EPISODE 751

[INTRODUCTION]

[00:00:00] JM: Prometheus is an open source monitoring system and time series data base. Prometheus includes a multidimensional data model, a query language called PromQL and a pool model for gathering metrics from your different services. Prometheus is widely used by large distributed systems deployments such as Cloud Foundry or Kubernetes. Prometheus gathers metrics from your services by periodically scraping those services. The metrics get gathered, they get compressed and they get stored onto disk for querying, but Prometheus is designed to store all of its records on one host in one set of files, which limits the scalability and the availability of those metrics.

Cortex is an open source project built to scale Prometheus. Cortex effectively shards Prometheus by paralyzing the indigestion and storage of Prometheus metrics. Cortex can take metrics from multiple Prometheus instances and store them across a distributed no SQL database, like DynamoDB, or Big Table, or Cassandra.

Bryan Boreham is an engineer at Weaveworks where he works on deployment, observability and monitoring tools for containers and microservices. Bryan wrote much of the code for Cortex, and we met up at KubeCon North America to talk about the motivation for creating Cortex and the broader landscape of Kubernetes monitoring as well as other approaches to scaling Prometheus. It was a great conversation with some technical depth.

Before we get started, we're gathering feedback in our 2019 listener survey. If you can fill that out, go to softwareengineering.com/survey. Tell us what we're doing right and what we're doing wrong, and you can also send me an email, jeff@softwareengineeringdaily.com to help me chart a course for 2019. We want to know what topics we should cover and what material is useful to you.

We're also looking for sponsors for 2019. If you're interested in reaching the 50,000+ developers that listen to Software Engineering Daily, please check out softwareengineeringdaily.com/sponsor and you can help us out by sending an email to your

marketing director, or your CMO, or your CEO. Many people are surprised by how effective podcast advertising turns out to be for their product marketing or for hiring. If you're hiring engineers, this is a great way to reach them, and it can often be difficult for us to convince advertisers to buy podcast adds. So your recommendation can go a long way.

We do invest heavily back into Software Engineering Daily. So these dollars are not going just to a big treasure chest. We want to reinvest them into interesting product offerings for software engineers and technologists. As always, you can send me an email, jeff@softwareengineeringdaily.com.

With that, let's get on with the show.

[SPONSOR MESSAGE]

[00:03:10] JM: Triplebyte fast-tracks your path to a great new career. Take the Triplebyte quiz and interview and then skip straight to final interview opportunities with over 450 top tech companies, such as Dropbox, Asana and Reddit. After you're in the Triplebyte system, you stay there, saving you tons of time and energy.

We ran an experiment earlier this year and Software Engineering Daily listeners who have taken the test are three times more likely to be in their top bracket of quiz scores. So take the quiz yourself anytime even just for fun at triplebyte.com/sedaily. It's free for engineers, and as you make it through the process, Triplebyte will even cover the cost of your flights and hotels for final interviews at the hiring companies. That's pretty sweet.

Triplebyte helps engineers identify high-growth opportunities, get a foot in the door and negotiate multiple offers. I recommend checking out triplebyte.com/sedaily, because going through the hiring process is really painful and really time-consuming. So Triplebyte saves you a lot of time. I'm a big fan of what they're doing over there and they're also doing a lot of research. You can check out the Triplebyte blog. You can check out some of the episodes we've done with Triplebyte founders. It's just a fascinating company and I think they're doing something that’s really useful to engineers. So check out Triplebyte. That's T-R-I-P-L-E-B-Y-T-E.com/sedaily. Triplebyte. Byte as in 8 bits.

Thanks to Triplebyte, and check it out.

[INTERVIEW]

[00:04:59] JM: Bryan Boreham, you are an engineer at Weaveworks. Welcome to Software Engineering Daily .

[00:05:02] BB: Thank you very much. Very happy to be here.

[00:05:04] JM: So we're talking about scaling Prometheus today, but we should start by talking about what Prometheus is. Can you explain what Prometheus is and what problem it solves?

[00:05:13] BB: Sure. Prometheus is a time series store. So the problem is if you want to look at some number like the CPU usage over time, you switch on your computer, it's not doing very much. That's going to be a low number, and then you start up a bunch of services and your CPU usage goes up. So that's what we call a time series, a number that varies over time.

Very often you want to aggregate that number up to the big picture and then you want a slice it down to find the what makes up that number. So that leads to the complexity of the problem. You will typically have tens of thousands of those time series of different kinds, telling you different things, and you want to aggregate them up, see the sum, see the average, that kind of thing, or slice them down to understand what they're showing you.

[00:05:59] JM: Prometheus is gathering all these data originally in the format of events. Is that correct?

[00:06:06] BB: I would say not. Basically, it's a process called scraping, and it goes out and makes a call like a web service call. It talks to a bunch of different services. That's one of the superpowers of Prometheus is that it can automatically discover what you're running. So if you're running one thing or 10 things or a hundred things, it can reach out through a variety of mechanisms, of service discovery mechanisms, and find them, make that call, and it's a pool-based model. It calls out and it pulls back a set of metrics.

Then on some interval, so we typically use every 15 seconds, it will do it again. So it gets a sample every 15 seconds for every time series, and that builds up overtime and that sunlight gives you the raw data which you typically render as a chart, a graph overtime.

[00:06:55] JM: Okay. These scraped metrics, we'll say every 15 seconds you scrape what your different Kubernetes services that are running.

[00:07:03] BB: Right. So if you're running Kubernetes, you'd want to probably scrape all the services, all the pods. You'd also want to gather information from noes, like the computers that they're running on. You probably want the ask Kubernetes for some metadata. For instance, a pod that is pending won't have any metrics of its own, because pending means they're not running. But that's an interesting metric. The number of pods that are pending, that tells you that something may be not quite right about your cluster. So actually Prometheus would interrogate that from Kubernetes itself.

So definitely you would configure it to gather all the metrics you can from your application software also from your infrastructure, from your middleware, from anywhere. The Prometheus community has a huge set of agents that can gather that data. They're usually called exporters, because they export some data. So they go by names, like node exporter, gives you the node level, the machine level data. There's a bunch of them, and they can convert from other types of metric data. A good example might be JMX, which is a sort of standard Java metric data, and bring it all into Prometheus no matter where it starts from. Bring it all in and have that data available for analysis.

[00:08:22] JM: So in the lifecycle of one of those 15-second cycles where you go out and scrape all the different services and pods and basically whatever you want to scrape, whatever it has an agent on it. What is the format that it's being stored in and is it a single time series entry or is there a different entry for each of the different things that you're scraping?

[00:08:51] BB: Well, let's break that down. So the data comes back from the source in basically textual format in ASCII with like the name of the metric. So to give a concrete example, HTTP request duration, name of the metric, and then some labels, because you want to be able to

slice and dice it down. So maybe the name of the service or the name of the URL path or whatever you like. You can get creative. The labels or the things you slice and dice on, and then the value itself. So if that was a duration that might be .1 of a second.

Then the next time it interrogates that metric, maybe it's not .2. Maybe it's a different number overtime, and the time series are stored individually for each unique set of labels. So if you had 10 copies of your service running, that will be 10 sets of time series. Each one of them might be reporting metrics for 50 different endpoints. Each one of those you might want to slice by whether it was an okay response or an error response. So you generally add all those tags on the labels to the data, and each unique set of labels gives you another time series. So you can quickly get up to the thousands of time series. If you're running a big infrastructure, you will be in the hundreds of thousands or millions of time series, and that's the basis of the system.

[00:10:15] JM: Is it writing to a database or writing to a file?

[00:10:18] BB: Yeah, Prometheus is writing to a file. I mean, it's effectively got its own time series database. It employs a compression mechanism. Goes by the name of Guerrilla compression, if you want to Google detail behind it. So this compression is basically taking advantage of the fact that one value is likely to be very same very similar to the one before. So it compresses down.

So you start off with like an 8-byte floating-point value and it'll compress down to like one byte and, yeah, that goes in a file basically. The time series are also indexed. So that's in the file as well. Yeah, that's basically how Prometheus works. It's very simple to run one process and it writes to one area on the disk.

[00:11:03] JM: Describe how a user queries Prometheus.

[00:11:06] BB: Okay. So there is a little UI built into Prometheus. They can they can start there. They can type a query in the PromQL language, and that basically looks like the name of the metric and some curly brackets and the values that you want to narrow down. Let's say you want to look at only one service or you want to look at only the 500 response codes. So you put that in the curly brackets and then you might want to apply some functions to the data, like you

want the average, or you want the sum of one thing divided by the sum of another thing. That is all built into the PromQL language, basically statistical functions.

The result of that will then be another time series. So typically you would use a tool that can take the numbers overtime and render that as a nice colorful chart. I would say one of the most popular is Grafana. We work with those guys. We've worked – We have our own dashboards, which are automatically generated. So you don't have to spend time designing them, but there's a range of tools that can render the data, take the PromQL query, which specifies what you want to see and display it in a way that's useful.

[00:12:26] JM: How long do people want to keep their Prometheus data?

[00:12:30] BB: There's kind of a tail off. Basically, the most common use case is you're looking at the data very recently, because you're trying to understand what's happening to your system right now. But for me and I think for a lot of people, it is useful to say, "Well, it was working okay yesterday. Can I go back and compare the very recent data against what it looks like yesterday and see if I can spot something that's different?" Then they might want to compare against a week ago.

Going further back, I mean same principle applies, but maybe another use case comes to the fore. You want to look at the trend or something like – You want to do maybe capacity planning. You can't really do capacity planning based on an hour's data. A month for three months or something like that is really going to give you the long-term trend, and if you're purchasing equipment or thinking about budgeting, or something like that. So it's based on the use case. I think the people that created Prometheus were very focused on the here and now, but there's certainly an argument for the use of the data overtime and wanting to store it for longer.

[00:13:38] JM: When people want to read their data in a PromQL query, how fast does that query need to render?

[00:13:47] BB: Well, that's kind of a question about human nature, isn't that? I mean, I would say the goal is to draw dashboards in the blink of an eye in under a second. But I suppose people understand that a lot of data takes time to crunch and they'd wait a few seconds for a

complex query. But it's in that range. I think people are not generally willing to wait like minutes for their dashboard to come back.

[00:14:16] JM: Does the data need to be replicated?

[00:14:18] BB: Well, maybe. I mean, I guess the question is what would happen if you lost that disk?If you had a disk failure where you've written that data. Now you could solve that at the disk hardware level with like a RAID array, or you could think about taking that data and storing it somewhere else.

[00:14:41] JM: We've given some contours for the basics of Prometheus. What are the scalability issues of running a Prometheus cluster?

[00:14:48] BB: Well, actually that word is – That's the kind of the crux of it. There isn't really a concept of a Prometheus cluster. Prometheus is one process that runs completely on itself. That's what makes it so easy to manage. So whatever it's doing has to fit in the memory of the machine that you're running on. All of that compression I talked about, that takes CPU power. So that all has to fit on however many CPU cores you've got. The disk I/O has to be available from one machine.

I mean, there are ways to kind of what they call Federation of Prometheus to cascade, to do roll ups from Prometheus that have full detail to like a less detailed view over multiple of them. That's the idea of federation. But there isn't really a notion of a cluster, and basically one Prometheus as big as one machine that you can get hold off to run it on.

[00:15:45] JM: But if you write a bunch of data and you never do anything with that data, then you end up with a ton of disks that you've horizontally scaled out to hold your data, right?

[00:15:57] BB: Yeah. So typically you can figure Prometheus with a retention time and it will delete data older than that. Yeah, that's kind of your choice how much disk you want to give it. I think somewhere in the tens to hundreds of terabytes it would start getting painful just to manage disk store in that way.

[00:16:17] JM: That brings us to Cortex. What is cortex?

[00:16:20] BB: Yeah. So Cortex is a horizontally scalable time series store built on Prometheus, and Cortex aims to allow you to run it as big as you need to. That's what we mean by horizontally scalable. Each part can be multiplied and extended, and it is also by design highly available. All the parts are replicated. It works with long-term store, stores designed for durability and massive scale. Cortex is also multitenant. We built that for Weaveworks managed service, which is like a hosted Prometheus. So we have many, many customers come to us. We run on their Prometheus effectively. We run their storage, but it's actually Cortex pretending to be multiple Prometheus. Cortex is multitenant from beginning to end.

[00:17:16] JM: We talked about scraping process. So we've gone over the basics of the data ingestion system, but Prometheus has a component called and ingester. Can explain what an ingester is?

[00:17:29] BB: Well, in Cortex, that's the piece that takes the individual sample.

[00:17:34] JM: Is there not an ingester in Prometheus itself?

[00:17:36] BB: Probably, yeah. I mean, same concept. It's taking the individual samples and stacking them up and applying the compression algorithm. So it builds up a data structure, which contains the time series overtime.

[00:17:49] JM: So just to be clear. So an ingester takes in the data that has been scraped and compresses it?

[00:17:57] BB: Yeah.

[00:17:58] JM: Compress it, formats it?

[00:17:59] BB: Prometheus is one process. All of the components are software, different parts of the software, but they're all linked together into one program, and it's sitting there. It's doing the scraping. It's doing the compressing. It's doing the storing. It's doing the serving of queries

altogether. The Cortex is broken out into microservices that each one does a little piece of the job, and that's how we can scale it by multiplying those services.

Yeah. Ingest, I mean, as a concept, is receiving the data and handling it. What we need to do fundamentally with the data is apply that Gorilla compression that I mentioned earlier, and once we got a certain amount of data in-memory, we want to flush it off to the long-term store. So that pattern is pretty much the same inside Prometheus as it is inside Cortex. They're engineered differently for different goals.

[SPONSOR MESSAGE]

[00:19:05] 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]

[00:21:12] JM: So as we started to talk about scaling Prometheus, what are we talking about scaling? Are we talking about scaling Prometheus to handle more things that it's scraping or are we just storing more data for a longer period of time? What exactly are we scaling when we're talking about that?

[00:21:31] BB: Yeah, any of those dimensions, more things in more detail stored over a longer time. You might want to scale any of those dimensions. I think Prometheus – Prometheus is very efficient. One Prometheus instance can scrape many things without really taking a lot of resource, but the compression and storage and serving queries adds to that burden. So you do reach a point where you just can't scrape anymore with one Prometheus.

Yes, scaling, if you imagine, you could have one Prometheus scraping a hundred services receiving million time series every 15 seconds. Usually, that have worked great. But there are quite a lot of people who operate a bigger scale than that. As you get bigger, you run a room and that's kind of why we built Cortex, because you can just carry on scaling one instance of Cortex to as big as you like.

[00:22:28] JM: Got it. So if you're some gigantic Kubernetes cluster, like one of the ones the jd.com runs and you want to monitor it with Prometheus, some different patterns you might take would be to – You could stand up multiple Prometheus processes and – You could just have the them divide and conquer across the cluster, or you could use Cortex? Are those alternatives?

[00:22:52] BB: Yeah, divide and conquer. You need to figure out some way of dividing it down, and I think for most people the scale problem is maybe not so much one gigantic cluster, but they have many teams. So they give each one their own Prometheus, and then they have the problem of how do I get a global view of all of my infrastructure, of all of my resource usage.

Yeah, you're right. There are people who have single clusters and they have another problem, which is it may not be easy to see a division between one thing and another. Some people divide apps versus infrastructure or one data center versus another data center. You got to find

those divisions if you're working that way, but Prometheus doesn't have a concept of automatically splitting up the data, sharding the data. You have to manually make that choice about where the divisions are.

[00:23:45] JM: So with that said, how are you scaling Prometheus? Can we think of this as a sharded version of Prometheus?

[00:23:54] BB: Yeah, exactly. Yeah, we take the basic components, we break them up into microservices so we can run multiples of them. We do an automatic sharding process, which is essentially based on hashing the labels in the time series. So we get a fairly even distribution of the incoming time series to our ingester processes. For a store, we don't just write to disk. We use a distributed database. We have number of backends in Cortex. Cassandra is one. DynamoDB is another. Google Big Table is another. So we use one of those stores, which are themselves massively scalable, to hold the compressed data and also to index it so we can find it again quickly.

[00:24:41] JM: So let me just try to see if I have it right. So in the normal Prometheus, let's say I'm just a small company. I've got 50 services. It's not too intense, and Prometheus is monitoring this whole thing. I've got a single Prometheus process that's periodically every 15 seconds or so scanning my cluster, scraping the metrics that I'm asking for based on labels and storing it in a file in a time series database. If on the other hand I'm running a gigantic cluster, then I need to scale my Prometheus somehow. Maybe I could just run my own Prometheus multi-process, multi-Prometheus divide and conquer strategy as we were talking about and maybe write to separate files or find some way to merge the files into the same thing to get some kind of global view.

But what cortex does is it just takes care of all of that for you by just saying, "Okay, we are going to scrape your entire cluster or scrape all of your clusters, scrape multiple clusters, and hide the details of that unified view for you."

[00:25:51] BB: Right, and we can effectively run a Prometheus instance as big as you like, because it's really Cortex. It's really automatically sharded across many processes where we speak the same API, but it's done differently internally.

I will say the scraping part actually, we still use regular Prometheus for that. So you may still run 5 or 10 or however many you need to Prometheus to do the original scraping, but because we're not using them for any queries, we're not using the storage feature, those are pretty lightweight. They just scrape the metrics and they send them off over an API that was created to do this kind of thing to use a different storage backend with Prometheus.

[00:26:35] JM: So if there is a high-volume of data coming through Prometheus, there's this ingester part that we've discussed where this is the compression component, right? You said the ingesture does the compression. Is there a bottleneck there if you're just having – If we're just talking about the single Prometheus deployment?

[00:26:54] BB: Well, yeah. It certainly takes a bit of CPU to do the compression. I'm not sure I would characterize it as a bottleneck. I mean, all of the different parts take a certain amount of effort to scraping, reaching out parsing the results that come back, compressing the data and then crating the index on this. All of those things take a certain amount of CPU, and Prometheus itself has been highly optimized over years to make each of those parts pretty efficient. So it's just when you get into the millions or tens of millions, hundreds of millions of time series that you run a power on one box, and that's when you look for some system that lets you shard it across many.

[00:27:36] JM: In the process of building a sharded Prometheus, you build parallelism for this ingestion process. So you have a hash ring that distributes the ingestion of all the data across a series of ingestors, rather than just having one ingester.

[00:27:55] BB: That's right.

[00:27:55] JM: Can you explain why that is? Explain the motivation for that.

[00:27:58] BB: So that hash ring is a type of distributed hash table. That's a well-worn technique that has been used in distributed systems for a couple of decades. So what we need is first of all consistency. We need to send the same time series to the same ingester every time, because it's building up a time series in-memory and we can't just kind of spray those around in

standard load balancer fashion. So we need consistent hashing, or sharding, which we do by hashing.

Another thing we need is do not want all the components to be continuously interlocking with each other and maybe interrogating a single database to know what to do. So we use this data structure distributed hash table, whereby all the components get the same view of who should be doing what. But the only time they communicate about that data structures is when you add or subtract the component, an ingester, basically from the system.

So if you're scaling up the system, then you need to update the hash ring, as you said, and everyone, all the components in the system receive that update. They don't have to receive it atomically, but they will get the new version. They'll start operating off the new version. But they each have their own copy of that structure and the distribution, the consistent hashing, consistent distribution of time series to the ingesters is done without any cross component interlock. So that's very important for scaling. You want all your components to operate independently, because that gives you the infinite scalability.

[00:29:38] JM: So the different Prometheus instances, they are each writing to one or more ingesters, right?

[]00:29:47 BB: We have a component called a distributor that sorts that out. So the Prometheus doesn't care who it's talking to. It talks to one distributor component in Cortex. The distributor – For reasons of efficiency, the Prometheus – The scraping Prometheus will send us batches of samples, batches of time series, like a hundred at a time or a thousand at a time if you like. In that batch, we'll probably have to spread them out across all the ingesters. So the distributor takes care of that. The samples come in batched from Prometheus to one distributor. You can have as many distributors as you'd like, because you need to be scalable horizontally without limit.

Based on the hash ring or the distributed hash table, that batch of a hundred samples are split down. Maybe 10 goes to A, 10 goes to B, 10 goes to C, and then the ingesters take care of doing the compression. So it is getting complicated at this part, but that's two parts. The distributor is responsible for receiving the samples and figuring out which ingester they go to.

Distributor is completely stateless. The ingester is billing up time series in-memory. So it's not stateless. It's important that we talk to the right one each time.

[00:31:02] JM: So each ingester is building up time series data in-memory and then periodically batching it out into compressed loads.

[00:31:09] BB: Right. We typically build a few hours into what we call a chunk and ship that off to the store. That'd be like Cassandra or DynamoDB like I mentioned earlier.

[00:31:20] JM: And these compressed chunks, these are – Do they have some readable metadata associated with them?

[00:31:28] BB: Yeah. The metadata is essentially the name of the metric and all the labels, all the label names and values.

[00:31:34] JM: So the chunks gets stored in a database, like Cassandra, DynamoDB, or Big Table. So these are the Dynamo-style masterless eventually consistent databases, NoSQL databases. Why is this class of database a good fit for the Cortex workloads?

[00:31:56] BB: It's a good question. Well, I guess fundamentally, because again it's horizontally scalable, you can continue adding capacity in the data store. Not have a limit on the size of your database. There're no coordination between different parts of the database. Yeah, it fits really well. We throw these chunks at the store. I mean, at Weave Cloud, we're storing many thousands a second of chunks, where each chunk has several hours of data in it and it just works. It's great.

[00:32:31] JM: These chunks are pretty big, so it doesn't actually – It doesn't matter if they're – Well, how do you need to arrange the chunks in the database? Because if a user issues a query to it, I'd imagine the user wants to know everything that happened along the span of time. So you would want to have these events in – Or I should say these time series samples in the database arranged in some fashion that you could query any time spent that you would want to. But if you're writing to this – Sorry.

[00:33:03] BB: We do that in a few ways. The first thing we do is divide the time into buckets, which are by default one day. So in some sense we actually arrange the database dozens of one day databases. So we start with what is the time span you're trying to look at, and that will give you a day or give you multiple days and we'll go look there in the database.

We're building an index. It's a special kind of index, an inverted index. But fundamentally, that lets us go to the right set of data. Once we got the right day, we go to the right name of the metric, like HTTP response time I used as an example earlier. Generally, in PromQL, you're asking about one metric at a time, and that one metric will have maybe tens of thousands of time series on that day. So you might have narrowed that down. You might've given the label for one service that gives us a narrowing there, and the inverted index lets us go straight to the set of time series that match that label.

So now we have a set of time series and what's in the index at that point is a set of the trunk IDs. So the chunks themselves are pretty much opaque. In fact, as an option in Cortex, you can store the chunks in another kind of store, like S3, which is sort of a bucket style. You don't really have any particular organization of the chunks in that sense, but the index is the thing that lets us find the right set of chunks.

At Weaveworks, we have billions and billions of chunks stored up. So we have this index, which lets us find the right set. One query for one hour might involve fetching 50,000 chunks. Probably that's quite a complex case, or maybe there's only a hundred chunks. It doesn't matter to the software. It goes and fetches them from the store. Unpacks the compressed samples that are in those chunks, and then we reuse the Prometheus query engine to do all of those statistical functions, averaging and whatever based on the role samples that we fetched from our store.

[00:35:17] JM: Contrast the lifecycle of a Cortex query with the lifecycle of a normal Prometheus query.

[00:35:23] BB: Well, it starts off – The query in the PromQL language is a string, comes in basically in the same way. In Cortex, we will receive that in any one of a number of a query or processes. In Prometheus, there's just the one process. But once it's received that string,

there's a process of parsing the query. The parser then understands which metrics it's interested in and it calls down into an API.

In regular Prometheus, that API goes to the Prometheus file store. There's a component in Prometheus called TSDB, time series database, which is looking at files on disk. In the Cortex case, we come through the same API and we enter that process of looking up the index. We look up the index. We do a bunch of speaking to the store, to Cassandra, or DynamoDB, or Big Table, whichever one.

In the Cortex case, we fetch back all those little individual chunks and then we hand back to Prometheus, and Prometheus uses an iterator pattern to step through value-by-value. So in the regular Prometheus case, it's stepping through data that is in a file that is memory mapped into the memory of the Prometheus process. In the Cortex case, it's stepping through those chunks that we returned from store.

In each case, it's getting sample-by-sample and it's doing its aggregation, sums, averages, rate over time. Those things that are really powerful in Prometheus, that's identical in both cases. There is one more wrinkle in Cortex we can take. If you do a big query, like a 30-day query, we will actually split that into 31-day queries. Run them all in parallel and stitch them back together again. That's another advantage of the microservice architecture. We split everything up. We know we have scale. We know we can run these things in parallel to a little bit of an extra wrinkle there just at the top layer. Yeah, basically the same pattern with a different store underneath it.

[SPONSOR MESSAGE]

[00:37:36] 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-prem 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]

[00:39:45] JM: I'd like to talk through some of the failure scenarios that can occur in your strategies for recovering. We don't. So let's take a distributor, for example. So the distributor, if I recall, is the kind of the load balancer between these different ingesters.

[00:40:03] BB: That's right, yeah. Yeah. So let's say a distributor crashes or the machine lost power under it. What would happen is the call coming in from the scraping Prometheus would never receive an answer. In the case of a crash, what would happen is it would see its socket close. The sending Prometheus will then retry that call, and if you've arranged your Cortex installation that way, it'll find a different distributor. Distributors are stateless. So the same batch of samples would come into a different distributor and hopefully it doesn't crash, and the call will proceed through to the ingesters.

[00:40:39] JM: What happens if there's a node failure in the query layer?

[00:40:42] BB: Probably the dashboard would show an error. The user would hit F5 or something like that, or most dashboards are on a refresh every 10 seconds or something like that. They would just see some kind of bad result come back from the query. If it was coming straight in as a simple query, that thing I mentioned where we shard like a 30-day query into 31-day queries. We will actually retry at that layer. So if we got 29 results back and one failure, we'd retry that one failure until we have the whole thing and stitch it together. So, yeah, the query part is completely stateless. The request can come back in and we'll just run it again.

[00:41:24] JM: When a user is configuring Cortex, are they manually describing, "Here's how many Prometheus instances there are. Here's how many ingesters I want," or is there any like auto- configuration based –

[00:41:38] BB: Yeah. I mean, the number of scraping Prometheus will be some feature of how your infrastructure is laid out, like hopefully you could get away with one. But if you had like one per data center or something like that, that would govern that arrangement.

The configuration, number of distributors, number of queriers, that could be auto-scaled. In Kubernetes with a horizontal called pod auto-scaler. I'm not actually aware of anyone doing that, but all of the stateless components of Cortex, that should work. The place where you need to be little bit careful is the ingesters, because they're holding a lot of data in-memory to do the compression, and scaling up would not really be a problem. But scaling down, you need to make sure you flushed everything to the store before you shut down the ingesters you're taking away.

So I wouldn't automate that process, because we don't really have enough kind of management automation around that at the moment. But it tends to be – The load tends to be pretty consistent overtime, because where it's coming from is scraping and you tend to have the same series going on for days and weeks. At least that's what we see.

I guess because we're running a hosted service, we have a variety of customers, they kind of average out. So we see a pretty consistent load and we don't actually scale that up and down very often. We do scale up as we get more customers.

[00:43:14] JM: One thing I am still a little bit confused on is if you have a single Prometheus instance, what are the circumstances where you get into a situation where you say, “I need scalable Prometheus. My Prometheus is starting to fall over.” What are the scalability bottlenecks that we’re hitting that we need to go and migrate to Cortex?

[00:43:34] BB: I think a very common one is a crash, because it's out of memory, which is very sad, because it is painful. Because Prometheus is designed as one process that does everything, when it crashes you lose everything. So the scraping stops the queries that were in flight all fail and the data that was being compressed is – Whatever was in memory at that point is lost.

So why would you run out of memory? Well, the scraping part as I was just saying can be fairly consistent overtime, but the querying is very dynamic, because people, maybe they just have a set of dashboards that they never look at and then all of a sudden there's an incident and everyone's looking at the dashboard, everyone’s hitting refresh.

So the load on the query side can be very variable. In the way Prometheus works is one process in one memory space. Let's say your machine has 64 gigs of memory. If you use it all up with a bunch of people doing queries, the program will fall over. It’s very sad. Yeah, that's a typical thing, program falling over because it’s out of memory.

I could imagine another symptom would be it’s slowing down, because it can’t keep up with scraping. So instead of a sample every 15 seconds, you’re getting a sample every 20 seconds, or every 30 seconds, because it's just not keeping up. Yeah, I mean most mostly when I talk to people who are in this situation, the symptom they see is it actually falling over.

[00:45:08] JM: So when you say running out of memory, that's because you're doing these scrapes and you're not batching them out to the ingester fast enough one?

[00:45:17] BB: No. I think in the Prometheus case, it would be the query side that causes –

[00:45:21] JM: The query side.

[00:45:21] BB: That causes an out of memory, because that's the thing that's very variable.

[00:45:25] JM: Oh! Got it. Oh! So if you have too many users querying at the same time –

[00:45:29] BB: Yeah. So imagine – The situation is my machine has got 64 gigs of RAM. My Prometheus normally runs at 50 gigs of RAM, because it's pretty busy. It's doing a lot of work, and 100 people have suddenly come in with a query and it just tips it over the edge. It's just one of these things, that if you run it over the limit, then it stops.

[00:45:54] JM: This has become a problem, because people build lots and lots of dashboards on the same Prometheus system?

[00:46:00] BB: Maybe. Yeah, I guess these things tend to get popular. People are getting a lot of value from the insights. Generally, one dashboard that's refreshing once a minute or something like that, it is not really a huge amount of load. I think the corner cases is when everyone gets excited to look at what's going on. Unfortunately, that can be the very time when you need the data.

So there are standard mitigation to run a pair so that if somehow you managed to kill one of them, then the other one will pick up. But yeah, this is the kind of situation that you hear about and that's what makes people manually shard their data so they don't have one Prometheus doing everything. They have more than one.

[00:46:44] JM: So this is just a case where like you enough operators, people operating against the cluster and doing SRE against this cluster that they're just issuing ad hoc queries and they take down the cluster and it's the same time where you actually need the data. I can see where that –

[00:47:00] BB: It can happen. Yeah.

[00:47:01] JM: Yeah. Okay. Makes sense. So let's take a step back. This is a project that you've built within Weaveworks. How does apply to Weaveworks and the companies you're working with?

[00:47:12] BB: Well, we built it because we wanted to offer Prometheus as a service. We run this service called Weave Cloud and people can sign up and they got a range of features to help them operate their Kubernetes cluster, and those include continuous deployment, the GitOps feature that I think you were talking to my colleagues about. It includes kind of dynamic exploration feature in the product and it includes the time series metrics and dashboards and alerting and all the good stuff. We find that bringing those things together is really powerful for instance, because we are handling the software deployments, we can mark all the dashboard when the deployment took place.

Quite often, you can see a very marked difference. There was a bug in that new version or it's just less efficient than the previous version and everything has slowed down, or there was a bug that meant the error rate jumped to 30% or something like that. Having that data together is a real feature of what we do.

We built Cortex, number one, because we did not want to provision regular Prometheus for every new customer. That would have been one way to go. Maybe run it in a pod or something like that and come up with some disk storage, and every time a customer signs up to our service, we go through that provisioning process and everyone's got their own Prometheus. We didn't like that idea. It sounded like a lot of ops work we didn't want to do. So we built a multitenant scalable system. So we run one Cortex for all of our customers, and they can sign up any time, middle of the night. They can start up a new instance, and Cortex just starts accepting their data.

[00:49:03] JM: Okay. So this could actually be quite useful to any cloud provider that wants to do a Prometheus as a service.

[00:49:09] BB: Oh, yeah! There are multiple that have come on board with the Cortex project. Some of the individual authors, shout out to Tom Wilkie, pretty much his brainchild to begin with.

He's at Grafana Labs now. So they provide Prometheus as a service. I could name Aspen mesh, Platform9. I apologize to the ones I'm forgetting.

Yeah, there's a bunch of people doing just that. It's an open-source project. We have a real community going. We have contributions coming from multiple different companies. Also people running like an internal service provision. So I could call out the EA, the games company, Electronic Arts. They need to monitor their backend so your favorite game is available, and they run Cortex. They have maintainers of Cortex on stuff. So internal service provision and external service provision are two very much active use cases.

[00:50:13] JM: Yeah. So in the internal case, all the different teams at EA can spin up on their own Prometheus instance, but the internal operator, because EA probably runs their own servers. But the internal operator has their own headaches to worry about and they can reduce those headaches by just having a single storage system that all of those Prometheus instances are running against.

[00:50:39] BB: Yeah. It's essentially the same problem at that scale. Instead of provisioning multiple Prometheus servers with multiple sets of disk storage, you can run one big Cortex.

[00:50:49] JM: What are some other approaches to scaling Prometheus?

[00:50:52] BB: If we widen it out to scaling time series databases, then there's a bunch of things in that space. The people who have started with Prometheus and built a solution based on that, the notable one is the Thanos Project. Thanos was created with a focus on long-term storage and better availability of Prometheus. Cortex was started with a focus on multitenant. So they kind of started in different places, but there a lot of similarities between the projects and it is interesting to see how that develops.

We all talk to each other. The maintainers of Prometheus, the maintainers of Cortex and the maintainers of Thanos kind of overlap in various ways, and yeah, we all talk to each other.

[00:51:40] JM: Bryan Boreham, thank you for coming on Software Engineering Daily. It's been great talking to you.

[00:51:43] BB: Thank you very much.

[END OF INTERVIEW]

[00:51:47] JM: HPE OneView is a foundation for building a software-defined data center. HPE OneView integrates compute, storage and networking resources across your data center and leverages a unified API to enable IT to manage infrastructure as code. Deploy infrastructure faster. Simplify lifecycle maintenance for your servers. Give IT the ability to deliver infrastructure to developers as a service, like the public cloud.

Go to softwareengineeringdaily.com/HPE to learn about how HPE OneView can improve your infrastructure operations. HPE OneView has easy integrations with Terraform, Kubernetes, Docker and more than 30 other infrastructure management tools. HPE OneView was recently named as CRN's Enterprise Software Product of the Year. To learn more about how HPE OneView can help you simplify your hybrid operations, go to softwareengineering daily.com/HPE to learn more and support Software Engineering Daily.

Thanks to HPE for being a sponsor of Software Engineering Daily. We appreciate the support.

[END]