EPISODE 690

[INTRODUCTION]

[00:00:00] JM: Mapillary is a platform for gathering photos taken by smartphones and using that data to build a 3D model of the world. Mapillary's model of the world includes labeled objects, such as traffic signs, trees, humans and buildings. This 3D model can be explored much like you would explore Google Street View. The dataset that underlies Mapillary is crowd-sourced from volunteer users who are taking pictures from different vantage points. These smartphone photos from the crowdsourcing are uploaded to Mapillary. They're queued and they're processed to constantly update and refine the Mapillary 3D model of the world.

Mapillary processes high volumes of photos from around the world. These images in the photos need to be correctly fit into Mapillary's model of the world, like a puzzle piece sliding into place, and these are 2D images that are being used to build a 3D vision of the world. Those images need to be segmented into the different entities within them. Those entities need to be put through object recognition algorithms. When two user images have a conflict, like let's say one picture is taken before an earthquake happens and one picture is taken after an earthquake happens, that conflict in the scene needs to be resolved somehow.

Mapillary is building a three-dimensional picture of the world and the company has so many interesting engineering problems. There's this high-volume of images and the level of processing has created the need for a unique sequence of indexing, queuing and distributed processing using Apache Storm.

In addition to processing all of these data and building a 3D model, Mapillary also serves an API for querying geo-locations about traffic signs and road conditions and bus stops. This is the business model, is selling the queries against the Mapillary model of the world.

Peter Neubauer is the cofounder of Mapillary. He's also a cofounder of Neo Technology, the company behind Neo4j, which is a graph database. Peter is a world-class engineer and he joins the show to give a detailed overview of the technology behind Mapillary. From ingressing the photos, to running data engineering jobs, to serving the API.

This was a great technical episode, and also touched on the aspects of business, because you don't have access to infinite capital when you're running a startup, and this is a capital-intensive engineering workflow with all the processing. So it's pretty interesting discussion of tradeoffs.

Before we get started, I want to mention that we are looking for sponsors for Q4. If you are a company that's interested in reaching the 50,000 engineers that listen to Software Engineering Daily, we'd love to have you as a sponsor. Please send us an email. I'm jeff@softwareengineeringdaily.com, or you can check out our website. We have a contact form. You can fill out the contact form and tell us what you're looking for, and we would love to have you as a sponsor.

We're also hiring right now. We're hiring a podcaster. We're hiring writers. We're hiring other jobs. You can find us at softwareengineeringdaily.com/jobs. We'd love to hear from you. With that, let's get to this episode.

[SPONSOR MESSAGE]

[00:03:43] JM: I have learned a ton from QCon. QCon San Francisco takes place this year, November 5th through 9th, 2018, and I will be going for my fourth year in a row. I always love going and seeing the talks, and in between the talks I hang out and eat some mixed nuts, chat with other engineering leadership about the latest talks and stuff that they're seeing, the 50 different stream processing systems that we're seeing, the different databases we're seeing, and QCon is a place where senior software developers and team leaders and managers and senior leaders, they all gather together, and you have fantastic conversations. You have fantastic presentations, and it's extremely high quality. Its technical. A lot of it is also cultural and about management, and you can get $100 by using the code SED100.

QCon is a great learning experience. The presentations this year include 18 editorial tracks with more than 140 speakers from places like Uber, Google, Dropbox, Slack, Twitter. They are curated high quality talks, and you can get $100 off if you use code SED100 at checkout for your ticket. Again, QCon San Francisco takes place November 5th through 9th, 2018, and the

conference is the 4th through 7th, November 8th through 9th are the workshops. You can go to qconsf.com to find out more.

Thank you to QCon. If you are going to buy a ticket, please use code SED100, and we really appreciate the sponsorship of QCon.

[INTERVIEW]

[00:05:40] JM: Peter Neubauer, you are the cofounder of Mapillar. Welcome to Software Engineering Daily.

[00:05:44] PN: Jeff, thanks for having me.

[00:05:46] JM: We're talking about Mapillary today, and this is a mapping platform. We have Google Maps. That's the mapping platform that people are most familiar with. Why do we need another mapping platform?

[00:05:57] PN: Mapillary is not actually a mapping platform. We don't build maps. We are helping to fix the world's map, crowdsourcing, computer vision and everything that we can do. Basically extract map features from ground truth.

[00:06:13] JM: What's the relationship between Mapillary and a mapping platform like a Google Maps or OpenStreetMap? What kind of other previous technologies do you build off of?

[00:06:25] PN: We build mostly not on top of maps, but actually on computer vision and deep learning and, of course, big data. So what we do is to take in ground truth in form of 2D data and extract and define map features that can be used to improve maps, but even do other things. We will talk about that later. But yes, we are a digitalization platform for converting analog ground truth into digitally usable information that others can, for instance, build maps on.

[00:07:00] JM: In order to build something, like a Google Street View, Google required really large investments to get these cars with these big cameras on top of them. How do you gather your imaging data?

[00:07:14] PN: We actually don't gather any imaging data at all by ourselves. We let anyone with a camera that can capture imagery and a possibility to geo-locate these to upload pictures to our platform or videos. We then see to the de-distortion and to the representation of these images in the same reference system that we have and make sure that all the cameras and all the angles and distortions get normalized so we can actually overlap these features, a 3D space. Maybe I can just briefly explain how these things work.

[00:07:54] JM: Please do.

[00:07:56] PN: So if someone uploads an image or a video to our platform, we will take these frames or images and we will first try to find semantically interesting features of these images. So we call it semantic segmentation. So we segment every picture in that image to its most probable class. One of our 150 classes that might be building sky, road, markings or whatever, we then take out the volatile parts of these mappings, for instance, clouds and sky, and people, cars and so on, things that are moving. We then take the rest that we assume to be static and we apply another concept to this, which is called structure for motion.

So we find matching features, matching pixels and edges and so on in images that overlap that we think on the same area and looking at the same scene. Via that, we do like a big triangulation of all these things simultaneously and we gain a depth map between for every image. How far do we think the pixels are from the camera? We then overlap these to gain a triangulated 3D model of that scene. So it's like a sparse point cloud that we get out of different images overlapping each other.

Then in the last step, we apply the segmentation that we have in the beginning to this point cloud. So we now know that this overlapping pixel over there is a lamppost, or is a fire hydrant, or whatever. So we get a point cloud that is semantically understood by our backend. Then the last step, of course, from that, we can use clustering algorithms and other routines to, for instance, find point objects that might be street signs, or line objects that might be fences or road markings, lane markings, crossroads, whatever. That we extract to be map features that are usable and interesting to others.

All these we expose by our APIs, and what you see on mapillary.com is basically just an example, viewing example on this. What you see in the viewer, which looks like street view, is really a 3D reconstruction scene with photo textures on that.

[00:10:30] JM: In contrast to Google Street View, where maybe if I'm looking at the street view, there are some contextual information, like there is some mapping between the street view image and whatever search results I'm looking at, or information about I can look in street view and sort of see, "Okay. This is the McDonalds that's on the corner." I'm not sure if there's signage information that's correlated and schematized and whatnot. But what Mapillary is doing is building more highly schematized, more high-resolution, more normalized representations of imagery.

[00:11:12] PN: I would say as for Google, Street View is basically a side product of their data collection for what they want to build self-driving cars on, which is like high-definition backend maps. For Mapillary, we do the similar things. We don't do high-definition maps, but we do map features, which can be used, for instance, high-definition maps. Also, our kind of street view frontend is just a byproduct.

With some of differences, we expose like not only the imagery or the textures. We're actually exposing all the data. So you get all the segmentations. You can ask Mapillary, "Show me all the pictures that have more than 2% of the pixels being occupied by cars in Sweden." That's where you can place Mapillary.

So the street view part both in Google and in Mapillary is just a byproduct. It's the photo textures in a navigational frame. Also, you can use Mapillary with a lot less restrictions than what you can do in Google. So that's why OpenStreetMap, for instance, is using Mapillary extensively for adding new features to the maps because of licensing reasons also. We are very, very licensing friendly to basically anybody.

[00:12:32] JM: What are some use cases for Mapillary? Some applications that have been built with Mapillary? I mean, you mentioned OpenStreetMap. Maybe you could talk more about that use case and perhaps some other ones.

[00:12:43] PN: Yeah. There's different reasons that people use Mapillary. One reason is documenting things. That is what, for instance, the World Bank is doing when they go in Africa and map areas that they find interesting, or threatened by, for instance, like nature catastrophes or others so they can go in afterwards and look back that, "Oh! This was a school, which is now destroyed rumble of we don't know." This is something that we should look extra on, and this is something we can now map afterwards.

The Red Cross has been driving all over Haiti to assess the infrastructure between storms and so on and so on, and this is then very, very interesting for the humanitarian OpenStreetMap task force, the HOT OSM, to go in and help these organization to when things actually hit to map the infrastructure and to visually assess damage on earthquake damages and so on.

Last year we imported 6 million images from Microsoft Bing Street View in the Houston area in Florida when the storms hit because we have so good integrations from the OpenStreetMap community and with others and then overlay that with new imagery that they themselves take via GoPros and mobile phones and so on.

This is used for kind of visual mapping. The data we extract is also serve, for instance, to OpenStreetMap where we serve all the traffic signs so they can be compared to what is or is not already in that community and they can add it and extract more data from there, and we have a lot of other classifications, like fire hydrants, or crossroads, or lamppost and so on that are highly interesting for these maps. Same thing goes for other big map providers that we have commercial agreements with that use this data to rectify the map. So that's one thing.

The documenting of places in time is another thing. Even private people are doing this to document changes in their environment, parts of towns being build up, schools being built, all sorts of things. Municipalities are using Mapillary for these reasons too for surveying reasons. They want to use the imagery that they themselves have to survey things overtime, like the quality of the roads, the change of environment when building is going on. The quality of the greenery on the side of streets and so on and so on, a lot of these, and they need inventories of all the infrastructure basically. Everything from speed signs, to fire hydrants, to parking signs, which is a huge problem in the U.S. The parking infrastructure is largely surveyed. So we actually right now launching a product with Amazon to assess, help doing this. So cities can get

the parking signs and the text under the parking signs to assess their infrastructure. There's like $53 billion spent on not using these parking spaces a year. That's a huge problem.

Of course, one big industry is self-driving cars. We are in contact with all these manufacturers, platforms and so on that deal with that space. Mapillary, of course, is a perfect platform to channel the information that is in the cars from all these cameras. I mean, you talked to comma.ai. So there's a huge flood of information coming which needs to be factored in to these HD maps and so on, and Mapillary is a backend that is vendor neutral and open that can be used to extract this information and delivery the extracted map features back to the backends that produce the HD maps together with other information that's coming from auto imagery and so on and so on.

[00:16:54] JM: The example of having cars going around Houston after the floods, or going through Haiti and building an understanding of the damage that has been caused by the natural disasters, that seems like a very different application than recognizing highly-defined structures, like stop signs, or stop lights, or street fronts. Is there a different data path that influx of data is following when you're trying to, for example, survey damage versus look at street signs?

[00:17:34] PN: Right now there's not. Right now, surveying for these organizations is about visually orienting themselves in a street view like manner. Maybe take some of the objects that we have into account to find entry points, but basically they're looking a route and assessing visually the infrastructure.

However, what we're doing at Mapillary is right now we are segmenting a static set of 150 classes that we ourselves find useful for both classification and structure for motion and so on, and for our big uses, which is street level normal classification, that's buildings, that's trees and so on. You can look it up on the website.

However, we are working on closing the kind of training loop even publicly. We already have some of that out that you can classify and approve or disprove detections that our algorithms had been doing, which then is used for retraining. You will be able to do that with random classes, with something that you come up with might be like double doors, or whatever you can think of. You can then classify things, say, a thousand signs, or a thousand rectangles in 2D

data. You can then let the system train on this such as more do a human feedback loop of approve, disprove, correct and so on until this is good enough to be retraining that algorithm. Then this comes back and finally you will get hopefully a satisfactory acceptance rate for these detections, and then you can merge this into objects or have these detections guide you and maybe do that in all of Haiti on custom classes, which will be fantastic.

[00:19:30] JM: Before we get into the engineering pipeline and the data, the series of steps that a piece of data or an image or a collection of images might go through in order to build this application platform, I want to understand where we are contextually and why there are so many startups related to mapping. So there's Mapbox, there's SafeGraph, there's Deep Maps. There's been a boom in new companies that are interested in mapping the physical world to more structured digital space. It's no surprise to me that this is useful, but I would like to know why has there been a boom now. Why didn't it come earlier? Were there's some pivotal technological breakthroughs that we needed to make or is it more an aspect of now there's market demand because self-driving cars are coming?

[00:20:30] PN: I think there are different aspects to it. I think that one of the breakthroughs that has been done is that we finally have ways to make the digitalization of reality scalable both in terms of humanly scalable, which OpenStreetMap is doing. You're engaging a lot of kind of fringe contributors. You don't need to be a mapping expert to actually produce high-quality data. Then on the automated side, everything from auto imagery to computer vision are powered by AI or I would rather call it deep learning during the last years breakthrough in computational power. I think you covered it in some other episodes. There has been this new surge of deep learning in this space, which powers a lot of these efforts. Then the rise of maps are coming into and mapping system coming into mainstream and also into automotive is certainly one more point. Making people realize that maps are a fundamental and integral part of our infrastructure that we need to make these systems work and the information in them needs to be democratized and open because it is part of the public space, and that needs to be usable even digitally.

[SPONSOR MESSAGE]

[00:22:07] JM: DigitalOcean is a reliable, easy to use cloud provider. I've used DigitalOcean for years whenever I want to get an application off the ground quickly, and I've always loved the focus on user experience, the great documentation and the simple user interface. More and more people are finding out about DigitalOcean and realizing that DigitalOcean is perfect for their application workloads.

This year, DigitalOcean is making that even easier with new node types. A $15 flexible droplet that can mix and match different configurations of CPU and RAM to get the perfect amount of resources for your application. There are also CPU optimized droplets, perfect for highly active frontend servers or CICD workloads, and running on the cloud can get expensive, which is why DigitalOcean makes it easy to choose the right size instance. The prices on standard instances have gone down too. You can check out all their new deals by going to do.co/sedaily, and as a bonus to our listeners, you will get $100 in credit to use over 60 days. That's a lot of money to experiment with. You can make a hundred dollars go pretty far on DigitalOcean. You can use the credit for hosting, or infrastructure, and that includes load balancers, object storage. DigitalOcean Spaces is a great new product that provides object storage, of course, computation.

Get your free $100 credit at do.co/sedaily, and thanks to DigitalOcean for being a sponsor. The cofounder of DigitalOcean, Moisey Uretsky, was one of the first people I interviewed, and his interview was really inspirational for me. So I've always thought of DigitalOcean as a pretty inspirational company. So thank you, DigitalOcean.

[INTERVIEW CONTINUED]

[00:24:14] JM: Just to talk a little bit more about the high-level business use cases before we dive into the engineering. What about drones and augmented reality? Augmented reality is interesting, because the SDKs that have come out are really powerful. You can build really cool stuff, but it's kind of like cryptocurrency where we haven't really seen the "killer app" of augmented reality yet. Drones are really interesting. They've been used for fairly basic things, just like flying around and taking pictures of stuff, which is super useful and there's a ton of work to do there. But I feel like when you take augmented reality or you take drones and you add object segmentation to it, either from the point of view of just the fact that drones could be used

to pull in images instead of using crowd sourcing people, except that there's perhaps a stigma against having drones flying around consumer spaces. That can get you a much faster influx of data. Then on the other hand, the applications that you could build if you had a semantically segmented world for augmented reality applications or for drones, that seems potentially huge.

[00:25:31] PN: Yes. That's certainly a huge use case. However, we are actually powering one of the bigger platforms for 3D reconstruction based on drones, which is called Open Drone Map. That is building on the same open source project, open structure for motion that we are putting out on GitHub as Mapillary itself is.

We feel, however, that the reconstruction and the augmented reality is not big enough of a problem or gain or killer app as you put it for us to actually go into there and kind of merge drone imagery, which is kind of orthogonal to the surface with street level imagery. What we at Mapillary try to do is concentrate on street level ground imagery, because that's scenario that is very, very relevant and cannot be covered with the existing technology. I mean, drones and space photography and so on and so on.

Also, the digitalization of the street level space is highly, highly relevant for all these industries that are talked about before and it is all build on computer vision and visual impressions, because the whole space of roads and paths and so on is built for humans and they take in 80% of their information via visual input. So that's where we see the strength of Mapillary. We could of course build a 3D representation of all the angles and so on, but we would spread ourselves thin. There's more to be taken to consideration if you also alter the attitude of auto imagery.

[00:27:20] JM: A data processing pipeline that just takes into account street level images is not necessarily a good fit for a set of drone images. It's trained in a different way.

[00:27:34] PN: It is trained in a different way, but also the altitude is actually one of the hardest to get write parameters in this space. The GPS altitude is not reliable, especially not in phones, and Mapillary is a platform where we bring our own devices, and the data that we get in is comparatively bad. The GPS positions, for instance, can vary up to like 15 meters or something and we still can process them to a degree where we can back correct visually by overlap with the other imagery the existing photos and make the positioning better. If you factor in drones,

the amount of good data you need goes up exponentially and we just don't think that the data is worth the effort right now.

[00:28:22] JM: It's like adding another access. I mean, for the street level view, you kind of just have X and Y, and drone you have the Z axis as well, which may be highly variable.

[00:28:29] PN: Yeah. We have the Z axis already, but it's very hard to get accurate data for it. We have, for instance, for positioning, the Z axis is very, very important. For instance, if you are a mountain and see a traffic sign. What you think is 10 meters away, if that is standing in San Francisco going downwards, then it's actually much farther away.

In the U.S., for instance, there's height data for the whole country that we can factor in so we know how to calibrate, because we cannot rely on the actual positions of the imagery we're getting. They have zeta positions, but it's too bad to actually be interesting worldwide. Given the data that we operate in the consumer space, it's simply safer to assume that things are 1.5 meter over ground.

[00:29:22] JM: Let's get into the data pipeline. Describe the data gathering in more detail. How does it happen? How does it involve the crowdsourcing mechanism? Describe the data gathering process.

[00:29:38] PN: Data gathering is actually done with anything that can give us positioned 2D data. So we have our own applications that basically are sequence shooting applications. You put them on and they will shoot an image every second or every other second depending on what you want and how much space you have on your memory card. When you get to Wi-Fi, you will upload that data. Mostly, we encode the interesting metadata into a JSON structure in the EXIF description field, which we then extract again.

We also gather GPX traces so we can later, for instance, make adjustments. It is, for instance, much more safe to assume a certain constant offset to the moving direction of a camera and mostly pointing forward and get the compass angle factor via that offset calculation along the trace. Then to actually trust the GPS compass on the phone, which always has drift and is not accurate and so on.

You then upload these images to Mapillary and we will harvest them. It's an upload so an S3 bucket and we will scan that bucket and harvest the image meta information. At the same time, we take the image and we blur it. It gets segmented with a special segmentation network that finds faces and license plates, which is very, very stable and it's one of our things that people want to use outside of Mapillary too, because it's currently the world's best blurring network. Considering GDPR, that's very interesting to a lot of people.

This gets segmented and anything that's classified as face or license plate is then blurred. Thumbnails are done in different resolutions. Put back on cloud frond so we can access them. After that process is done, we calculate the sequence trace of all these uploaded images in a sequence. So that's where the green lines on the Mapillary map come from. When that is done, when we have collected, say, you uploaded a hundred photos or a thousand and they get connected into one sequence and ordered by time and are privacy blurred, then you get a notification and the image is publicly available that this image has been processed.

After this, we are having several kind of processing ways on these images. So one is, for instance, this segmentation on the street level stuff that we need to actually specially connect it. The first thing you get is just time-wise is sequence hop that you can basically do a stop motion movement along that sequence. When that next process is done, we wait for it a couple of hours, because we want to collect more changes in different tiles that we process. When you upload an image, you actually dirty a little tile around that image where already there is a 3D reconstruction.

In order to process this incrementally and not need to reprocess every time an image comes in, we wait an hour until maybe all the 50 images that cross this tile in a sequence are uploaded and we then reprocess that dirty tile with the merging of the new visual model, or the 3D point cloud, which then has effects on the already existing imagery. Because we actually wriggle the camera positions into place so that they best fit the visual overlap in their accuracy radiuses. So that means if I come with a bad consumer grade phone and do a shitty sequence through New York last year, which has an accuracy level in these urban canyons of maybe just 15 meters, and today the city of New York uploads high-resolution, super nice GPS with just a few centimeters accuracy radius in that tile, then my images will actually have better corrected

positions than last year, because basically the visual model gets anchored at the points of the municipality's imagery, right?

[00:34:10] JM: Wow! There's a lot there to unpack, but I want to first go back to just the crowd-sourcer part of it. I'm a crowd-sources. I've downloaded the app to my phone and I'm going to start taking pictures. Why am I doing that? What's my incentive as a crowd-sourcer?

[00:34:27] PN: There're incentives. Some people are just wanting to have a documentation of the places. There's parents documenting their Sunday walks for their kids and sending them the links. Then there's open street mappers who want to map. A lot of people is attracted by the collectiveness of Mapillary. They actually collect stuff, right? A lot of times in the OpenStreetMap, for instance, there're lots of collectors, but they cannot really collect a lot of things anymore because all the streets in, for instance, Germany or Middle Europe, Central Europe are basically there. You can contribute tags. You can contribute opening hours and whatnot, but the initial fill these white spots on the map, euphoria is not there anymore.

People want to go out and to actually exercise and collect data. Then you have the more structured mappers, like municipalities, departments of transportation, national authorities, national road authorities that import data that they either want to keep to themselves, or that they already have and want to use the Mapillary backend and at the same time make this information publicly available to the citizens, because all the stuff that comes out of Mapillary and processed images and so is creative common share likes. So you can use the Wikipedia on the websites and so on and so on, which is very friendly for these municipalities and they know that they easily can reuse that data and let others use that data too.

[00:36:10] JM: Yeah. I think I heard you talking about the fact that you wanted to be really careful about whether or not you introduced a paid component of this, where you would potentially pay these crowd sourcers. It was an interesting point that you made, because you were just conveying the fact that if you do introduce money into a crowdsourcing kind of environment, you really risk degrading the feeling of – Maybe you would want to call it charity or just social contribution, and it sounds like that is such a significant source of momentum for building your image repository and your data labeling that you have been – I don't think you've

done any paid acquisition of crowd sourcers. Maybe you've paid some of the ambassadors or like some of the leaders?

[00:37:03] PN: No.

[00:37:04] JM: No? None of them?

[00:37:05] PN: No. We haven't paid anybody. However, some of our contributors are paying others and they ask us to help with that with the organization and so on of these paid captures. I mean, there are cities in the U.S. that pay taxi drivers to capture the city for them, because it's too stressful for them to put cameras on garbage trucks and go around. Even though if that is fully possible and then they ask us to coordinate that. So we have one team member who does data acquisition, but that's on a level of helping these partners that want to get data into Mapillary. So they need help with the technicalities and the coordination and so on.

Yes, we don't pay anybody to do that. People get hopefully enough value out of Mapillary by free hosting and by opening this and the dataset that we produce is not cheap comparing the CPU hours and GPU hours that we crunch there. It's not a cheap dataset.

[00:38:01] JM: Do you think this structure of crowdsourcing is comprehensive enough? There's enough people who want to do it that you can get a big enough dataset, or are there tail cases where you're just like, "Man! None of these people are going to this place in Zimbabwe. We really might need to pay somebody to go to Zimbabwe and take some pictures."

[00:38:20] PN: There're two things to it. On one hand, Mapillary comes from Capillary. So we are assuming that anybody who finds something interesting and has the will to do it can contribute data. So if someone thinks that their village in southern Bangladesh is not mapped enough, then take out a camera and do it, or send someone with a GoPro around, right?

The other thing is that we want to have data where it matters. For many players, it's not that important how much the global coverage of Mapillary is. It's more important how big the coverage is in their local communities. A lot of people are focused on what we call shapes. We have – On the platform, you can create shapes and mostly people are doing that and are most

concerned about their shapes, their neighborhoods, their municipality and so on. That's where they care. They don't care if some player is mapped or not.

[00:39:22] JM: That is an amazing network effect that you've got going there. Let's talk more about the data pipeline. So you've got all these people. They're uploading their photos to S3 buckets automatically with the app. The photos land in an S3 bucket and then what happens? I know you mentioned that in abstract, but let's dive in a little bit deeper.

[00:39:44] PN: Yeah. Basically, we have two types of backend processing systems. One is a stateless worker-based system, and the other one is a stateful system that actually takes order into account, right? So the first thing that happens is that our harvesters go in. There're two different types of harvesting. One is what I explained to you about the apps that basically have self-contained EXIF data in the images. So you just put them there and they know where they belong, what sequence they belong to. So all that is encoded in the EXIF, and then we have another way of uploading, which is you put images that don't have any EXIF and a GBX file or a video with a GBX file into a sub-bucket, or a subfolder and we will harvest that and time match the images to that GBX file to get the positions and the offsets and so on. These are two types of harvesting, two types of information coming in.

When we see these images, we're first creating an entry in our system or we are creating a unique UID and sending that to our Kafka event source center. We send it to a topic in Kafka, and on the other hand, there is a consumer consisting of an Apache Storm cluster. That's one of your episodes that was highly interesting. This cluster then talks to the different backend.

We currently mostly store our data in a post.js cluster and Elasticsearch cluster, and that Storm cluster will update these events into these backends. For instance, in this case, an image created with a key and basically nothing. Then after that, we have schedulers looking in, for instance, elastic search every minute also for unprocessed images. So we keep the image state, what has it been processed with, what's the metadata and so on in this image document. The scheduler will look in and say, "Oh, there's like no processing information on this at all. So I will schedule it for basic processing," which is basically EXIF extraction and thumbnailing.

That gets put on to a queue by that scheduler, that's RabbitMQ, and these queues are stateless worker queues. So there's now an image process job on that queue and then we have a lot of processors, image processing processors that are deployed on Amazon and look at this queue. They will act the message when they have processed it, and they would do that in parallel as fast as they can.

Meanwhile, the scheduler sends an event. The image state processing has now started to process, gets persistent by Storm, again, into Elasticsearch. It will not be rescheduled, because the scheduler will see, "Oh! This is actually processing right now." When this is done, it will send, again, a message that says, "State is not processed." Then the next wave of state processing goes in. Now we need to structure from motion it, then we need to do this and this." So we have this dependency graph of processing that is going through to all these.

[SPONSOR MESSAGE]

[00:43:09] JM: OpenShift is a Kubernetes platform from Red Hat. OpenShift takes the Kubernetes container orchestration system and adds features that let you build software more quickly. OpenShift includes service discovery, CI/CD built-in monitoring and health management, and scalability. With OpenShift, you can avoid being locked into any of the particular large cloud providers. You can move your workloads easily between public and private cloud infrastructure as well as your own on-prim hardware.

OpenShift from Red Hat gives you Kubernetes without the complication. Security, log management, container networking, configuration management, you can focus on your application instead of complex Kubernetes issues.

OpenShift is open source technology built to enable everyone to launch their big ideas. Whether you're an engineer at a large enterprise, or a developer getting your startup off the ground, you can check out OpenShift from Red Hat by going to softwareengineeringdaily.com/redhat. That's softwareengineeringdaily.com/redhat.

I remember the earliest shows I did about Kubernetes and trying to understand its potential and what it was for, and I remember people saying that this is a platform for building platforms. So

Kubernetes was not meant to be used from raw Kubernetes to have a platform as a service. It was meant as a lower level infrastructure piece to build platforms as a service on top of, which is why OpenShift came into manifestation.

So you could check it out by going to softwareengineeringdaily.com/redhat and find out about OpenShift.

[INTERVIEW CONTINUED]

[00:45:18] JM: This is a really interesting application of Elasticsearch. If I understand it correctly, because you got this big influx of images, they get put in S3 and then Storm is routinely grabbing the metadata from these images and indexing that metadata in Elasticsearch, and then you have these schedulers, or these orchestration systems that are talking to Elasticsearch and saying, "Hey, Elasticsearch, have there been any new photos that have been uploaded that have not been processed yet?" and Elasticsearch says, "Yes, these ones have not been processed," and then the schedulers grab those photos and start kicking off all of these image processing jobs. Is that right?

[00:46:00] PN: Yes. That's basically right.

[00:46:03] JM: That's fascinating. I don't think I've seen or I've heard of Elasticsearch being used that kind of application. Maybe just to give more context, explain why Elasticsearch is a good data store for that? Why couldn't you just use like a SQL database and then have a field that says, "Has this been processed or not?"

[00:46:21] PN: Actually, we did that in the beginning. The whole system evolved from a little Ruby on Rails systems based on post.js and we stored the current processing state as a dictionary in an [inaudible 00:46:35] column, and that way coordinated the processing of the images and so on. Until recently it was actually used, but half a year ago, the whole thing just ran out of bounce. At that point we crossed 200,000 images a day. We are now at almost a million a day.

The Rails application was starting to – It keeps all the state back in PostgreS. The amount of transactions to do this was just skyrocketing and we he had at that log, there's like 2 billion transaction IDs that you can have in PostgreS. We were at one point 3 billion and we had six weeks to go until we would hit 2 billion, because the auto vacuum process could only vacuum 400 million of these while we were adding 600 million new transactions to the database. We were running out of transaction IDs, and that's a hard limit. I mean, at that point, PostgreS will stop responding until it has cleaned. The vacuum process, one run, took 3 weeks.

We had to migrate all that data and everything to Elasticsearch. We were using that in parallel before, but as you say, the state keeping was actually killing us. Then we moved it to Elasticsearch, and also we moved the whole logic from Rails to Storm and the amount of transactions to get an image through the whole pipeline went to basically zero. Because as you know, in Storm, if you do it right, everything gets batched at the end and there's like bulk inserts and updates going on. That, of course, is vastly more effective than an event-wise several transactions Rails application.

[00:48:38] JM: But aren't you like transactional still? Like transactional databases that you would rather use than something else? I'm not sure transactional is the right word, but Cassandra, or isn't there something other than like Elasticsearch, the search thing?

[00:48:55] PN: What made us use Elasticsearch was the fantastic searchability of Elasticsearch. I mean, the underlying Lucene is fantastic. We were actually experimenting with Neo4j in the beginning, because I'm one of the oldest Neo4j founders, and it made the application very, very fast, but we had to batch, recreate it like every half hour, but that didn't work out in the end, because the varies and the problems we have are highly spatial-based. We are basically always having either a shape, a polygon, or bounding blocks that we materialize the data for. Therefore, good spatial support, fast spatial support together with good sharding of indexes and so is paramount for us.

Elasticsearch is actually fitting that build quite well. What it didn't fill the build for in the beginning, we started at 1.7, is the actual operationablity. Field data was just skyrocketing as we got more data. It was not having any throttling on requests. We had a lot of trouble with that. We are still running our Elasticsearch cluster ourselves and not trusting Amazon or anyone else to

run it, because we actually need to profile the heap and so on sometimes. But it has been, since we upgraded to 6.3, it has been much smoother sailing. It's still hiccups and so and performance problems, but not near the level that had before.

[00:50:26] JM: Is Elasticsearch like the source of truth data storage system for the different –

[00:50:34] PN: No.

[00:50:35] JM: No. Okay.

[00:50:36] PN: The source of truth, it's now a truly event source system. So the source of truth is the Kafka events. When we run into trouble, we have a retention of 7 days, or 100 gigabyte per topic for the Kafka queues. Within a week, we can actually replay the whole event log for a topic back into Storm and basically refill or correct the data in both Elasticsearch and PostgreS. However, after that, we kind of trust the database, which is snapshot. So if something is happening older than that, we're not always keeping the logs since the dawn of time. We're actually cutting them and archiving them at some point. So it becomes impractical to actually rerun the whole system since the dawn of time. We could do that, but it's not practical.

So at that point, we rely on old snapshots to catch up to a certain amount of time, or we always have the possibility to actually rerun harvesting. So one of our catastrophic scenarios as a devops is to actually say, "The database is corrupted or it's just gone." So we need to recreate the data from the source, which is the uploaded imagery. If bad comes to worst, that's the source of truth. We will re-harvest. I just fixed the bug where we had, for instance, a timestamp problem in 2017, which we couldn't fix, because the timestamps in the EXIF was actually wrong.

Now, a year later, we got to it and actually fixed the harvester so we would take account for these faulty android version, and we re-harvested to the 207,000 images that were harvested with that, including the reprocessing of all – The follow up effects of that. Yes, that's the source of truth if it is on the long run. Yes.

[00:52:33] JM: Right. Okay. I'm sure we could spend much more time on this influx process, the ingress process. But let's fast-forward to you've got these orchestrators that are looking up fresh

images that are in the S3 buckets. They're querying Elasticsearch. They're getting the fresh images. They're doing some processing on those images, some various learning machine learning based, or models that are doing classification and segmentation and things like that. Then eventually you're going to be adding insights from these images into your model of the world, which get served by an API layer.

We only got like 15 minutes left, so I want to get through all the way to the API layer. I think probably the next phase is just to talk about what are you doing in that model processing phase, and then how is that data getting integrating into your big model of the real world? Maybe like what is the data structure or the database or whatever for how you are representing that real world schema that can be queried?

[00:53:42] PN: What I just explained was kind of the stateful part of the system. The stateless part of the system is a lot of Docker containers doing different image processing on this data. We scale everything by Docker containers that we deploy on a Mezos cluster, which is running on Amazon slaves.

For the image processing layer, we have actually a constraint layer, that's our burst layer that runs with spot instances. So we get a bit down on cost on that and we scale it dynamically looking at the queues in RabbitMQ. Then the different – So some of the processors, as I explained earlier, are extracting stateful EXIF information. Some of the others are, for instance, creating that segmentation information and submitting the data again as events into Kafka and later on into Elasticsearch, putting out polygons of different classifications. So these polygons get persisted into different indexes. For instance, the traffic sign segmentation index or the street level general segmentation index that classifies these 150 classes.

Per image, there might be around 200 to 800 polygons. So that's like one image document per segmenter, segmentation process with these polygons. So right now we have about – I think it's 40 billion polygons on these 350, 400 million images. This is all stored in Elasticsearch, so it's quite hot. So we can actually vary it, so it can find give me stop signs classifications that have more than three instances on the image taking up a space. We store the area of 30% of the image in the lower left corner. That's quite [inaudible 00:55:32] varies. You can combine that of course with the position and say, "In the State of Wyoming by this user," and so on and so on.

So that gets stored in Elasticsearch. Then the structure from motion processor is taking this semantic information and only considering the polygons that are static things. It will create a depth map or a point cloud, local point cloud on the image. That is stored in S3. Then served together with the image under the same image key.

Then we have a merging processor that is taking a lot of these local depth maps and segmentations or the local point clouds of these images and it's merging them together into a global point cloud. That one is, again, stored as actually vector tiles format in S3 or cloud front. So we're using the Mapbox vector format to access these, because that's very, very effective. Then when we then reconstruct the sequences and the images are done, we will extract vector tiles that we put on S3 also in order to save hits on the database.

When you vary the map features and so on Mapillary, you're actually getting vector tiles with the different layers and the positions of the images and so on at the keys and so on. So that makes it much more scalable without hitting the databases live too much.

[00:57:00] JM: Explain what a vector tile is.

[00:57:02] PN: A vector tile is a map format that Mapbox put out. It encodes the data that you need, normally what you have in maps is raster tiles. Raster tiles are basically images with transparency that give you a visual representations of features on a map in a layer. Say, you're building a layer, raster tiles is just PNJs that cover the tile that you're asking for, the geographical bounding box in different levels, and you can layer them over each other.

Vector tiles on the other hand are tiled data. It's not tiled imagery. It's actually tiled data, which means you do basically the same varies. You say, "I need zoom level 15 center of this and this, or a geohash," and you get back a number of tiles, which are data tiles, and the vector tile data client can stitch them together. So they become a part of the global data layer that is bounding blocks.

The good thing with that is that you then can apply styles and so to them dynamically just with any CSS on the client side instead of having raster tiles that need to be styled on the server

side. Yes, totally dynamic. What you display, how you display it and so on while keeping the scalability of the zooming and tiling approach to maps that has started with these raster tiles. Leaflet for instance is one library that's very good in extracting these raster tiles. So one Mapbox vector tiles are doing that on data layer.

[00:58:44] JM: Okay. Take me through the life of a query, a user incoming query from just an API request to Mapillary.

[00:58:53] PN: It depends on what query that is, of course. For pure images, we have a static endpoint that goes to CloudFront and it goes through an image server that can recode that and has a bit of indirection there.

For normal API requests, we first have a S3 load balancer that goes to an – That terminates the SSL traffic, of course, and does some basic stuff. Then it comes down to our marathon Mesos cluster that has a DNS load balancer, an AJ proxy in front of it that balances round robin containers that are scaled to different instances. So we have an AJ proxy Docker container that scale to, say, five instances and gets round robin requests from outside. It will then determine what endpoint this is. Do the authentication on the client ID and it will decode the JWT token that you have to identify yourself, and attach that to the request and send it to the different backends. We have actually different backends serving different things.

Further down, most of our version 3 requests are going to a backend server that runs a Falcor-based API. Netflix Falcor is an alternative to GraphQL. That API is a graph of data that it can query that we expose as REST and for our own applications as actually a graph. That one will then make the actual calls to the databases or serve you cache data. If you're asking the same user and their sequences two times, you will actually get a cached graph subset of that data. So that saves us a lot of data requests to the backend.

Then this gets JSON packeted or REST packeted and served back. So our database is basically read views of our data, and the only thing that writes to the databases is the Storm cluster through the event host. Even a user interaction is resulting in API events that go through Kafka, through Storm into the backend and you will get a cached version of what the result

might look like up to the endpoint, but it will be eventually consistent after few seconds, or one second. You will actually get the real data that was persisted.

[01:01:19] JM: So the API request, is that like a geo location? Like, I want this location in space, and then the response is vector tiles?

[01:01:27] PN: A lot of requests are either GeoJSON or vector tiles. Actually, we have a vector tile encoder that serves you dynamic vector tiles. If you ask a bounding box, you might on some request that are vector tilable, get these back, because they're just so much easier to consume on the client. So you wouldn't really know if you get statically pre-fabricated vector tile request back, or our API is giving you a dynamically generated vector tile. You can also request basically all of our APIs as GeoJSON.

[01:02:00] JM: Okay. I know we're running out of time and we could certainly go deeper on that element as well. By the way, this is one of the interesting software architectures of the interviews I've done. You've got a lot of unique problems to solve and that shows in the solutions that you're building. Is cost management an issue?

[01:02:20] PN: Yes, cost management is an issue. We're actually spending a lot of time on thinking about how to run this data, because there's a lot of data in there, and we can generate even more data. The question is how interesting that data is both in terms of freshness and in terms of global coverage. For instance, we have been producing line features all over the world, because we can. We can merge features, say, curbs, or fences, or lane markings or so just for the sake of it, because we can. However, to do this on a global level, that's very, very expensive recalculations, because for one lane marking, this might stretch over like a hundred kilometers. We need to consider a lot of detections.

One lane marking might have hundred thousand detections. Computational wise, this becomes a very, very expensive line, right? Nobody might ever look at that line. So you have to consider how much you materialize upfront and if it is better to actually have demo areas where you demonstrate this is what we can do. Then request this for your shape, because it might be vastly more efficient to actually run a better algorithm with better knowledge on better data in

your area than to do it kind of fast, but not that accurate globally. But it's not as useful for you as a user. So we're walking that line.

Also, before we do things nowadays, we just don't run a better blurring algorithm just like that. We actually optimize it. Everything we do has to be incremental and not global. A lot, for instance, structure from motion reconstruction is global algorithm, like the old Microsoft Photosynth. That was a global algorithm that takes all the data [inaudible 01:04:15] that reconstructs the scene. We can't afford that. We cannot recalculate all the world.

We are spending a lot of time on making these things effective both in production and also in our research labs. That's one reason why we can win these segmentation challenges with almost no hardware. We have a couple of GPUs in our office in Graz that are making people sweat in summer, but this is nowhere near the type of GPU clusters that Baidu or Google are using to brute force their results.

[01:04:48] JM: Okay. Last question, how do you expect your business to change as these self-driving cars get on the road and you have just a gigantic influx of demand and influx of data?

[01:05:01] PN: We're seeing that this would be a fantastic search towards the aim of Mapillary. This will pretty much, if we can get partnership with some of these providers, a lot of them doesn't want to open their data to others so others can reuse it. This is what we want, because Mapillary has the mission to make the data public. When we get there, and we get there kind of like country-wise, tone-wise and so on. People come to us and want to do this. Then we will have fantastic coverage, fresh coverage of all the places where these cars go.

Of course, this is nowhere near what Mapillary is trying to do. I mean, we're talking bike path, we're talking hiking trails, national parks, disaster, areas. I mean, this is of course one step closer to where data matters, but data does not only matter on roads. That's what might seem interesting and that's what Google has been focusing on. But there's a lot of other places that people want to have digitalized. Yes, self-driving cars will be a huge boost in that sector, but it's not the end of the road.

[01:06:11] JM: Peter, thank you for coming on Software Engineering Daily. It's been really great talking to you.

[01:06:14] PN: Thank you. It was fun being here.

[END OF INTERVIEW]

[01:06:20] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It's open source and free to use, and GoCD has all the features you need for continuous delivery. Model your deployment pipelines without installing any plug-ins. Use the value stream map to visualize your end-to-end workflow, and if you use Kubernetes, GoCD is a natural fit to add continuous delivery to your project.

With GoCD running on Kubernetes, you define your build workflow and let GoCD provision and scale your infrastructure on-the-fly. GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn about how you can get started. GoCD was built with the learnings of the ThoughtWorks engineering team who have talked about building the product in previous episodes of Software Engineering Daily, and it's great to see the continued progress on GoCD with the new Kubernetes integrations. You can check it out for yourself at gocd.org/sedaily.

Thank you so much to ThoughtWorks for being a longtime sponsor of Software Engineering Daily. We are proud to have ThoughtWorks and GoCD as sponsors of the show.

[END]