Real-Time Pipelines with Flink and Kafka

Day 2 Lab

Transcript:

2:02:13
we're almost done with this one guys so with Flink we got uh you saw in like the
2:02:19
job folder here we had this start job which we actually still want to work
2:02:26
with uh because of the fact that it dumps to Kafka right it because we have
2:02:32
this right to Kafka and then this right here will allow us to have a job that
2:02:38
will essentially we can have another job that reads from this right and that will
2:02:44
make it so that we can aggregate on other things too like we can maybe
2:02:49
aggregate along like country as well so we can kind of see uh we can build some
2:02:54
kind of tumbling Windows around that kind of stuff so the whole idea here is
2:02:59
we will end up using that name right what this process events Kafka that's
2:03:05
going to be the name that we're going to be working with but one of the things that we need to do though is we need to create another entry point because I
2:03:12
don't know if you'll yall remember like um in the uh oh wait no wrong one
2:03:21
so you know how in here we have the or in the make file we have uh make
2:03:27
job which is like this big old thing here we need to have like another line here and we're going to call this uh
2:03:34
we'll just call this like window job like window underscore job and then uh change this to uh we'll
2:03:42
just call it aggregation job because that's the name of the file and I don't want to rename the file so
2:03:50
aggregation job so I'm going to uh I'm G send this in the in the chat but
2:03:55
obviously this needed to be like actually in the in the file there we go put this in the chat here so y'all can
2:04:02

uh put that in the make file just so uh we can have that kind of up and running so obviously first thing I want to do
2:04:08
here is I want to say make up so we can um get all of our uh okay so there we go
2:04:14
we have a Flink is now running we have all that up and going and like let's go
2:04:19
look at this AG aggreg ation job real quick because I think that that's something that is going to be important
2:04:27
um okay so you saw how uh this name here needs to be different this uh create
2:04:33
processed events Source this needs to match the if we go back to start job is called process events kofka that's what
2:04:40
we're going to actually call it so want to change this name here to process events Kafka in the aggregation job
2:04:47
so one of the things that this allows us to do is Let's uh kind of go step by
2:04:55
step through this uh aggregation job and I think that will make sense as to like
2:05:01
what is going on here so uh we have log aggregation this is going to be our job you see at the bottom here we have the
2:05:07
main looking great so in this case we uh again set up our execution environment
2:05:14
we say we're in streaming mode and then we create our table environment this is very like standard Flink stuff that
2:05:21
you're going to see in like every single job then we go ahead and create our source table which is going to be that
2:05:27
process events Kafka topic which is actually dumped to from that other job
2:05:34
so what we want to do is on this screen we want to say make job and I don't know what's going to happen if all of us are
2:05:41
running these jobs but we're just going to go with it so uh we have this make job which is going to have those it's
2:05:47
going to have those two jobs running and what you want to do is you want to keep this going CU this is going to dump the
2:05:54

geocoded data into Kafka into that new Kafka queue and then what you want to do
2:06:00
is open a new tab right so
2:06:05
um right and one second why is it why is it so
2:06:11
funky see it's like it's like it didn't update like it's like okay whatever it's
2:06:17
like it's like sad and I don't got my coloring but okay so anyways in here right we could say make aggregation job
2:06:23
and it's probably not going to work but it might work and then we'll have our third job running right okay so it's
2:06:31
breaking right let's see what it's saying here a group window expects a Time attribute for grouping in a stream
2:06:38
environment o so let's uh let's see what's going on there so it's saying that like a Time attribute is the
2:06:44
problem here okay so let's go look at our things here right so one of the
2:06:49
things that happened here is you'll see uh we have our tumble over and then we have event time stamp and I think in our
2:06:58
table yeah see we have event Tim stamp here but it's a varar right and that's
2:07:04
uh that's like a problem because that's not what we want we so what we need to do is we need to add another column here
2:07:11
uh which we can do it's pretty easy uh like it's actually in the other job we need to do that so what we want to do is
2:07:19
um where is it there I have a perfect example here yeah this one so we want to get this uh
2:07:27
pattern here because I think we have the because we end up writing it out and uh
2:07:33
oh yeah no we want this pattern this is going to be the pattern that we use so but that's fine we can copy this one and just get rid of the Z and the T so we go
2:07:40
back to the aggregation job okay we have this pattern here but we actually don't have the T and we don't have the Z
2:07:47
because this is the pattern that we actually write out I think that that matches that matches the this um format

2:07:56
right oh no there's just no SSS though you don't wa we we have just the lower we just have the little s's not the
2:08:02
uppercase s's okay so I think those well what I'm going to do I'm just going to
2:08:07
copy this it's like make it be exactly the same okay now we know for sure it's exactly the same and so what we want to
2:08:14
do here is we want to make a new field here we're going to call this maybe like window Tim stamp and then uh want to do
2:08:20
it the same way that do it in this other job so was like two time stamp right two
2:08:26
time stamp this one want to grab this say as that so window timestamp as
2:08:34
to timestamp and then this is not event time this is event time stamp because like it's the same as this one and then
2:08:40
that will use the pattern that we shifted it to when we wrote it out in the Pro in the other in the other job
2:08:46
that is currently running so um that looks right that looks good to me um so
2:08:53
one of the things here is that should fix our problem for the rest of this code so one of the big differences here
2:09:00
though is now you'll see down here um where we're defining the window see okay
2:09:07
so we have our source which is great and then we have our aggregated table and our aggregated sync which are working
2:09:13
fine but then what we what we have here you see this tumble we have this tumble over so this is going to be our our our
2:09:19
we're going to have a window that is opened every 5 minutes and this is now like this is where we want to update
2:09:25
this to window timestamp instead of event timestamp and that should give us
2:09:31
um what we're looking for as um kind of the idea
2:09:37
of of like the so this is going to give us our window and this is where like you
2:09:43

see we actually get group by here and so this is where we're going to get a little bit of a key right so we're going
2:09:48
to have windows that are based on both both host and based on like time so we
2:09:55
have like five minute chunks so what I want to do here is I want to say make up because we need to uh
2:10:01
essentially uh start them up and oh wait that did not okay I think we actually
2:10:06
have to kill the other job real quick because uh we need to do make down because we need to get the the new the
2:10:13
new updates into the into the aggregation job right so make down make up so we can get
2:10:21
the actual job running here so there we go now that's running and I want to go back to the other tab here and say make
2:10:27
job because that should give us the um the actual data here there's probably
2:10:32
more steps that we're going to be missing here to get this aggregation actually up and running but so now that's working that that's the other job
2:10:39
but then we can say make aggregation job over here and so this is hopefully going to
2:10:45
work now okay a group by window
2:10:51
the stream environment isn't it giving it's the same air why why is it mad now is there like
2:10:59
another ah it's because it's here you got to do it in both because this is it's the same problem here right because
2:11:05
we have another job here that has the same problem so we need to change this to window time stamp and like obviously
2:11:10
like if you it's like if Flink is not syntactically correct it's going to make that mistake right and we'll go we'll go
2:11:17
over both of these windows here in just one second I just want to get this job up and running so we got to do the same
2:11:22
like make down make up like Madness right to get the um the new the new code
2:11:30
there so uh then we can do make job see oh oh it just kills it if you do make

2:11:35
down cool great and then now we now here we can say make aggregation
2:11:44
job oh is it going to work it might work how is it still the the same bug
2:11:53
because now now this is like now this is right because now we have our window
2:11:58
time stamp and that seems right and we have like everything here that seems to
2:12:05
be what we're looking for because then we have event hour window Tim stamp and
2:12:10
then this is we shifted it up here right and we have our two timestamp pattern
2:12:15
here that seems right to me though because then we have our group which is our table name which we are doing right
2:12:22
right here so okay why is it upset um like I want to
2:12:31
look back at this so we say validation error a group window expects a Time attribute for grouping in a stream
2:12:39
environment does does it have the actual line oh because this is all in Java
2:12:44
right and it's like no we can't do it okay so that means it's actually fail it's it's getting here and then it's not
2:12:50
um uh working the way you would expect though so let's um let's kind of go
2:12:55
through this a little bit and see like what it could possibly be here because we have the window Tim stamp here about
2:13:01
this aggregated table there's something in here that is okay that's time stamp that's fine and that's event hour which
2:13:07
should be fine and then um interesting interesting that's so weird because I I
2:13:14
definitely had this working before so we have the window timestamp which is going
2:13:20
to be right I mean unless it's like not processing it right if it's not this actual like window Tim stamp here like
2:13:28
if this is not like if this is not doing the right thing for some reason if this
2:13:33
is like parsing it wrong then that could be what's going on and that like it's

2:13:38
not actually uh processing that the right way but that seems right to me because
2:13:46
because that's how we are dumping it out over here right because I think that that's the problem is that the date
2:13:52
format here is not giving us the time stamp that we're expecting it to give us and that's like why it's like yelling at
2:13:58
us right now so anyways um I want to I want to okay there is uh one of the
2:14:04
things I want to try here is if we change this to latest just change this to latest real quick and try it with
2:14:11
that because that will like essentially have it show up as um uh like we'll only
2:14:18
get the latest data and maybe there's like weird data in kka that isn't parsing and that's that's the reason and
2:14:25
so now we'll only get the the latest data that might see if this if this barfs as well I'll be surprised a a a
2:14:32
group window expects a Time attribute for grouping in a stream environment and
2:14:37
then this is this has got to be that line it's got to be down here it's got to be like literally right here that's
2:14:43
that's the only line that even would Mak sense we have uh I want to go over the code and I know that there's just like a
2:14:49
small like timestamp like casting problem here that I like I will figure
2:14:55
out but like here's kind of the idea of how tumbling works right so you define a window which in this case we're going to
2:15:01
do every five minutes which but this can be every five minutes based on processing time or based on the event
2:15:10
time right and in this case we're trying to do the event time which usually is what you want to process on you don't
2:15:15
want to process on processing time um and then what we're going to do is we're going to group those windows together in
2:15:22

this case we're going to group on the window and the host in this case we have our host here and then uh from there we
2:15:30
have our uh event hour host numb hits all that stuff and then uh this is going
2:15:36
to essentially do an aggregation every five minutes or hopefully that would be that ideally that's what's going to
2:15:42
happen here so um what I want to try I want to just try one thing real quick I want to try using the other um uh like
2:15:51
the other pattern here because my I still I still wholeheartedly believe like what is going on here is this
2:15:57
pattern here is just not like it's maybe we want to try two two more tries here
2:16:02
on using this pattern and then also the pattern with the with the seconds so we'll try the second ones first and then
2:16:08
we'll try the one with the Z and everything in it after that and then if both of those don't work then I have
2:16:13
failed y'all and like we won't be able to like there's going to be like some other crazy thing going on here but this
2:16:20
should uh with the seconds is it gonna is it going to give it to us with the with the with the
2:16:25
Millies okay with the Millies it still has the same problem but like what about if we change it to the this the exact
2:16:32
same uh one that we had uh the exact same pattern that we have
2:16:38
from um the other
2:16:44
job okay so actually no because it's failing without even reading in data so it's saying it's it's not like this time
2:16:51
stamp is not right it's like oh it's saying expects a Time attribute for a
2:16:57
job right and so I wonder if uh it's like it's not
2:17:04
quite it's not like time this is like a time like maybe there's a difference between time and time yeah Kyle do you
2:17:10

have any thoughts on this curious like oo dude that dude if that's what it
2:17:18
is like oh man that that that actually freaking probably is what it is dude
2:17:24
let's just throw in the watermark real quick uh we'll call this uh we're going to for we're GNA call this window
2:17:30
timestamp and then as we're GNA say window Tim stamp dude that's dude that's dude
2:17:37
that's totally what it is that's totally what it is and then this is going to like my this is going to just be the
2:17:43
that right it's going to be that regular one that matches the other pattern you're totally right 100% they're that's
2:17:50
totally what it is is it's like totally forgot the watermark dude if this just works now
2:17:58
like it probably will just work and then then then I'll feel better I oh oh it's
2:18:04
not failing it's it's definitely getting F Yeah Boy let's go thank you Kyle you
2:18:09
saved my life right there this this that was this was almost a demo fail okay so um let's go look at like what is going
2:18:17
on in uh in Flink here because we're going to have a lot more jobs running
2:18:22
right so um we also oh yeah we need to run the other job right so we need to do
2:18:27
like make job over here this might actually not let us do all four
2:18:32
um so I will uh I'm going to send y'all that Watermark definition here in just one second too I just want to see if
2:18:40
there we go so now I think the problem here is that we don't we don't have slots for four yeah we only have slots
2:18:45
for three but okay but but so one of the things I want to walk through here real
2:18:51
quick in like these window jobs is that they are more complicated so you'll see in this case we actually end up having
2:18:58
two there's like two records here right you have the um or there's like two
2:19:03

steps right because you have the um like you have the the first step of like reading in the data and then you have
2:19:10
the group window aggregate step which ends up so this is where you get this is where watermarks are useful this is
2:19:17
where like uh because watermarks like if you are getting data and you're just reading and writing data as like rows
2:19:23
and you're not doing any grouping then the ordering doesn't matter that much right but now you'll see we have a
2:19:28
watermark and Kyle's totally right in that like we needed to have that Watermark right that was exactly why we
2:19:35
couldn't window because you can't window without a watermark in Flink so what we need to do is we need to go and create
2:19:41
that table real quick so we can say create table processed events aggregated
2:19:46
and in this case uh this table has three columns I think there's uh oh there's
2:19:52
two tables here yeah there's processed events aggregated and processed events aggregated source so these are two
2:19:57
separate tables uh but like you can essentially just create them from the ddls and Flink uh I will pass this to
2:20:03
y'all as well so that y'all can have this but so this is going to be our first one where we have our processed
2:20:10
events aggregated and then we also have one for Source because we wanted to also look at um who is winning based on like
2:20:19
not just uh like like who's winning from Twitter who's winning from which place
2:20:25
right so so I'm say create table this is processed events aggregated Source this
2:20:34
has uh these three columns in it I want to run this guy so now these tables
2:20:39
should be already processing though like like once you make them like it should just immediately uh be uh pumping out
2:20:47
data right there we go so now you'll see we have uh essentially data coming in um

2:20:53
every uh we have data for like every five minutes right essentially for all
2:20:59
of these you'll see that a lot of this data is for like exactly and um Tech
2:21:05
Creator which makes sense but we should be able to say where host equals we can say Zack wilson. Tec
2:21:11
creator. because this should give us other data there we go because now you'll see this this see this 1855 this
2:21:18
is going to be the one that really is the the one voting so then you'll see we have three three hits there so really we
2:21:25
want to put an in here so we want to say Zach Wilson we say lulu. Tec
2:21:30
creator. and we want um Dutch engineer so Dutch engineer is uh Sarah
2:21:37
so have dunch Dutch engineer. Tech creator. and so then we have R three
2:21:42
here and then we should be able to run this and there should be more data than
2:21:48
that there we go so you'll see here we have our data coming in and oh this is like for the ninth
2:21:55
though so uh you'll see in our in Docker compose here in this Docker compose at
2:22:01
the very bottom you'll see this task manager number of task slots I'm going
2:22:06
to bump this up to five and then I'm going to just uh we're going to do make down and then we'll essentially make up
2:22:13
and then uh uh make it so that we don't have any jobs that
2:22:19
are uh like failing because they don't have enough task slots say make aggregation job but
2:22:27
like that's a great thing and we're about to find out if we need to increase parallelism or not and so uh this should
2:22:34
work okay so if we go back to running here we should have four yeah we're good
2:22:40
we're good because like like you would you should see some like purple here but you don't you see how there's no purple
2:22:46

like it's all running now it's all looking pretty good and so now I mean we should we should be seeing way more than
2:22:52
this though like this is trying to like say that like freaking nobody has like nobody has clicked on my links like I
2:22:59
don't believe that because like that doesn't make any
2:23:04
sense to me at all like there's uh Because unless uh the data in it's still
2:23:10
like chugging through all the data which okay so if we go to the what processed events aggregated Source if we go into
2:23:17
here this should be like pumping data right okay it is so one of the things
2:23:23
you'll see here is like it actually does give out like data sets here right so you see it says like okay we've sent 500
2:23:30
records and we've received 200 records oh I bet it's because of the like the five minute marks like these like five
2:23:37
minute marks is probably the thing because we have that window and it's like needing to open and close these windows but a lot of those windows
2:23:43
should be done like a lot of those windows like like like that's it still seems very strange to me that there's no
2:23:49
what if we look at the source table T processed aggregated events
2:23:54
Source okay this is looking again though again we like we don't okay see there there's someone
2:24:01
there is someone who's coming from Twitter like that's the t.co is doing right there like but uh this is still
2:24:08
like seems wrong though this still seems like this is off by like a day like the the time is not like because it's not
2:24:15
the ninth right it's actually the 10th so um seems really weird like that might
2:24:22
be like a latest thing but like I swear we are processing on each thing here we're only processing latest if we go to
2:24:28
the aggregation job I put in in here right there's just we use
2:24:33

latest offset and then in the start job we okay we're doing earliest offset in
2:24:38
the start job so that might mean that it's still pumping out new data into
2:24:43
Kafka there should be way more records here than than eight there should be like because it's like this is like for
2:24:49
every 5 minutes like there should be like way more than just like that's so
2:24:55
so strange to me that like uh because it's weird that it's like processing
2:25:00
data but like it only has like this many records right and it's like and this
2:25:07
must just be like spare traffic that like it's not actually okay what about the other job though because this other
2:25:12
job should be um what this process events kofka this job oh ah there it is
2:25:19
that's why this guy is struggling that's why this guy is definitely struggling this guy is
2:25:25
not writing data right now that's what's going on you're I think you're totally right Kyle that like this guy is like
2:25:30
he's um like he has like he's kind of backed up right he's kind of backed up I wish I could like so this is like I
2:25:37
think this is a great example of where like so what we should probably do here and this like this is an easy fix I feel
2:25:44
like which will uh make it so we can at least get some new data from people who are coming in is so I'm going to stop
2:25:51
the the start job and then um what I'm going to do is let's go back into that
2:25:56
job and change this earliest offset to latest offset so that like we aren't
2:26:01
like processing like everything right we just process the new data and that
2:26:07
should that should make a pretty big difference oh wait I got to do the got to do the whole thing I wish that like
2:26:14
it would just put it up there and I don't have to like Docker all this all the time but this should make it so that
2:26:22

we have more of a like actual kind of data set that can pull in more like it
2:26:29
will only pull in like new data as it comes in and then and then then we have to like have that like kind of those
2:26:34
five minute sort of interval things but at least then we'll have data that's like it's not just like struggling but
2:26:40
you're you're totally right that's exactly what was going on is that like it was uh not uh like it was totally not
2:26:49
like it was struggling to read all the data because of the fact that I don't think it because I'm so popular I think
2:26:55
a part of it is because like a lot of y'all have been running the jobs right and so uh that kofka queue actually has
2:27:01
a lot of duplicate data in it I want to see real quick that's actually like a uh that's a good um little thing that we
2:27:07
can add to this lab because now okay all my jobs are up and running now right we got okay they're all up and running and
2:27:13
so now this should have more upto-date data right or it's just going to not
2:27:19
have oh no it's cuz it hasn't been 5 minutes it hasn't been 5 minutes we have to wait till
2:27:24
7:40 and then uh then we will see CU that that's when that went the the the the tumbling window will close but I'm
2:27:31
like in the meantime we can go see how like messed up kfka is like and see how much data y'all like dumped into kfka
2:27:38
because I bet it's like I bet it's gnarly okay so here we are um here's kof
2:27:45
oh look at that what are we whoa this is um not what I'm looking I want to look
2:27:51
at process events cof okay it's saying that there's just
2:27:56
like 25 mags in there not like obviously something happened here with this IO sex
2:28:01
right and I think it might be because I have just like a really like tiny kfka cluster and that's what's going on here
2:28:06

and that's why it can't like handle handle it right um because because Boot
2:28:13
Camp events I'm guessing is probably going to be fine oh yeah but you can see that there is a step up here from like
2:28:18
uh when I made that post so there is more data in the queue for sure for sure for sure I'm just trying to get it like
2:28:24
to have so we have like Flink into Flink that's like the idea that's what I wanted to like really show in this class
2:28:30
there we go there we go okay we're good we're good fam it's there we hit this we
2:28:36
hit the five minute Mark so now we have uh that uh the 7:35 window closed and
2:28:41
you'll see uh Sarah is beating me by one and then Lulu has 16 hits uh and so
2:28:48
that's kind of the idea and you'll see if we put in like Source here then uh
2:28:54
the F minute window here closed as well and then you can kind of get the break down and you'll see like like Twitter
2:29:01
hates me I literally have no votes from Twitter none not even one all my votes
2:29:07
came from LinkedIn that's so sad uh Lulu's got some love some Twitter love
2:29:12
and then someone someone uh someone went directly there too you see there's like a null so like uh so TW Lulu is like
2:29:20
uh like mostly LinkedIn 12 from LinkedIn three from Twitter and then one direct so in this kind of uh competition that
2:29:27
we're doing right so this is working now this is working and um I'm glad it's working and we kind of discovered like
2:29:33
what's going on and why this is happening that's where this like latest offset versus earliest offset does make
2:29:39
a pretty big difference especially like when you're processing things because like you might end up like trying to
2:29:44
read in all the data sequentially and like if you're doing windowing on that that just becomes like even more Nar
2:29:50

right so um that's essentially what's going to happen here and we're going to build out
2:29:56
some more like stuff here well one of the things I wanted to go back over here with uh oh wait not that one with in in
2:30:05
the aggregation job so let's let's go back into the code now now that we actually see it working and I'll show
2:30:12
you essentially like what is actually happening here right so we are reading
2:30:18
in our source table we're doing a 5 minute window we're grouping on the window and the host and then we're
2:30:25
essentially uh this is giving us our time and then we have the host and then the number of hits that's what is in
2:30:32
here when we look at when we get rid of source here if we look at this table
2:30:38
you'll see uh that's exactly what's going on here we have the time and the number of hits and obviously this does
2:30:43
have that little bit of latency but you'll see like in in 2 minutes we're going to get more data and we'll have
2:30:49
like more data for every 5 minute interval and that's like what's really powerful about this is that it just like
2:30:54
automatically shows up it's pretty cool um and so uh and then the only
2:31:00
difference between that one and this next one is we also group on referrer which is going to be like did they come
2:31:06
from Twitter did they come from LinkedIn like where are they coming from and so it does this aggregation kind of in real
2:31:12
time which is pretty cool like uh do we need to do this in real time that's a
2:31:17
great question because sometimes aggregations are things that we don't maybe necessarily need to do in real
2:31:22
time uh so this is kind of how tumble windows work and like we're going to be
2:31:29
able to see as these windows close like how they kind of process over time which
2:31:34

I think will be pretty cool to see one of the other things that we could do here that like would probably you saw
2:31:41
how that choked right that the the last job kind of choked uh because of the fact that it um I one of the things that
2:31:49
we can do and uh I want to try this actually I think this is something that I think would be pretty cool to see just
2:31:55
to see like if it actually does the right thing I think that uh this will be a nice little experiment so what I what
2:32:02
I want youall to try out is like maybe change the the the offset back to
2:32:08
earliest offset for the aggregation job so what I'm going to do is I'm going to kill the aggregation job real quick and
2:32:15
then what I want to do is uh I want to you'll see how we have does m without set
2:32:21
parallelism so let's set that to three so that like maybe we can have because
2:32:27
uh that's how many uh if you saw in uh in this process events Kafka you'll
2:32:33
see that we actually have three partitions of data so we know like uh that the data is split up in three
2:32:39
buckets already and so that that will be something that will help us get more uh
2:32:45
like parallelism and we should be able to Crunch this data a lot faster if we have three tasks versus one so then what
2:32:51
I want to do is I want to say make down and then uh make
2:33:01
up and then here I want to say make job and here I want to say make
2:33:07
aggregation job and then I also want to show youall what that looks like differently in the Flink UI because this
2:33:13
is essentially how you set the parallelism so that Flink knows how many tasks to kind of spin up I I I'm
2:33:20
wondering if we might need to have more task slots but I don't think so because I think uh in this case the task and the
2:33:25
parallelism are separate I think it's like workers but um that I want to be

2:33:31
more sure about okay so we did okay so you see here how um this one is uh we're
2:33:37
running out of uh see how we get the these are scheduled tasks right so
2:33:42
you'll see in here how we now have parallelism of three but this is three
2:33:47
on each uh on each task here so the number of task
2:33:53
slots here we probably need to bump this up quite a bit because there's actually going to be uh so because here you see
2:33:59
how this changed to six that's because there's three t or there's there's the default parallelism is there's three
2:34:06
tasks per stage or and so we have six and six here so that's 12 that's 14 I
2:34:14
want to change this to 15 real quick I'm going to change the uh this guy to to 15
2:34:20
just so we have like enough spots and then let's try make down one more time
2:34:25
just to kind of show youall how like cuz you really do want to learn how to work
2:34:30
with uh parallelism and all this kind of stuff because this is these are the real
2:34:35
problems that happen in streaming right this happens all the time so like uh and this is the kind of things that will
2:34:41
like just kind of playing around with this stuff will hopefully this is going to give us what we're looking for so
2:34:52
bo bo boom and then this should be able to just crunch the data way faster like we can have like this like beast mode
2:34:57
kind of crunching of the data okay there we go now we're getting some more more jobs in here okay
2:35:03
initializing initializing there we go now now see now I got the six and sixes running and so
2:35:10
that's great that's going to uh that's going to make it so that these things are running way faster right we should
2:35:15
be able to see like way more data coming in through here and it will uh should just um kind of crunch all through it
2:35:22

and we should see a lot more data showing up in postgress as well so if
2:35:27
I okay so there okay so you see it did process the next one down here you see
2:35:34
the this is the the the 740 slot and um I did a little bit better there I was
2:35:40
able to to swing and get 25 in that in that case so um this this seems to be
2:35:46
working actually I think like I'm guessing there's actually like a lot of data here okay there we go see now see
2:35:53
now we were able to crank it up and get the parallelism up to like what you would expect and you see now it's just
2:35:58
like it's making quick work of that data like it's just like it was able instead of feeling clogged right it's just
2:36:05
murdering it now because it it's splitting it all up and it's reading each partition at once so we're getting
2:36:11
at least a 3X speed up here and so um that's what these parallelism that's what this parallelism is going to do
2:36:17
seems like this is working pretty well now um and so you definitely want to there's like this kind of trade-off
2:36:23
right between like parallelism and this stuff like one of the things that's nice about Flink versus spark in this way is
2:36:31
that uh the parallelism numbers are usually lower right because like you see how I moved it from one to three like in
2:36:36
spark you're always like thinking like oh should this be 200 or 100 or how many tasks or how should this split up right
2:36:43
and uh CU you know in in spark the default is 200 but in Flink the default is one because and and that's like one
2:36:50
of the fundamental differences between batch and streaming though right is because in streaming it's like okay we're going to process the data as it
2:36:56
comes in and that's going to give us what we're looking for and now this is going to essentially just work and uh it
2:37:02

will make super quick work of it but obviously one of the things here that is
2:37:08
can be uh kind of tricky is now like this job is going to be running all the time and there's going to be times when
2:37:16
there's not a spike in the data that and we we don't need need six we really don't need six uh tasks to process the
2:37:23
data we need one or two and that's where uh it's a tradeoff right where I I
2:37:29
bumped it up to six because I wanted to show y'all in this lab and I was like impatient right and I wanted to show y'all like how this actually is going to
2:37:36
work so um because you'll see in here you can see uh how the the votes are
2:37:42
coming in and oh you can oh we can actually even see one second let's see the The Source we can uh look at source
2:37:48
too okay so why is it like not all ordered okay there we go did Zach get
2:37:55
did I get any Twitter love I hope I got some Twitter love right oh like okay
2:38:00
good three people one of the things that I think is uh important here is that
2:38:07
there is so many just like jobs that you can run right and that like should I
2:38:12
think one of the other important questions here is like should these jobs be ran should this aggregation be ran in
2:38:19
Flink or should it be ran in spark or like should it be ran in batch should it be ran in streaming like how do you know
2:38:27
like because they both add latency right this adds 5 Seconds of lat or this adds five minutes of latency right and so
2:38:33
it's not like you're just uh like you you get this for free right in terms of latency so um one of the things to think
2:38:41
about there is okay like what kind of aggregation are you doing over what time frame so one of the reasons why this
2:38:48
aggregation is really nice to do in Flink is because it's a five minute aggregation and so and since it's a five
2:38:54

minute aggregation like doing that like imagine running a a running a spark job
2:39:00
that ran a group by every five minutes like that job's a joke because it's like
2:39:05
the time it takes to start up freaking spark to then run the group by is going
2:39:11
to take more than five minutes or it's going to take like 3 minutes just to start up and then it's going to run its query and then it's going to it's going
2:39:17
to essentially be running the whole time anyway except like you're going to have all this start up and tear down cost
2:39:22
where Flink is actually going to run more efficiently in those cases and so like there is a line where I think that
2:39:29
it it flips and I think that that's generally at about an hour so if you have a group buy that's like an hour or
2:39:36
longer then that's when batch kind of wins and kind of takes over like if you
2:39:42
have a if you're trying to group and DD over like longer periods than an hour
2:39:47
then uh you probably want to do this you don't want to do a tumble window that's an hour you do a tumble window that is I
2:39:55
don't know 30 minutes or something like that and then you do another aggregation in spark for the hour or the 5 hour or
2:40:01
what however big of a aggregation window you want to do like generally that that's kind of like what I've noticed
2:40:08
when I've been working with streaming that that's kind of there's a line there between like hourly and like sub hourly
2:40:15
where that where that's where it kind of transitions from like you should do that aggregation in streaming to you should
2:40:20
do the aggregation in batch uh obviously this only matters at like that rule of
2:40:27
thumb really only matters when you're working at very large scale uh like you can like Flink is totally possible like
2:40:35
at smaller scales you can have Flink do a daily aggregation and just hold on to all the data like this and kind of

2:40:41
aggregate it down because this is not that much data like it really isn't so like and since this is a 5 minute
2:40:47
interval it actually writes it out and it doesn't hold it on the memory anymore right it once it once it once the window
2:40:53
closes it writes it out and it go it persists to dis at that point so you don't have to like have tons and tons of
2:40:59
ram that's why batch is better for bigger Windows though because Flink just needs a weird amount of ram in order to
2:41:06
like hold on to so many windows for such so many windows that have so many like data points for such a long time whereas
2:41:15
uh like spark is going to be able to handle that much better so so um yeah
2:41:20
I'd say that that's kind of the perspective of like how to do these aggregations congrats on making it to the end of the streaming data pipelines
2:41:27
course that's one of the most complex things that you can do in data engineering so I'm proud of you
2:41:35
[Music]