EPISODE 450

[INTRODUCTION]

[0:00:00.9] JM: The New York Times is a newspaper that has evolved into a digital publication. Across it's 166 year history, the times has been known for long form journalistic quality. in addition to its ability to quickly churn out news content. Some content on the New York Times is old but timeless, evergreen content. Readers of the New York Times's website are not only looking for the most recent breaking news. They want to know what the headlines were the day after pearl harbor. They want to read editorials about Martin Luther king.

Over the last 30 years, New York Times has moved itself online, bringing old material with it. Since the 90's, several different content management systems, CMS's, have been used by journalists within the times to publish material. These different sources of content stored data in different formats. This is a data management problem. Users want to search over the entire history of articles published by the times. Which means that the times needs to unify these articles in a single index.

These are articles from the 1920's that were digitized using OCR, Optical character recognition. There are articles from 1998 that were written on a legacy CMS and there are articles from 2017 that used the latest CMS. Boerge Svingen is the director of engineering at New York Times and he wrote about this problem and its solution on Medium. This story describes the flexibility of Apache Kafka.

In contrast to the applications of Kafka as a place to buffer high volumes of data, the New York Times uses Kafka as a place to unify data and allow for other specific materialized views to be built on top of it. We have covered Kafka in the past with interviews of some of its creators including Jay Kreps and Neha Narkhede and you can find these old episodes by downloading the Software Engineering Daily app for iOS or for android. With these apps, we're building a new way to consume content about software engineering and they're open sourced at github.com/softwareengineeringdaily.

If you're looking for an open source project to get involved with, we would love to get your help. The special thing about these apps is that they have all of our old content. Not just a limited subset of the episodes like you will find in podcast players.

So, with that, let's get to this episodes.

[SPONSOR MESSAGE]

[0:02:35.4] JM: Auth0 makes authentication easy. As a developer, you love building things that are fun and authentication is not fun. Authentication is a pain. It can take hours to implement and even once you have authentication, you have to keep all of your authentication code up to date. Auth0 is the easiest and fastest way to implement real world authentication and authorization architectures into your apps and API's.

Allow your users to log in however you want. Regular username and password, Facebook, Twitter, enterprise, identity providers like AD and Office 365 or you can just let them log in without passwords using an email login like Slack or phone login like WhatsApp.

Getting started is easy, you just grab the Auth0 SDK for any platform that you need and you add a few lines of code to your project. Whether you're building a mobile up, a website or an API, they all need authentication. Signup for Auth0, that's the number zero and get a free plan or try the enterprise plan for 21 days at auth0.io/sedaily.

There's no credit card required and Auth0 is trusted by developers at Atlassian and Mozilla and the Wall Street Journal and many other companies who use authentication. Simplify your authentication today and try it out at auth.io/sedaily. Stop struggling with authentication and get back to building your core features with Auth0.

[INTERVIEW]

[0:04:25.2] JM: Boerge Svingen is the director of engineering at the New York Times.

Boerge, welcome to Software Engineering Daily.

[0:04:30.5] BF: Thank you.

[0:04:31.2] JM: Then New York Times is a newspaper that has evolved into a digital publication and today we're going to talk about some of the infrastructure that has contributed to that evolution. Give us some history on the different ways that stories have been published on the New York Times over the years as it has evolved into a digital publication?

[0:04:51.3] BF: That's a much bigger story than I can tell since obviously have not been there for most of that history but the New York Times is a little over 160 years old. For most of that period, we only published on paper. We started producing content in – producing it in a digital form. So printing it in the 60's I think and we now have the original digital versions starting in 1980. So everything before that ,or almost everything before that has been OCR'd.

[0:05:33.9] JM: Okay.

[0:05:35.2] BF: For the digital versions we have now.

[0:05:37.6] JM: It got digitized using optical character recognition where something scanned the text and it was turned into a digital version?

[0:05:46.0] BF: Yes, then starting in '96, we got the website. So for a long time, the website had content that had been created originally for paper and then adapted to the website and then starting, I think in 2006 we changed to a web-first approach where things were produced for the website first and then also would go to print.

[0:06:14.2] JM: Everybody knows the New York Times, it's a well-respected newspaper and you've got all this old content and people are still reading the old content under different circumstances, they're also reading the new content of course. How do the access patterns of the different ages of content, like the older content versus the newer content, how do those access patterns differ? Do you need to be able to readily serve people with the same latency of a request to an old document as you do to a new document?

[0:06:48.0] BF: Yes, it's served the same way. You can go to the website and get any article published since the beginning in 1861. So there's obviously a long tale here. A very big majority of what's being viewed is from the last couple of weeks. Then we have a good amount of traffic for the older stuff and some of it has a lot more traffic than others so for instance, we have a lot of important historical events like the Civil War and basically everything since that people go back and look at.

So for research reasons, historians use it and it's also a lot of entertainment value in waiting some of these old stuff. We have a twitter account called NYT Archives that publishes stuff from the archives every day, which can be a lot of fun and also a service called TimesMachine, you can use to go back and look at the old stuff.

[0:07:54.7] JM: The reason I want to talk to you on this episode is because you wrote about using Kafka at the New York Times to solve a very specific problem. I think this is a great description of a top down, problem statement and technical solution. I want to start getting into why Kafka was a meaningful technology as you were implementing the solution.

The solution is essentially how do you create a system where this content from all these different data sources in a huge span of historical record, how do you make sure that all of these are accessible for people who want to access the content? And all of this content is in different CMS databases, it's in the archives, it's on these other data sources and you'd like to get it all into at least a single interface.

Maybe it doesn't need to get all into one place, maybe it just needs to get into a place, in its desperate sources and you just put an API in front of it. We'll talk about those different approaches that you could take to presenting a unified interface over all these different content sources. But before we get there, I think I'll just spoil it and say that Kafka is part of the solution and I think we should run through Kafka a little bit so that people are familiar with it.

For listeners who do not remember our previous shows on Kafka or they haven't heard them, give an overview for what the technology does?

[0:09:29.4] BF: Kafka is in principle very simple. It's an attend only file on disk. Kafka producers will write to the end of that file, consumers can read from any point of the file so they can start at the beginning, they can start somewhere in the middle, or they can just consume new data as its coming in. What Kafka does is make this distributed so you can have replicas of your logs for redundancy and you can partition your log for scaling, reads and writes. It's basically just a log that you write data to and then read it back in the same order.

[0:10:14.2] JM: That term "log", you're not referring to a logging system where crashes and errors are happening and you're logging them. It's a log of records, right? Maybe you want to disambiguate that term?

[0:10:28.9] BF: Yes, you can use it for that case as well but yes, a log in this sense is just a totally ordered list of content where you have a file, you'll always appending to as you get it totally ordered record.

[0:10:46.6] JM: Kafka, it's a pub sub system, it's about producers and consumers. It's not just about reading from the entire file. You might have sections of this long file that are labeled as a particular channel and producers might publish to that channel, consumers might consume from that channel so that they can read a subset of the information.

What else would you like to say about producers and consumers in the top context of Kafka?

[0:11:21.3] BF: It's fairly straight forward. So like you were saying, a Kafka producer writes to the end of the file and the abstraction is, in Kafka for this, is a topic. A topic is basically a log, produces right to specific topics. Topics can also be partitioned, which means that you partition them based on some key. The default is just to use a basic hash of the key for each record, which means that you can distribute this on multiple files for scaling.

For a consumer — a Kafka consumer has an offset. Basically, everything written to the Kafka topic has an offset which is just the count from the beginning, it's a record count form the start of that partition. A consumer has a current offsets. The consumer can maintain that itself or it can let Kafka maintain it for you. The way this works is that when a consumer starts up, it will start

consuming from its current offset when it processes a bunch of records from Kafka, it will then commit a new offset, which is how far it's gotten so far.

That's why it's also arguable whether we should look at Kafka as a pub sub system because in most pub sub systems, you deal with individual messages and you're ACKing individual messages. With Kafka, you don't, the Kafka consumer will act a specific offset and basically saying, I've seen everything up until this point. There's no concept in Kafka of ACKing a message.

[0:13:05.0] JM: Okay. In the case of the New York Times, the use case that we're going to get into is that Kafka serves as the system of record for all of the published content. The producers of the content — you mentioned the producer term — are the CMS databases, the archives, the other data sources where people are publishing information.

How does this data from all these different data sources, how does it get into Kafka?

[0:13:36.3] BF: First of all, just to clarify what we mean by publish content in this case is that this is content that has been written and edited and is ready for consumption. So this does not apply to any content that's still being edited in the CMS. It's only when it's considered ready for the public that it actually goes into the system. The way this works is that we have a protobuf, ski moss, Google protocol buffers, which defines every – we have a schema for every type of content you can publish. We have a service called the Gateway, which validates everything you're trying to publish according to that schema.

So all the producers need to create that Protobuff binary, it will send that as a publish request to the gateway. If it validates, they will get a publish response back and what we call a publish event will be written to the actual Kafka log, which is then another Protobuff binary that wraps the actual asset with some additional meta data. As soon as it's there, it can then be consumed by anyone.

[0:14:51.5] JM: Once this data is in Kafka, it is again in a file that is sitting on disk. Is that correct? Does any of it get pulled into memory?

[0:15:01.7] BF: It's stored on disk but the way Kafka works is that it uses the OS page cache, which means that when multiple consumers are always consuming the latest offsets, most of them will be actually reading it from the page cache, which means that Kafka scales very well for a high number of consumers that are all reading the latest stuff.

[SPONSOR MESSAGE]

[0:15:33.3] JM: The octopus, a sea creature known for its intelligence and flexibility. Octopus Deploy, a friendly deployment automation tool for deploying applications like .NET apps, Java apps and more. Ask any developer and they'll tell you that it's never fun pushing code at 5 p.m. on a Friday and then crossing your fingers hoping for the best. We've all been there. We've all done that, and that's where Octopus Deploy comes into the picture.

Octopus Deploy is a friendly deployment automation tool taking over where your build or CI server ends. Use Octopus to promote releases on-prem or to the cloud. Octopus integrates with your existing build pipeline, TFS and VSTS, Bamboo, Team City and Jenkins. It integrates with AWS, Azure and on-prem environments. You can reliably and repeatedly deploy your .NET and Java apps and more. If you can package it, Octopus can deploy it.

It's quick and easy to install and you can just go to octopus.com to trial Octopus free for 45 days. That's octopus.com, O-C-T-O-P-U-S.com.

[INTERVIEW CONTINUED]

[0:17:04.5] JM: That's cool. I actually – Maybe this is a dumb question but the page cache that is an in-memory structure.

[0:17:11.0] BF: Yes.

[0:17:11.5] JM: Okay, got it. So there's consumers like you said that want to pull this data from Kafka and if it's getting accessed to buy a bunch of people, it gets into the page cache so the memory access is faster. What are the systems that want to pull data out of Kafka? Are there a

bunch of different types of systems or is it just, “I’m sitting down to read an article and I read it from Kafka, essentially?”

[0:17:38.6] BF: We have a range of different systems. First of all, all the systems doing this now are back end systems. For instance, Native apps do not read directly from Kafka and I don’t know if they ever will. We have a range of different systems. First of all, we have a system we call bodega which is basically a store of the latest version of every published assets.

Bodega just consumes Kafka continuously and makes every asset available for fast look up.

[0:18:14.7] JM: Asset meaning like images and stuff?

[0:18:16.8] BF: It can be an image, it can be an article, it can be any piece of content that goes on to Kafka. For images, we’re not storing the actual image binary in Kafka, it’s only metadata. When you’re reading an article on the webpage or in the native apps, it comes from Bodega.

[0:18:34.9] JM: I see.

[0:18:36.4] BF: Then we have a number of feed generators reading from Kafka to generate feeds. A lot of our content goes up to third parties.

[0:18:48.4] JM: Bodega is an example of a materialized view where you have – that is the serving view of content in images and all kinds of other things that when a front end client makes a request to the New York Times, the data that goes into an article that the person is eventually going to read on the internet is just all getting aggregated from Bodega?

[0:19:16.7] BF: Yes, that is correct. Another example of a consumer is our search clusters. We have an elastic search cluster, which handles both site search and a number of other use cases. We also have what we call a list services. Basically different services that consume the log and use that information to create lists of contents. For instance if you go to the science section and you see a list of all the latest science stories that’s created by a list service.

In that case, the actual members of the list come from the list service whereas the actual article comes from Bodega.

[0:20:06.3] JM: There is a couple types of things I'd like to go through. When you first stood up this Kafka solution, you had to pull in all of the old data into Kafka because Kafka was a new system that wasn't around when the civil war occurred. So you pull in all these old legacy articles and stuff into Kafka and then you've got also new articles that get published. So you've got some back end content management system and when a journalist is writing an article and they finish it and they click publish, it gets sent to Kafka and then you've got materialized views that need to update at some point.

You've got an elastic search cluster. You've got Bodega, which needs to serve the article whenever a user goes to Newyorktimes.com. You've got this list system that aggregates list of articles so if I want to go find all the articles about relationships between fat consumption and weight loss, then maybe I need to go to a list that is a materialized view.

Are there different SLA's on how these different materialized views are getting created? What is the system of keeping those materialized views up to date to a period where you want them to? Because obviously, if you were poling Kafka all the time to update the elastic search cluster, that might be a process that is compute resource intensive. So maybe you just want to have a regularly scheduled system that is looking at Kafka and saying, "Oh are there any changes? There have been changes, we should pull those into elastic search."

[0:22:01.8] BF: First of all, all these consumers do consume form Kafka continuously. Goes directly into an elastic search engine, all these other systems. The SLA's for this is different for different systems. For all the really important things like when an article is updated, our SLA is one second. Basically one second after the editor clicks publish, the latest version has to be available on the site, which means that it has to go through Kafka, it has to go be consumed form Kafka and into Bodega and then made available.

Same thing with things like the list service when something is updated it has to go out very fast. That influence is the kind of systems we can use for this different things. For instance, the main

list service we have is implement to using Postgres and the way it works is that everything on Kafka is published in a normalized way.

So you have articles referencing images for instance where the image is published separately. That is consumed into a similar normalized model in Postgres and then whenever something is published, we trigger on the types of lists that disk thing now became a member of. Which means that we can make lists available on a – with a very low latency. Whereas for something like elastic search, we need to have a denormalized model. So when you put something, an elastic search is basically one big JSON, which means that if we want tot be able to search on the caption of the image, that image has to go into the article object in elastic search.

That means that to create that denormalized bubble, there might be delays. For instance one of the types of assets we're publishing is tags that specify the concepts in an article. For instance, all articles about New York will be tagged with New York. If one of those tags are updated and that means we have to go into elastic search and update every documents that references that tag.

Which means that it's very hard. We can't maintain an SLA in one second in elastic search because of that normalized problem. Which means that we can only use elastic search for things like site search where that is less important because if a new articles doesn't show up for a search in a minute, then that's fine.

[0:24:55.8] JM: This term SLA that we've both mentioned, I want to point out for people who don't know is service level agreement and that's essentially, if you work at a big enough company, you have different people who area accessing different systems and these different systems will often times say, "We have an SLA, we have a service level agreement that we're going to get your response back to you in one second or five seconds or a millisecond." It's just a term used at big tech companies sometimes.

The approach of Kafka where you decided to centralize the system of record within Kafka. This is not the only solution that you could have taken. You were essentially faced with a problem where you've got lots of old data sources and I have seen solutions at other companies where

when they have legacy API's, that they want to update or they want to brand a degree of consistency between those legacy API's.

They standup another API in between those legacy API's and all of the consumers of those legacy API's so that they get a unified API that is papering over the past. You have this in some sense in the gateway because you have a gateway that sits in front of Kafka and validates that anybody who is publishing to Kafka is compliant with the Kafka schema that all of the other consumers are going to read from Kafka later on.

If you've got that gateway, why wouldn't you just make an API based approach and you know, not have to maintain this huge Kafka cluster? Then you could just have elastic search, wants to update, elastic search can just ping all of the legacy data sources and the new data sources through a unified API.

What's the problem with the approach of the API based unification?

[0:27:09.5] BF: This is similar to the system we did have earlier where we had a lot of different API's. Yes, we could have created a common API to sit in front of all of these things but there would be a number of problems with that.

First of all, those API's would be limited by the underlying technology in each producer of content. For instance if we want to be able to take the elastic search example and go and just get all the data and create a new index in elastic search. That would require all those data sources to support API's for somehow streaming data, which most data bases don't do very well. Our existing systems use common databases like MySQL and MongoDB and a few other things and there's not really any easy way to go to MySQL and say, "Give me everything and then keep giving me everything new," and have that perform well.

Basically, all the producers behind the API gateway would have to support that functionality. The API gateway wouldn't give you that for free. Also, in that setup, the producers would have to live forever. They'd basically have to maintain the data storage forever. Any CMS that ever produced content will learn how to live on so that they can still get requests through that API gateway.

Whereas when we put things in Kafka, that is the source of truth of published content, which means that all these, the old CMS systems that have archive data can then just be thrown away, we don't need them anymore. We don't need to maintain those API's. It simplifies things quite a lot for that. In general, the main thing we're looking for here in Kafka is the ability to do replay of data. So basically say, "Start at this point and just give me everything," which Kafka serves very well.

[0:29:31.6] JM: Why is Kafka better at doing that than for example a big MySQL database of all the information?

[0:29:42.0] BF: First of all, you can do this select star on MySQL and get everything out. That doesn't really perform very well. What you can do is take an actual dump and dump that to the file and export that to some other system and import it. The problem is that while you're doing this, things change in the database, which means that the snapshot you made immediately is outdated. Which means that as soon as you're done, you don't have to go back and do a smaller one for everything that changed. And you have to keep doing that and eventually you'll end up doing a lot of select just to get the latest things, which just doesn't perform well. Most databases are not made to solve that use case, they're made to serve specific queries.

[0:30:36.8] JM: Can you talk, contrast a little bit more between use cases where people might want to use a database versus when people might want to use Kafka? Because Kafka, this log-structured data store has become such a common piece of infrastructure in large companies. I mean, arguably, it's become not as common as a database but it's become extremely common but I think there's still a lot of people who probably don't know about it or what it does. Maybe you could just contrast the use cases. Generalize the use cases a little bit more.

When do you want Kafka, where do you want Kafka, relative to when you want a database?

[0:31:27.6] BF: I think the easiest way of approaching that is to talk about the concept of a log based architecture. It's basically a different way of architecting a system. Log based architectures have been championed by my Martin Kleppmann, who I think you also had on the show earlier.

[0:31:50.0] JM: He's coming back on soon.

[0:31:51.3] BF: Nice. The basic idea here is that in the database, you have a transaction log. Where basically every update is written to the transaction log and then the actual update is made into the database and in most databases that's just a way of insuring that every change actually happens even if the database crashes throughout an operation. What you do with a log based architecture is essentially that you extract that transaction log and make statures of source of truth, which is why this also asking reporter as an inverted database.

And what that means is that instead of storing the resulting changes from something that happens you're storing the events themselves in the logs. So you basically have one big ordered event log of events, things that happened and then you put your actual database as a consumer of that log and that gives you a lot of advantages over regular database. So you're still using a database because there is a lot of things you can't do with a log. Like you can't do a look up of a specific object from a log because you would have to seek the whole log to find it. You can't do any advanced querying. The only thing you can do with the log is just read it sequentially.

[SPONSOR MESSAGE]

[0:33:33.5] JM: You are programming a new service for your users or you are hacking on a side project. Whatever you're building you need to send email and for sending email developers use SendGrid. SendGrid is the API for email trusted by developers. Send transactional emails through the SendGrid API, build marketing campaigns with a beautiful interface for crafting the perfect email.

SendGrid is trusted by Uber, Airbnb and Spotify but anyone can start for free and send 40,000 emails in their first month. After the first month, you can send 100 emails per day for free. Just go to sendgrid.com/sedaily to get started. Your email is important, make sure it gets delivered properly with SendGrid, a leading email platform. Get started with 40,000 emails in your first month at sendgrid.com/sedaily.

[INTERVIEW CONTINUED]

[0:34:48.2] JM: So I think followers of Bitcoin and cryptocurrencies, this might sound familiar to them because in Bitcoin you've got a super long list of all the transactions that have ever occurred in Bitcoin and if you're a full node on the Bitcoin network, you pull off the entire list of transactions so that you have a statement of history and other people have that statement of record as well. So that you can have a shared view of the history of the world.

And Kafka if you think about the Bitcoin blockchain, that is a ton of data and then you look at, "Oh okay well maybe that makes sense for companies themselves," which I don't know how the size of the New York Time's Kafka cluster compares to the Bitcoin blockchain. Eventually I'm sure the Bitcoin blockchain will be longer. Well actually I don't even know about that, but in any case, you can maintain a distributed ledger of all of these records.

This is a doable engineering problem. It's not like it's so much data that you can't maintain a distributed record. So it's definitely very useful because if you just want to have this entire historical record of every event that has happened within the New York Times systems then you can pull all kinds of operational views off of that like we mentioned, those materialized views that fit different access patterns.

So do you want to talk a little more about the operational advantages of working with a log based architecture?

[0:36:43.5] BF: Yeah, so part of the problem with using a traditional database as your long term store is that once you have that database it is very hard to change. Because it is hard to make big changes to a database schema without downtime and it also becomes hard to replicate changes from a database. So like I mentioned earlier most databases don't support any easy way of just getting everything in the database to another system and what that means is that databases very easily become monoliths that you can't get rid of because they have all of your important data and it's too hard to get it out somewhere else.

Which ironically means that databases are actually not that good for storing data [inaudible]. Whereas if you use a log based architecture instead your database just becomes a derived store. It's a materialized view of what's on the log, which means that if you want to change the

schema in your database, you can create a new database with the different schema and just replay the entire log into that new database and then start switching your database consumers over to use the new database instead of the old one, which makes it much, much easier to change things.

It also means that you can have multiple databases instead of trying to have one database that should serve everyone. Every system can have it's own database, it's own materialized view that contains specific meta data that that system needs in the form they needed it.

[0:38:34.2] JM: The goal here was to create a log based architecture and Kafka is not the only product that you could use to get a log based architecture. I said product; I should have said open sourced project that can also be purchased as a product. There are these manage option like Google Pub/Sub and AWS Kinesis, how does Kafka compare to these?

[0:38:58.9] BF: So my argument is that those systems can't actually be used to make a log base architectures and I've had long discussions both with Google and Amazon on that topic. There are two things that are missing to do that because first of all, to make a log based architecture you need to be able to persist data forever because that is your record and your log can't go away, which also Kafka is very different from the way most people use Kafka because for a lot of use cases, it's okay to get rid of all data and our use case it's not.

And the other thing is that this data needs to be ordered because the assets we published refer to other assets and the ordering decides which version you reference it. So basically if we try to do this for this order that was not ordered, we would have to add all sorts of logic in every consumer to handle the fact that you can get things out of order because when you have an event based system you need to know that you are handling your events in the correct order. So I am actually not aware of any good alternatives to Kafka for this kind of systems.

[0:40:20.6] JM: Okay, well let's talk a little bit more about the New York Times use case. You've got every historical asset and article from the New York Times in the log, how is everything arranged in that big log?

[0:40:34.3] BF: So basically like I mentioned, we are publishing assets in a normalized model. What that means is it's very similar to a typical relational database in that we have different types of assets and those reference each other. So we have one asset type is what they call times tags, which are essentially concepts we use to tag content to tell us what this content is about and that's a separate at the type. We have things like images and video. We have different type of articles. We have slideshows that does a whole range of different types and all of these assets can reference each other in different ways and the way we're publishing this is both at the same time chronologically and topologically.

So basically every assets is published according to it's publication date, but it is also published in a way that any assets referenced by another asset comes before that asset on the log and more importantly, if an article references and image, when you consume that article, you know that the version of the image is the previous version of the image you saw on the log. There is no explicit version numbers of anything. It's all according to the order it goes on topic.

[0:42:14.5] JM: Everything in Kafka at the New York Times is on a single partition, explain what a partition is in Kafka.

[0:42:22.3] BF: Yes, so a partition is basically a separate log in a way. A separate file on the disk. So like I mentioned when you write something to Kafka it is an actual file on the disk, which assets are appended to and what that means is that you're limited at how fast you can published based on how quickly you can write to that single file and one way of scaling that is to use what Kafka calls partition.

So a partition is basically just another file that holds a partition of the data. So if you have a topic, you can say, "This topic should have 16 partitions and then there will be 16 files and you have some partitioning algorithm that decides for each assets with partition where it goes on."

[0:43:19.0] JM: What are the advantages of using partitions?

[0:43:22.1] BF: Yeah, the advantages of using partitions is that it lets you scale things up. So you can then consume — first of all, you can write more at a higher rate and you can also consume in parallel. So instead of having one consumer consuming the topic, you can divide it

out on multiple consumers that can each consume one partition each. So it's basically a way of scaling.

[0:43:46.8] JM: Right so if you have a huge high through put logging system like you were logging all the data from an IOT factory, you have a factory with all these sensors and stuff and you wanted to log all of these pieces of data and you wanted to get them all buffered up in Kafka then you wanted to pull them all into some centralized analytics tool then you could imagine wanting to partition the topic of that logging data so that the analytics cluster could parallelize the influx of the logging data into the analytics cluster. Would that be accurate?

[0:44:31.3] BF: Yes.

[0:44:32.0] JM: Okay, cool. So that brings up my question, the amount of data that is in the historical amount of what people have written on the New York Times because if this is like a content, a huge historical content buffer in Kafka, how does that compare to the type of example I just gave with the IOT factory? Does the New York Times have a lot of data or is it just not that much compared to some massive logging server?

[0:45:11.4] BF: This is very little data. So the order of magnitude is a 100 gigabytes and the reason for that is this is text written by humans and there's a limit to how fast our journalists can produce new content. So that's also why we can use just one partition for our case because we really don't have much data going into it but we do use partitioning for another use case. So the log we have talked about so far is what we call the monologue. That is the main source of truth for all the published content and it is a totally ordered single partition on Kafka topic.

But then we also have what we call the de-normalized log and that gets back to the normalized model because we have a lot of consumers that actually want the de-normalize view of the data where you basically can read off an article and get everything that is a part of that article and an elastic search is one example of that. That is how you want to put things into a lasting surgeon in a de-normalized way.

So what we do is that we have a component called the de-normalizer, which consumes the monologue, maintains its own local states of the latest version of everything is published and

then whenever something is updated, it is to write the whole asset in a de-normalized way to a de-normalized log. So that means that if an image is updated, every article referencing that image will be republished and that de-normalized log is partitioned. Because once we have the de-normalized log, we don't need the total ordering. We only need the ordering relative for each assets.

[0:47:15.7] JM: Explain in more detail, what is normalized and what is de-normalized. What do these two terms mean?

[0:47:22.4] BF: So normalized is the model you would have in the relational database. So you have an article table, you have an image table, and you have a key for the image and in your article table for every article referencing that image, you would just reference that key instead of copying the data and that's how things go on the monologue even if an image is used by multiple article it is published once and then referenced by the article.

Whereas on the de-normalized log, the article and the image and everything else referenced by that article is published together as one big chunk of data.

[0:48:07.4] JM: Explain how you use the Kafka streams API and maybe just explain what Kafka streams is.

[0:48:13.5] BF: Kafka streams is basically a Java library that makes it easier to write code that consumes Kafka. So Kafka streams will handle partitioning in local state and a lot of other things for you so you can basically bite a streaming application that leads from one source and write to some other source. So we use Kafka streams for the de-normalizer. So the same that leads the monologue and it creates the de-normalized view and writes that out to the de-normalized log.

That's a Kafka streams application and as opposed to other streaming services like Storm for instance, Kafka streams is just an API. Sorry, it is just a library using a new code.

[0:49:04.1] JM: What are the problems that its solving that you would otherwise have to hardcode yourself?

[0:49:09.5] BF: So there is a lot of small things. So for instance when you are reading from a partitioned log, you will have multiple instances of your application each leading from a subset of partitions and if one of those instances go down then the partitions handled by that instance would have to handled by someone else and all of that mechanic is handled automatically for you by Kafka streams. It will also help you with something like de-normalize the way you want to have a manic state.

So Kafka streams can maintain that state for you in a distributed way along multiple instances of your application and then it can do a lot more advanced stuff. So for instance, if you want to read two different Kafka topics and do join across those streams, that is something Kafka streams make very easy.

[0:50:14.2] JM: At this point we have given people really detailed view into how Kafka is used at the New York Times. Let's talk about how this fits into your bigger infrastructure given overview of how this Kafka pipeline was implemented in terms of engineering time and service providers and just the strategy of getting it deployed.

[0:50:43.3] BF: So first of all, this is a system that we have been building out a little bit over a year now. So we started working on this last summer and we are still rolling it out. So we are live in production now but we still have a lot of producers and consumers that are still using old API's that we still have to migrate over and we still have a lot of data sources that we also have to migrate over. All of this is deployed to Google Cloud, GCP.

So the New York Times has had a lot of stuff running on Amazon. We are now doing a lot of stuff on GCP. This is one of the first big projects we have done on GCP, so there has been a lot of learning involved in that because GCP is different from Amazon in a lot of interesting ways. The actual Kafka cluster on Google is running on compute instances. So the Google equivalent to ECT servers.

Everything else we have is running in Kubernetes. We've considered if we want to run Kafka on Kubernetes as well but we have decided not to keep that for now at least. It seems to be more complicated in just running compute instances and we don't see any obvious benefits from that

but that is an ongoing consideration and I know a lot of people do run KAFKA in Kubernetes so it's definitely possible.

[0:52:24.6] JM: What are the advantages that those people get out of it?

[0:52:26.8] BF: That's a good question. I think one argument from our perspective is that since we run everything else in Kubernetes, it would be convenient to have Kafka there as well just for operations purposes and being able to manage everything.

[0:52:44.2] JM: Did you consider giving the responsibilities of Kafka to a managed Kafka provider like a Confluent type of company?

[0:52:55.4] BF: Yes, so basically Confluent have now what they call Confluent Cloud, which to say manage Kafka. Unfortunately still only available on Amazon and we want to run it on Google. So if they decide to run this on Google then that's something we will definitely seriously consider because I don't particularly want to manage Kafka if we don't have to.

[0:53:27.5] JM: It's not exactly the New York Times competitive strength to be managing a Kafka cluster.

[0:53:33.4] BF: No it's not and we have had a strategy that we want to use manage services whenever we can and we do that for almost everything else. Kafka is an exception that is not.

[0:53:48.3] JM: Okay that brings up a good point because this is something I try to explore on the podcast a lot is build versus buy. Are there other areas of your infrastructure where you sometimes say to yourself, "This makes no sense that we're managing this and in five years we should absolutely be able to plug into a managed provider?

[0:54:08.4] BF: I think most of the things we are doing. So we come from a background where we use to have – well actually still have our own datacenters. That is something we are migrating away from and we have had services running on Amazon now for a good number of years and basically, we want to run everything in the cloud if we can and when we run things on the cloud, we want to use managed services when we can and also, server less is something

we're looking at quite a lot now to see how much we can to do with that because ideally, we don't want to do operations work. That is not what we want to focus on.

From the other perspective, most I think all the applications we actually run are developed in house. We have our own CMS system. We have most of the things are made in house and that is something we want to continue doing because that is a core competency. So the way we published information is very important to how quickly we can get new features out and present journalism in better ways. It is very important for us to actually have that ourselves because this is our code that we're writing.

[0:55:37.5] JM: What's the feedback loop there between engineers and journalist using that content management system?

[0:55:45.1] BF: That feedback loop is quite good. So part of the way that works for us is that — So the New York Times has The Newsroom which is basically all the journalists and the reporters and then we have the tech organization, which implements all of this. But we also have a smaller organization inside The Newsroom, which is kind of an intermediate between those of us developing the services and the journalists and the reporters using it and then that's a model that works quite well.

[0:56:21.6] JM: So –

[0:56:23.0] BF: I hear about it quite quickly when something breaks.

[0:56:26.4] JM: So I feel like the New York Times, like however you feel about the direction that our politics has gone, I feel like I have a renewed sense of respect for old media industries just because of the amount of questionable news sources that have found their way into the prominent public opinion and so we have started to really realize, "Oh we really need these old institutions that do fact checking that are very serious about the information that they publish."

Have you felt a enlivened sense of responsibility or a weight of responsibility in your own job since perhaps the 2016 election or just since the rise of questionable news sources on the Internet?

[0:57:25.0] BF: Yes, I mean, this is a big part of the reason why people want to work at The New York Times is because we're clearly doing good work here. This as an actual purpose for society. That's a good kind of job to have where it's so – it's not like a startup where you have to invent some mission for what you're doing. It's very obvious why we're doing what we're doing. That's very nice.

[0:57:49.0] JM: You wake up feeling inspired on regular basis I'm sure. You just say, "Okay, it's time to go and do the job of bringing a sense of truth to the people I guess."

[0:58:03.4] BF: That's not my first thought in the morning now, but I get your point.

[0:58:09.2] JM: Yeah, okay. All right, well just to close off, we've got a couple of more minutes but what are you working on right now, what are the big technical challenges at the New York Times?

[0:58:20.8] BF: From my perspective, the big focus now which will probably take most of next year is to migrate all the additional systems we have over to the new publishing pipeline. That's a lot of work, we have, our legacy publishing API has I think still about 50 different clients and all of them have to be migrated over to this new platform. Then we also have a lot of work on the content side.

What we're seeing is that when you have almost now 170 years of content and a large number of different systems and file formats, to take all that content and correctly put that into a single schema is not easy to do. Because there's a lot of complexities in how data has been used that's a lot of – there's a lot of inconsistencies between different systems where fields seem like there's a same thing but we're really not.

We have a lot of issues with our CMS system where the data in the data bases is not what we thought it would be because it's 10 years old, unless it's been written over a long period of time and some of the data was written where there was some bug that has long since been resolved but the data is still there. So there's still a lot of details to be resolved with all of us.

[1:00:01.2] JM: Well, I wish you the best of luck with that data cleaning and with the continued migration and I hope that the cloud service providers are able to eventually take a lot of the work off your hands.

[1:00:15.3] BF: Thank you, I hope so too.

[1:00:17.0] JM: Okay, Boerge. Thanks for coming on the show.

[1:00:20.1] BF: Thank you.

[END OF INTERVIEW]

[1:00:24.0] JM: Thanks to Symphono for sponsoring Software Engineering Daily. Symphono is a custom engineering shop where senior engineers tackle big tech challenges while learning from each other. Check it out at symphono.com/sedaily.

Thanks to Symphono for being a sponsor of Software Engineering Daily, for almost a year now. Your continued support allows us to deliver content to the listeners on a regular basis.

[END]