you need to
1:11
understand the differences between these whereas your stakeholders are not your stakeholders are not going to know the difference like if you if your if your
1:18
stakeholder says we need the data in real time that might not mean use Flink that might mean something completely

1:24
different but for the most part what we want to say here is uh streaming pipelines process data in a low latency
1:31
way and in this case what I mean by low latency is usually that means that the data is processed on in the order of
1:39
minutes uh uh depending on like your windowing function or something like that maybe the order of a couple minutes
1:45
to half an hour something like that would be kind of your latency for like when the data is generated to when it is
1:51
processed so that's the low latency way of thinking about it like um another
1:56
word that I like to use here is intraday because a lot of pipelines in the
2:02
business world are uh daily pipelines like they actually are um once a day they they
2:10
kick off at UTC midnight which is like 5:00 pm and then they run and then they maybe run for a couple hours they run
2:17
from like 5: to 7 and then that's it and then they're done and so this is different where intraday means that it's
2:24
going to run more than once a day and that's where this gets complicated and
2:29
we're going to talk about all the different ways that things can run more than once a day so yeah let's dig into
2:35
it okay we have a couple words here right we have streaming we have near
2:41
real time and real time and also micro batch which is connected microbat and
2:48
and near real time are pretty much exact synonyms so like let's go over this a
2:54
little bit so streaming which is Flink which is what we're going to do in the lab today is
3:00
streaming is is also called continuous processing which you can think of as
3:05
okay every time uh an event is generated we process it immediately and so if and
3:13
it's processed like a a river you can think of it as like a river of literally like a stream or a river like the data

3:19
just keeps flowing and flowing and flowing and it never stops that is a great way to think about it um the the
3:26
best example of a streaming continuous process engine is Apache Flink which is
3:32
what we're going to be using in the lab today uh then you have near real time near real time is a little bit different
3:38
where instead of processing data as it is generated you wait maybe a couple
3:44
minutes or a minute or two minutes to collect all the data that has happened in that period of time and then you
3:50
process that data as it comes in and uh the big poster child here is going to be spark structured streaming so spark
3:57
structured streaming is not yet a Contin processing framework even though data bricks recently announced that they will
4:05
be committing to making spark structured streaming uh continuous uh continuous
4:10
processing framework and at that time when when data bricks does announce that
4:16
it's going to be uh Flink and Spark streaming are going to be essentially
4:23
the same at that point they're going to kind of converge just like a lot of other Technologies kind of converge these two technologies are going to
4:29
converge a a little bit as well so that's a good thing to remember uh so
4:35
what that means though is that real time and streaming are sometimes synonyms but
4:40
they're often not synonyms and you want to make sure that
4:46
you understand the latency requirements of the pipeline and not just take your
4:52
stakeholders uh words where if your stakeholder like oh I need it in real time that doesn't mean hey I need to go
4:58
and bust out a Hatchy Flink and get real technical like I mean it almost never
5:03
means that actually I think in my career uh when a stakeholder has said real time when they actually meant we need a

5:10
streaming pipeline that has happened two times two times in my nine-year data
5:15
engineering career when like people have said they need real time at least two dozen times so it's like maybe 10% of
5:22
the time 10% of the time when a stakeholder says they need real-time data they they that means they need a
5:28
streaming pipeline and so remember that that like streaming pipelines the actual
5:34
use cases for them are rare they're like one in 10 of like actual pipelines at
5:39
least as a kind of a a general rule that you can use when navigating relationships with stakeholders and
5:46
we're going to deter like we're going to talk more about like what what constitutes like a valid use case and
5:52
what doesn't so yeah don't worry about that yeah let's let's dig a little bit deeper as we kind of like learn more
5:57
about the different verbes here so so another word I said here that's not on the slide is micro batch so you can see
6:03
like uh near real time data is processed in small batches small batch micro batch
6:10
near real time there are like all the same that's all the same like very very
6:15
like those are like exactly the same whereas uh microbatch and real time are
6:20
not exactly the same they like they're similar but they're different so that's a key thing to remember as you kind of
6:26
like go through these processes so what does Real Time mean from stakeholders right it it rarely means
6:34
streaming very rarely um it usually means low latency or predictable refresh
6:41
rate so I've I've even had stakeholders that say we need this data in real time and what they meant by that is they
6:47
needed the data up to date like they needed a daily pipeline up to date by 9:00 a.m. every day and it the data

6:54
actually only needed to refresh once a day so it's funny because as a data engineer stakeholder says I need real
7:00
time but what they actually need is just a batch pipeline they need just a a regular old plane Jane batch pipeline
7:07
which hopefully is like a you know an efficient low latency batch pipeline that is regularly refreshed and is
7:12
reliable and that's a lot of times what stakeholders kind of mistake here is
7:18
they think uh like what real time means is like reliable and predictable and
7:25
it's it's like different words and that's where one of the very good things
7:30
that you can talk about with stakeholders is you can have this thing called the SLA discussion which is the
7:36
service level agreement discussion which essentially means like you ask them okay
7:42
like when do you want the data refreshed you can ask them that question and a lot of times they'll be like oh we need it
7:49
eight hours after midnight or 10 hours after midnight or whatever and then you can make an agreement of like okay our
7:55
data we can guarantee that the data will be available by then and that
8:00
conversation will be way better than like just uh not having that conversation and then being like okay
8:05
we're just going to do everything in streaming and we're going to go the whole other way and just like not talk with people that's why like data
8:13
engineering is also in some ways more communication than software engineering
8:19
is because you really need to talk with people about like what their actual latency latency requirements are because
8:26
otherwise like you're going to over complicate things and don't do that don't don't don't build them a Ferrari
8:32

when a bicycle will work right don't do it don't do it I promise like the tech debt and the pain and the maintenance
8:39
that you will uh end up creating for yourself is not worth it so yeah let's uh let's talk a little bit
8:46
more there's a perspective of should you use streaming is it like something that
8:52
uh is worth it is uh like so these these questions here I think are going to be
8:59
great things that you can ask yourself so I'd say one of the number one things you need to ask
9:06
is do the members of my team have a streaming skill
9:11
set I want to give an anecdote here about this one specifically so when I
9:17
was working at Netflix most of our pipelines uh in the security space were
9:22
done in batch right batch or micro batch and there was a big Push by the the
9:29
people in security to be like yo we need lower latency even though we were giving them like data every 15 minutes right we
9:36
were like that's good enough right we can catch the bad guy if like we might catch him 15 minutes late but like what
9:43
kind of damage can he do in 15 minutes so I don't know from our perspective as a data engineering team we were like
9:50
well like do we actually start to implement streaming and change the
9:55
perspective of it because we have our entire team is very good batch pipelines with a patchy spark and we know those in
10:02
and out we are like the Rockstar team of batch pipelines so like why would we
10:08
move to something that's brand new and we're going to have to have the team learn it piece by piece and anyways what
10:16
happened was the security team was like we have to have it lower latency there's no way around it and so uh one of the
10:24
members of my team he uh started migrating some of the pipelines from bat
10:29
to streaming and one of the things that happened was he became an island where

10:35
when the streaming jobs broke like he was pretty much the only one who could fix it and then I kind of like started
10:40
to learn about how to fix it as well that was like where I picked up some Flink skills but then again it was just
10:46
me and him and we were on a team of 15 and I was always like this is unfair why are we the only ones who have to
10:51
troubleshoot the streaming pipelines I don't get it and so this is why when
10:58
you're working on a team you should understand like there should not you like you don't want to be in the worst case scenario where like you build a
11:04
streaming Pipeline and you're the only one who can troubleshoot it because you're the only one with the skill set because that's terrible it's a terrible
11:10
horrible no good very bad place to be uh hopefully not too as well because being
11:16
on call 26 weeks a year is also not fun um so hopefully you have at
11:23
least uh two like two other people so you have like three people on the team who understand streaming Pipelines if
11:29
you uh that's still going to be a little bit uh taxing but not as it should still be pretty decent so then you also have
11:36
to think about like what is the incremental incremental benefit so one of the big things that can happen here
11:41
is okay we lower latency like in in this in this Netflix example where we have 15
11:46
minutes and maybe we can lower it to like two minutes or three minutes and it's like okay we can stop the bad guy
11:53
in his tracks a little bit faster we can give him only you know we give him only uh like a couple minutes instead of like
11:58
50 15 minutes that's the main benefit that we would see and uh how are we going to do that and so that could be a
12:05
win obviously like and and in that case in the security space like you can actually make a case for it whereas like

12:12
in other cases you can't make the case where it's like okay it's available like the data is available at 3:00 a.m.
12:18
refreshed versus like okay we could have had it available at 1:00 a.m. refreshed and it's like is anyone going to even be
12:26
querying that data at that time anyways does like like you going to have some an an analyst this is who's doing an
12:32
emergency data analytics project at 2 in the morning like probably not so that's
12:38
another important case to remember incremental benefit like what is the actual value that we get from reducing
12:44
the latency important um okay homogeneity of your pipelines uh
12:52
I use this word in another presentation like there's a couple words that I really like in data engineering and
12:58
obviously yall know my my favorite one which is item potent and that's my favorite word I love that word a lot but
13:04
this is another one that I really like is homogeneity so if you're uh a data engineering team and you have 99% batch
13:13
pipelines like okay like why add a streaming pipeline unless there's a the only time that would be the case is like
13:19
if you have people on the team who have the skills and there's a a strong incremental benefit to it then maybe
13:25
maybe that's the case but generally speaking if you're are if you're a batch team you should stay with batch and I
13:30
also agree on the flip side like if you are a streaming team you should stick with streaming and that's what Uber does
13:37
right Uber runs with this thing called Kappa architecture which is a streaming first architecture so for them like it's
13:44
almost the opposite where they're like okay we have all these streaming pipelines why would we add a batch pipeline that's just going to complicate
13:49

things and uh red and we're going to have more of a uh you know a heterogenous uh mix mix of pipelines
13:56
instead of a homogeneous mix of pipelines and so that can just increase the maintenance burden and the on call
14:02
Overhead like when you are considering whether or not to use streaming and then uh I think this is
14:09
probably one of the most important points is okay what is the tradeoff between these four different options
14:16
right so you have daily batch hourly batch micro batch and streaming those are going to be your four options of
14:23
essentially how you can process this data and a lot of times it like I'm going to show on I think the next slide
14:29
there's a Continuum here of kind of complexity versus latency and the
14:35
tradeoffs there where again you want to go back to what is the incremental benefit and maybe batch a daily batch is
14:42
going to be fine and then another important thing to consider is okay where are we going to put data quality
14:48
because data quality is a lot harder to do in streaming it's a lot harder to do in streaming than it is in batch because
14:54
in batch you have like these obvious steps right you have a then B than C
15:00
right and then so you can essentially at any between a and b or between C and D
15:05
you can put a cut and you can be like okay I'm going to insert quality here and stop the pipeline if there is a
15:12
problem but with streaming it's like it just runs all the time man it just runs
15:17
forever there's not a then B then C it's just always on so uh data quality can be
15:24
a harder thing to do and that uh for streaming so that's another kind of trade-off it goes in the trade-off
15:29
bucket and that's where batch daily batch is going to have the the easier kind of data quality path so hopefully

15:38
those are some things that you can consider when uh a state the next time a stakeholder asks you hey we need real-
15:43
time data these are uh things that you should consider before using Apache
15:49
Flink okay so
15:54
um streaming only use cases so these are some cases where uh there isn't there is
16:00
a massive trade-off that you can't make uh with like say you use hourly batch or
16:06
micro batch there's just you can't use those as your uh options because it makes or breaks the use case so the like
16:13
you have to have extremely low latency for there to be any value at all um uh
16:19
the number one example I always point to is detecting fraud so like if someone steals my credit card I want to have
16:26
them not be able to go on a one day spending spree before the fraud is
16:32
detected the next day I I think that that's kind of a terrible fraud
16:37
detection system I don't think that that's what they should be doing and uh you should probably be able to to detect
16:43
fraud in a little bit more of a real time kind of way because that will uh
16:48
actually make the the product a lot better so um obviously there's other options
16:56
here like high frequency trading where it's like a imine uh you're like oh man oh man like I I found the perfect setup
17:03
for a trade yesterday right because the data is on daily batch and it's like okay well like we good for you bro you
17:10
can't that setup is gone like you can't use it anymore it's like too little too late so you have to be able to process
17:16
things in a low latency way so you can actually make the trades that you're looking to make another good good
17:22

example is like sports analytics so if you're like watching a real time game and like or watching a football game and
17:28
they're like doing all sorts of I if you ever seen them like draw the circles and like give them give the stats in the
17:33
game a lot of that's need needs to come from uh real time pipelines because again if it's like oh that play happened
17:40
even if it's like a near real time and it's micro match it's like oh here are the stats for the play that happened
17:45
five minutes ago and people are going to be like what like this is terrible boo so
17:54
like the key thing that I want you to remember here is there's like for the streaming only use cases there is an
18:01
obvious signal that we should be using streaming because if we don't use
18:07
streaming it's terrible it breaks the product it makes it an unusable
18:13
product here's a good example of where things get a little bit blurry so I
18:19
would say there's essentially two areas that are kind of gray area use cases for when streaming could be used microbatch
18:27
could be used Etc Etc so if you have a master data that is Upstream in your in
18:35
your data warehouse that is that is like one of the furthest Upstream data sets that is then depended on by a lot of
18:41
other data sets then streaming could be very powerful because it could make it so that all the downstream data sets can
18:48
fire sooner because the the daily data will be ready sooner and that can be a
18:54
massive massive benefit to the pipeline and when things are available and can cause your whole because one of the
19:00
other big things that that does is it allows your your Warehouse to be kind of amortized so you can use all of the
19:07
compute throughout the day as opposed to having some dead compute in the middle of the day and then at midnight there's

19:13
a big spike which happens a lot by the way in a lot of different warehouses like
19:18
midnight Midnight UTC to like 2 a.m. UTC is like a spike and then uh it kind of
19:25
drops down and then there then as people come in to the day and they come in
19:31
around 5:00 p.m. UTC 5:00 p.m is when they kind of like come in and then they
19:36
will and then you get more of those ad hoc queries and it kind of like you get more load as like you have the data
19:41
analysts and data scientists running their queries that are more ad hoc and so if you can actually amortize your
19:48
compute throughout the day it makes better use of your Cloud resources especially like if you aren't like using
19:54
like on demand resources and you have like you like renting you're renting out your ec2 instances and you're paying the
20:00
same no matter what then you want to maximize the compute for each one of those ec2 instances and that can be a
20:07
very powerful way to do it and anyways Master data latency important so I want
20:14
to talk about a use case that happened for me uh where I failed at implementing streaming because it was freaking really
20:20
hard dude it was like so hard and so I was working in notifications and I
20:27
needed to work on this uh notifications event fact data so this data set was
20:32
essentially every row was a notification event you can think of like a notification sent delivered generated
20:40
you click on it all those different events in the event stream but the thing is is like they can be duplicated
20:45
because you can click on a notification twice so one of the things that my boss was saying he's like Zack this is a
20:51
perfect perfect use case for streaming and then I'm like okay and then I like
20:56

pump it into streaming and and no matter what I did it just o out of memory out
21:02
of memory out of memory like even if I maxed out the memory setting still out of memory and the reason for that is because like U the duplicate could
21:08
happen at any point throughout the day so even if I'm you know it's it's it's morning and then I uh I click on it in
21:15
the morning and then I click on it at 11 o'clock at night those duplicates still need to be managed and so that means
21:20
that you have to hold on to every notification ID for the entire day which means that you have to hold on to
21:26
everything in RAM for like the whole day which means that and that notification data set was like 20 terabytes so it's
21:33
like okay we just need good old 20 terabytes of RAM and we can just uh go ahead and do this right and obviously
21:40
that wasn't in the cards that was just like a lot of ram because if you think about it from the perspective of spark
21:45
right the what one executive needs 16 16 gigs of RAM that's the most that you can
21:51
give it so imagine 40 terabytes of ram it's just it's you can't do it so instead like for me I in that case I
21:58
tried streaming didn't work and instead I moved everything to micro batch and that worked great uh I think I shared
22:05
that repo with y'all about how you can kind of do microbatch in like a tree fashion to do dding and it was great and
22:10
that made it so that instead of Landing uh nine hours and so it used to land at 9:00 a.m. UTC and now and then after I
22:19
migrated it it landed at 1:00 a.m. UTC which allowed all the downstream pipelines to pick up and run a lot
22:25
sooner and that gave all like made all the notification data sets way more up
22:31
to dat and uh analysts love me for it because then they didn't have to like be like oh wow we have to wait for
22:37

notifications to update because that nine hours after um midnight meant that
22:42
like some days it was going to be it's going to bleed into the afternoon of the next day so my whole point here is
22:50
microbatch can work streaming can work in these cases and the last point I have
22:55
on the slide is like if you're serving data to customers it should probably be pretty up to date because customers have a high expectation on latency they
23:03
expect your data that you're serving them to be as up to date as possible because if you serve them like data from a week ago they're
23:09
like what like why are you doing this so you can get some good things with that I
23:14
I found that like when you're serving data to customers one of the things that you want to be careful about is you have
23:20
that tradeoff right where it's harder to implement data quality so a lot of times like streaming still might not be a good
23:26
fit and microbatch still could be the better fit because microbatch is easier to implement like whole data set data
23:33
quality and and even for customers if it's an hour behind depending on the use
23:39
case as long as it's not one of those streaming only use cases I was talking about in the last slide they might be
23:45
okay with it being an hour behind and uh so that can be good just many many days behind is when customers are going to
23:51
get upset so there are some gray areas here where streaming could be used and streaming might not be used uh I
23:58
generally found most of these gray areas are going to be solved with microbatch but I have seen some of these gray areas
24:05
be implemented with streaming as well a big one here is if the analyst is
24:14
just upset that the data isn't up to date and uh and it's those cases where he's
24:21
querying it early in the morning or something like that like just let it be let it be like doesn't matter like uh

24:27
like because what are they do with it anyways because that's the big thing to remember about like if an analyst is complaining that the data is not up to
24:33
date like you need to ask yourself okay if it was up to date what would change in the business and most of the time
24:41
nothing and so it's probably a waste of your time to reduce the latency and so
24:46
in those cases because you think about like the the velocity that business decisions are made at most business
24:53
decisions aren't made within hours they're not like oh man like like we're looking at this data and we need to
24:58
immediately act we need to immediately solve this problem that never happens like business does not move that quickly
25:05
like ever and so that like that's a big thing to remember when your analytics partners are like complaining about
25:11
latency is that like so what man so what if it's data from yesterday like so what
25:16
like it can be data from yesterday and that's fine it's most of the time it's fine that's just something that you find annoying and not actually impacting the
25:23
business that much and that you get the same results anyways even if you're making decisions based on data that's
25:28
one day late um so you got to think about it in those cases it's like okay
25:33
even if the data was immediately available and they could watch it update
25:38
in real time how would that impact their decisions I mean in some cases I I got I
25:46
got some news for you there can actually potentially even be uh a net uh you know
25:54
negative to having real- time data the most upto-date data because like then people like might be just stuck on the
26:00
screen and mesmerized by the fact that the numbers keep updating and they're like wow this is so cool like sometimes
26:07

like we don't want we as humans don't need to know that we don't need to know how the data is coming in like and we
26:13
can just have data once a day have you ever seen those people who are like addicted to the stock market when they're like looking on on Robin Hood or
26:18
something like that and they're watching the stock change the value change and they're like wow this is so cool but then they get so myopic on like the
26:25
minute to- minute changes of the data that they wait a bunch of time and it's like it doesn't matter it's just a
26:31
minute to- minute fluctuation and maybe just looking at like the daily open and close of the stock market is probably
26:36
going to give you maybe even you might even make better decisions based off of that because the the minute to- minute
26:41
fluctuations are just noise and it causes you to kind of overly focus on noise over uh signal and so if that's
26:49
happening like like you want to be careful be be real care real careful when you're
26:54
working with that kind of stuff so key thing here analytics Partners business
26:59
moves slowly most of the time if they're complaining that the data is not up to date just tell them what would be
27:08
different if it was up to date and the answer most of time is nothing so and
27:13
it's beautiful that's like one of those things that should be very relieving as a data engineer is that like if B like
27:18
if business was moving quicker right and then they were like yeah we got to solve our problems faster and we have to have
27:23
everything in real time I think being a data engineer would be even more stressful than it already is and so
27:29
that's a good thing so impatient analytic stakeholders can definitely be annoying but that doesn't mean you
27:35
should you should use Flink let's talk a little bit about how streaming and batch pipelines are
27:41

different because they're very very different they're like almost the opposite so a streaming pipeline runs
27:48
all the time it runs every second of every day all the time batch pipelines run for a small percentage of the day
27:55
they usually run even the beefiest of Pipelines will run like 4 hours four out
28:00
of 24 so one sixth of the day and that's going to be those are those Bera big
28:06
giant Behemoth whale pipelines a lot of Y y'all pipelines if you aren't in big Tech they run for like five
28:13
minutes so like five minutes out of a day is like not even 1% whereas your
28:19
streaming pipeline will run all the time so one of the things that's kind of bad about that though is did y'all know that
28:26
uh um Men actually pay more more uh for insurance on their car than women do and
28:34
one of the reasons for that one of the biggest reason for that is men drive cars more and it's like if you drive
28:41
enough miles in your car you will crash your car guaranteed 100% guaranteed if
28:46
you drive like you're statistically guaranteed to crash your car if you drive enough even if you're the safest
28:52
driver in the world because you're going to get ran into because there's just all that Randomness that can happen so my
28:58
whole point there is you have a streaming pipeline that's running all the time the probability that it breaks
29:03
is higher than u a batch pipeline that's running for five minutes a day because the rest of the time the batch
29:08
pipeline's just vibing and it's just like sitting on the couch just waiting waiting to to run for the next day so in
29:14
that way like strey pipelines require a completely different skill set they require because they are they they act
29:22
streaming pipelines act like a web server almost they act like a a server that is like serving data moving data

29:29
and is acting like it's very software engineering very software engineering and I think that's one of those things
29:35
that uh is why another reason why streaming isn't as implemented because
29:41
it's not better it's not better in some regards as well if you have something that's running all the time it's not better it's definitely not better so um
29:49
it's that's where you can get a lot more failures and like it's different so you also for
29:54
streaming uh it's kind of like sparking this way but you need unit tests and integration tests for your streaming
30:00
jobs even more than for your spark jobs because for two reasons one is that it
30:06
acts more like a server it runs all the time so you're going to need to have like you need to have that software
30:11
engineering mindset when working with streaming pipelines but also it's harder to implement data quality like in the
30:18
data quality checks which the batch pipelines and Spark they get that they
30:23
get the batch pip they get the the quality checks on the batch pipelines so and so you get that second fail safe
30:30
with batch pipeline so it's like even if you're like wow even if I push a bad build or I push a an incorrect query and
30:37
Spark at least like uh the data quality checks and Great Expectations is going to save my ass but like you don't get
30:44
that luxury in streaming so you need to have more unit test and integration tested streaming otherwise you're going to have a you're going to have a hard
30:50
time so remember that like that streaming is it's a lot like there's a lot of things that go on it's a lot more
30:57
complex oh yeah here's the here's that Continuum I was talking about in the earlier slide
31:02
so you essentially have a a Continuum here from not complex to very complex
31:10

and also from high latency to low latency so you see on the left side here we have SIMPLE High latency daily batch
31:20
and then on the right side we have like Flink low latency and high engineering
31:25
complexity so remember that like like when you're working with latency stuff that there's a Continuum here where the
31:31
lower the latency the solution the higher the complexity of the engineering it's not like Flink and Spark are equal
31:38
in terms of complexity because they're not and so like as you reduce the latency of your solution the complexity
31:46
goes up so that's a good good thing to remember as like you are thinking about oh should I use batch should I use
31:52
streaming is send people this send people this screenshot
31:58
real time is actually a myth because even even with Flink even with Flink
32:05
it's a myth because of the fact that there is Network time so a lot of times
32:11
how it works like for example for the lab today I have this thing called an Interceptor that what it does is every
32:17
time my website gets a web request it intercepts it and logs it to Kafka and
32:22
then the average latency for that step right is about 150 m seconds and then
32:29
from there Kafka has it and then Flink needs to pick it up from there and then Flink picking it up from Kafka the the
32:35
latency there is anything from like uh 100 milliseconds to like a couple seconds and then Flink then needs to
32:42
also write that out to a sink and that really depends because Flink can use
32:47
windowing and if there's windowing involved the latency could be multiple minutes but if it's just like uh an
32:53
enrichment where there's you're just uh like WR reading and writing rows and maybe adding columns which is what we're
33:00

doing in the first lab here by the way that um if it's an enrichment then we need to uh like the the sink can be
33:08
written out too almost immediately but then there's going to be the latency on like postgress right or a latency on uh
33:15
riding out to another kofka que so what I'm trying to say here is that this step
33:20
from event generation the cough going to Flink to sink like I remember when uh um
33:26
I was working at Netflix and we were trying to detect security anomalies and
33:31
uh they have this thing called red team testing at Netflix which is where they have people try to hack Netflix and then
33:38
they inject bad data or messy data into the environments and try to hack stuff
33:44
and uh they essentially did a red team test and we had all of our streaming pipelines ready to go and for the red
33:51
team test to go from like hack compromise to kofka to to Flink to sink
33:59
to uh alert that process which should be you know our promise was hey real time so
34:08
when a compromise happens we will immediately know about it that wasn't the case it wasn't the case at all it
34:14
still took a couple minutes so it's like I don't know like obviously there's going to be time in between just because
34:20
you have all the other technical steps that can happen in between when something actually happened and when something is processed and discovered
34:26
and generated so there's going to be all those different pieces right and so we
34:31
already talked about the four categories here of daily batch hourly batch micro batch and continuous processing so I
34:37
don't need to really talk about that as much but like the whole idea here is real time isn't it's not like immediate real time
34:46
A lot of times people think of there's a synonym of real time means instantaneous and no cuz technical things take time
34:54

and like if we could get there I mean I think smaller scale you can actually get things to be close like you can get
35:01
things to happen and you can get a alerting and all that stuff on the order of um you know a single digit seconds
35:09
but as you scale up and everything things can take a little bit longer because it's like you have to sift through stuff and like there's more data
35:15
to look at and like there there's more resource contention and there's like all these other things that can happen that make things a little bit slower that
35:22
make the the real time part of it take a little bit longer so there's a Continuum
35:27
here that's the whole point here and uh just remember real time does not mean
35:34
instantly talk about the structure talk about the structure of streaming pipelines um there's a couple things
35:41
here that we want to talk about um so you the sources uh the poster child of
35:46
streaming uh sources is Kafka kofka is um been around for a long time like I
35:52
even used Kafka like way back in the day like back when like Obama was President I used Kafka back in like 2014 and so
35:59
kofka has been around for a long time same with rabbit so kofka and rabbit are like the two big competitors uh Rabbit
36:05
doesn't have as much throughput so like it it doesn't scale quite as well as
36:10
kofka does uh but rabbit has more complex routing mechanisms so like for
36:16
example rabbit can be like a message broker where you can do like a pub sub a
36:22
lot easier with rabbit than you can with kfka whereas kofka is more like a fire hose in One Direction and it's like more
36:28
of like a big old river that like you can't really like divert it's just like
36:33
here's my stream of events now process them and but in because of that architecture it's really fast and can
36:39

handle a lot of data so you have the sources you can have another source called uh um a dimensional Source like
36:46
maybe what you're trying to do here is you have your events coming in and you want to have that event bring in some
36:53
dimensional sources like maybe you want to do some of that like denormalization of your fact data where you want to
37:00
bring in some different columns from uh like your STD table or whatever type of
37:06
table you want to bring in and these are called side inputs so if you want to learn more about this like I would
37:12
Google Flink side inputs and that's going to be a way to look at how you can
37:17
kind of marry these two worlds so you can have a side input of data that's coming from uh like a a regular data at
37:25
rest Source whether whether that be like Iceberg or postgress or any Source
37:31
that's considered like a table like a data table instead of a data Q or Flink and kofka and rabbit are like cues and
37:39
and they're another word for that is like data in motion so you can have dimensional sources that enrich your
37:47
event data that can be very powerful as well and then these things will Refresh on a Cadence so like what can happen is
37:54
you can set up your Flink job that will have like a dimensional data that that enriches the events and then that
38:00
dimensional data will Refresh on a Cadence maybe every 3 hours or 6 hours you get a pick how frequently that
38:07
dimensional data refreshes so that you get the right dimension for that event and obviously like if your dimensions
38:13
are changing a lot then that Cadence needs to refresh very often but you don't want to have it refresh too much
38:19
because then like you're going to like waste all this query and compute for data that didn't even really change but

38:26
those are the two big Source inputs for your streaming pipeline uh and that's where your data comes
38:34
from um these are you can think of this as like the spark of a streaming Pipeline and it's how we actually make
38:41
sense of our pipelines and like what's going on with them uh your two options here are going to be mostly Flink and
38:47
Spark streaming there's other options out there as well but these are the two state-ofthe-art ones and um these are
38:53
the ones that actually do like the sequel or the crunching of the data and
38:59
you can learn a lot about like how this works with these things Flink is very powerful and like how it manages like
39:05
Windows and uh sessionization and watermarking and out of order events
39:10
there's a lot of things that Flink does well that we're going to talk about here in the next couple slides but keeping in mind so you have the source then you
39:16
have the compute then you have the destination which is where is the data going so you
39:26
process your data in real time like you your data lands in real time in
39:31
Kafka Flink picks it up in real time and then it dumps it somewhere and generally
39:37
speaking there's three choices you either dump to another C kfka topic or you dump it to something like the data
39:44
Lake in this case like iceberg is probably a good example I want to give a little aside about iceberg real quick so
39:51
iceberg is one of the reasons why it was created was to solve this stream
39:57
problem where before you had Hive The Hive metast store and if you wanted to add
40:04
data to The Hive metastore you had to overwrite the old partition so you had to like essentially take all of the data
40:10
and overwrite it again all the data that was already there so there was no way to like add new data to a partition that
40:17

already existed that just didn't work Hive didn't let you do that and Iceberg was like okay we we want to let people
40:23
do that I think that that's a very important thing to do is let people append new data to partitions as opposed
40:30
to having to overwrite because Hive metastore was very batch oriented it was a very big batch oriented uh meta store
40:39
whereas Iceberg was like we need to be more streaming friendly and so that's why Netflix moved to Iceberg is so that
40:45
they could have data tables that could be populated by streaming and then
40:52
easily queried by batch pipelines as well so you get both if you use Iceberg which is is so cool that's like like of
40:59
everything that's like the number one reason to switch to Iceberg is like you have streaming events coming in and you
41:05
want to append and then obviously you have postgress as another sync that can be another very powerful way to uh set
41:11
up um uh a sync let's talk about streaming
41:17
challenges this stuff is always really interesting so one of the things remember I was
41:23
talking about how there's latency between when uh data is generated and when it lands in
41:30
kofka so that is totally true and one of the
41:35
things that can happen when that when data is generated when it lands in Kafka is because there's latency you can
41:42
actually have data that was generated before other data land after it so you
41:47
have like older data uh that is or you're going to have like newer data
41:53
that is ahead of older data in the data set and so the data is not in the right order and that can cause a lot of
41:59
problems that's a a big issue for streaming pipelines that you want to worry about is how do you manage out of
42:06

order events especially like in cases of like if you're looking at a funnel analysis or you're looking at uh all
42:13
sorts of different uh event streams that like can have out of order events is something that I think would can can be
42:21
a challenge for streaming and we're going to talk more about like what what is done to kind of minimize the impact
42:28
of that then you also have late arriving data so late arriving data generally speaking in the batch world doesn't have
42:36
as big of an impact and the main reason it doesn't have as big of an impact is because of the fact that you only have
42:43
to worry about it at midnight like maybe like 2 minutes after midnight and like a
42:48
lot of times like the the batch job doesn't even fire until 10 minutes after
42:54
midnight or even 15 20 minutes after midnight and it's like okay then you have the entire set for
43:00
batch like you don't have to really worry about late arriving data unless it's like extremely extremely extremely late and in those cases like most of the
43:07
time people are like well that data's gone forever and we're just going to not process it and so uh you want to think
43:13
about late arriving data Flink has some interesting ways to manage that as well and another big one is recovering from
43:20
failures so one of the things that's tricky about streaming pipelines is it's a little bit different from batch
43:26
pipelines in that that when you uh when it fails you need to like reset it and
43:33
then have it run again like and the longer you wait to fix a streaming
43:38
pipeline failure the more and more data Gets behind it and it gets like backed
43:43
up more and more and more and more whereas B whereas batch pipelines that's not the case right you just have your one batch of data and that's it so
43:50
there's not as much pressure to necessarily recover from failures when you are in the batch World versus the

43:56
streaming world there are ways to fix this problem in some regards in the streaming
44:02
environment as well and we'll talk a little bit more about those as well but these are the three big things that you want to be more aware of when you're
44:08
working in the streaming environment where like most of these things like in the batch environment like they just
44:14
don't exist or they don't matter so like these are different problems that you won't need to be aware
44:20
of so let's talk about uh outof order events so Flink can deal with outof
44:25
order events so how it works is when you have an event stream in Flink you can
44:31
specify this thing called a watermark and a watermark says okay all of the
44:37
events after there's no events that are newer uh than than the watermark and and
44:43
essentially like once that Watermark is hit we can guarantee that all the events inside that Watermark are going to be
44:49
older than the events after the watermark and so how the watermark works is it looks at uh the event time of the
44:57
events coming in and usually there's some buffer so like maybe it's like a 15sec buffer so like you have an event
45:03
time and then you say okay everything within the next 15 seconds could be out of order but something 16 seconds away
45:11
is not out of order we can we're essentially guaranteeing that all of the the orderedness is going to happen in
45:18
the next 15 seconds and so that's how uh water marking works is you give yourself
45:24
a window for each event and then you have like the next end number of seconds
45:29
that is considered okay there's a possibility in this um uh in this window
45:35
that things could be out of order and then Flink will fix the ordering in that in that window automatically for you
45:42

which is cool that's one of the things I like about Flink it does some really cool things for you so that's watermarking so uh we're going to cover

45:48

water marking a little bit in today's lab as well it's one of the pieces of the puzzle here and hopefully uh y'all

45:54

will uh find that interesting okay let's talk about uh recovering from

46:00

failures so Flink manages there's so many different ways that you can manage

46:05

recovering from failures so uh the big one is going to be uh actually

46:10

checkpoints checkpoints is going to be the main way and you can tell Flink to checkpoint every um n number of seconds

46:18

and it essentially saves the state of the job at that moment in time and then it knows like where to read from and

46:25

where to write to if if things fail so that when you when you start it back up again it doesn't like read everything

46:31

again and that kind of leads it to a segue around what are called offsets so

46:37

kfka has this this notion of what's called an offset and essentially when

46:43

Flink starts up you have to tell it whether to do earliest offset latest

46:49

offset or a specific Moment In Time those are the three choices that you have or it can pick up from a checkpoint

46:56

or save point those are all those things that are kind of different ways that uh

47:02

this stuff can can work out so earliest offset means read in everything that's

47:07

in cofa read in all the way back as far back as we can go and so that's a lot of data latest offset means only read in

47:15

new data after the job starts so that it will only read in any new incoming data

47:22

and then you have a specific Tim stamp which is like okay only read offsets that are at this time stamp or newer and

47:29

then okay one of the things I want talk about here real quick is the difference between save points and checkpoints so

47:36
checkpoints are they're internal to Flink and they
47:42
work uh kind of in a way that Flink can manage things like kind of internally
47:48
and it's kind of the the data that it it recovers from is like this internal Flink binary whereas save points are
47:55
actually um more agnostic they're they are more like
48:01
you could think of it as like okay it's going to be more like a CSV file of like okay we processed this data and we got
48:07
this far because sometimes if Flink fails we want other systems to be aware of like hey this this failed and this is
48:14
where we got to and so so save points are more used for like other systems than for Flink whereas checkpoints are
48:20
going to be used for Flink itself so those are kind of like the main ways that Flink recovers from failures we're
48:26
going to talk in this case we're going to talk more about offsets and uh checkpoints in the lab today okay late
48:33
arriving data there's uh this is another thing that link can do uh which is where
48:39
okay how late is too late and you do have to pick a time there there's
48:44
actually a time that you have to pick around like okay uh is it freaking five minutes that's too late 10 minutes
48:51
that's too late like how far away is it that things need to be before um how
48:57
late is too late and keeping in mind that there are two different uh we're going to talk more about this on
49:03
Thursday but late arriving data and watermarking are related in some regards
49:09
right because you could think about water marking as being like okay there's 15 seconds and it can be out of order
49:15
and it's like in some regards out of order also is kind of like late arriving because it it it came in in the wrong

49:22
time because other data was able to sneak in before it so watermarking and late arriving data are similar Concepts
49:30
but watermarking is more for like the 99% of data and then the late arriving
49:37
data is for the very the long tail of like the small amount of data that might come in like exceptionally late whereas
49:44
watermarking is more for the the data that arrives out of order in the window so yeah late aring data is not really a
49:52
problem for batch because um the only time it's a problem for batch is around UTC midnight and that's only if your
49:58
jobs fire really quickly so uh yeah that's essentially it for late arriving
50:03
data congrats on getting to the end of the streaming data pipelines lecture day one if you're taking this class for