EPISODE 1047

[INTRODUCTION]

[00:00:00] JM: Ceph is a storage system that can be used for provisioning object storage, block storage and file storage. These storage primitives can be used as the underlying medium for databases, queuing systems and bucket storage. Ceph is used in circumstances where the developer may not want to use public cloud resources like Amazon S3. As an example, consider telecom infrastructure. Telecom companies that have their own data centers need software layers, which make it simpler for the operators and developers that are working with that infrastructure to spin up databases and other abstractions with the same easy experience that's provided by a cloud provider such as AWS.

Sage Weil has been a core developer on Ceph since 2005, and the company that he helped start around Ceph sold to Red Hat for $175 million. Sage joins the show to talk about the engineering behind Ceph and his time spent developing companies. I had a great time talking to Sage. You can find all of our old episodes about open source distributed systems projects by going to softwaredaily.com. We've got more than 1,500 episodes about these different subjects, and we also have a new feature where you can write about some of these different topics. If you're interested in having an interactive learning process, you can go to softwaredaily.com/write.

[SPONSOR MESSAGE]

[00:01:37] JM: When you spend your spare time learning, you can accelerate your career. O'Reilly lets you learn through high-quality books, videos, courses and interactive experiences. O'Reilly content has been built over decades. They're a trusted source of effective technology education. If you're an individual leveling up on your own, you can use O'Reilly to chart a course for your career goals. If you manage a team or a company, you can get access to O'Reilly's career development resources for your whole organization.

Go to softwareengineeringdaily.com/oreilly to explore O'Reilly's e-learning experiences. You can build the skills you need to future-proof your career. Check out softwareengineeringdaily.com/

oreilly, and thank you to O'Reilly for being a partner with Software Engineering Daily for many years now.

[INTERVIEW]

[00:02:39] JM: Sage Weil, welcome to the show.

[00:02:41] SW: Thanks. Thanks for having me.

[00:02:43] JM: You started Ceph in 2005. What were your initial goals with the project?

[00:02:48] SW: 2004. It all happened in grad school. I went to University of California Sta. Cruz and my first year there got involved in this storage research group and was specifically sort of tasked with figuring out how to do distributed metadata management for the sort of petabyte scale storage system they were trying to design. It all started there.

That summer, I did an internship at Lawrence Livermore National Labs, and that was where the first finding code is written. Initially started as a – Well, I wrote a prototype simulator in Java previously just to sort of show that that overall approach to dynamically balancing whether the cross service would work, but then that summer, wrote the first line of C++ code that has turned into Ceph.

The goal of that research group was to create a petabyte scale file system for super-computing applications. The research was funded by Department of Energy, Los Alamos, Livermore and Sandia, and it was really targeting these ridiculous, high-performance computing codes where it's thousands of machines all doing some processing and then writing with the files of the same directory or to the same file or whatever it is and trying to build a file system that could deal with these applications. That was the original motivation.

[00:04:08] JM: Did you realize there would be widespread business applications to distributed file system at the time or were you mostly just concerned with these high-performance computing applications?

[00:04:22] SW: Not initially. I was sort of pretty much sort of focused on that niche application. But before going to grad school, I was involved in a webhost startup, DreamHost. So we've done sort of lots of Linux-based webhosting stuff. We bought a bunch of NetApps and sort of – We've loved our first NetApp, because it centralized all the storages, great to manage, and then by the time we got to 100, it was a huge, huge pain in the butt. I had some appreciation for what sort of the rest of the world needed in terms of storage. Also, that there is this sort of glaring gap in terms of what was available that was open source and where you had to go buy expensive enterprise systems. It was sort of midway through I think the whole grad school process that sort of realized that this was a real need in the open source, open infrastructure community. We're really trying to create something that was both innovative and game-changing within the storage field of itself regardless whether it's open source or not, but also to fill this gap in the open source community.

[00:05:25] JM: Tell me more about the bottlenecks that you ran into at DreamHost.

[00:05:31] SW: I think, there, it was mostly sort of a management nightmare. I think the thing that we're trying to scale was really just management of all these data. If you have thousands of servers and then each one has a disk that is going to fill up if one user uploads a bunch of data, then you have all those stuff where you're trying to move things around and it was just annoying. Then you had the problem where if you lost a disk, then it was gone.

Centralizing all your storage would solve that, but the systems at the time would only scale to be so big. So we're really focused with Ceph on just making it as big as possible. In architecting the overall system, just making sure that there were sort of no single points of failure for reliability and also no sort of glaring scalability bottlenecks, so we'd be able to build it. At the time, a petabyte was a really big number. Today, that's not so impressive anymore, but the same principles hold. If you want to be able to create a very large source system, centralize everything in one pool so that you had better management, and also as a result, smarter data placement and replication and so on. You could have overall higher reliability.

[00:06:38] JM: Ceph is a distributed storage system and that doesn't specify the interface of storing information. Ceph can do object storage, block storage and file storage. Can you give a brief overview of these different types of storage? Maybe the kinds of applications that users need these different types of storage systems for?

[00:07:10] SW: Sure. Ceph actually started as just a file system, file store system, and as we sort of build the whole thing, we realized that it's sort of generalized to creating other – Servicing other APIs as well. File is really the one that people are most comfortable with. That's files and directories. That's what you're used to on your desktop or any random Linux system. The goal is just really to have the storage not on your server, or on your node, but to mount it on some centralized system so it's successful from lots of different nodes. That's file. People are usually using NFS or systems that provide NFS to talk to remote storage. In the Ceph case, we invented our own protocol that was smarter and more efficient and scaled better and all that stuff.

But in building the system, we realized that the way that we architected it, sort of the core of the cluster that manages the replication of data and so on used internally, used an object-based interface. We realized you could build lots of other stuff on top of that besides just a file system. The second piece to come along was a block service called RADOS Block Device. The idea there is just to take sort of a virtual disk, like a one, and you strip it over lots of objects and then you store those objects in RADOS in the sub-cluster.

The idea for this actually came from a community member in like 2008 or 2009. I don't quite remember. They wrote a patch for QEMU, the KVM virtual machine hypervisor that would present a virtual disk or a virtual machine that was stored in a sub-cluster on RADOS. We realized that was a great idea. We ended up rewriting the thing as a library and doing lots of stuff. That eventually became very popular in the open sec community as sort of open sourced cloud computing infrastructure took off. They needed storage for all these virtual machines and Ceph was the perfect thing, because it was also scaled out and open sourced and sort of fit nicely into that gap.

Then the third piece is object storage, which is in this context really means something with an S3 or S3-like API. The idea with object storage is that you don't really have directories or files. The closest thing to a file is an object, but you just sort of dump it in what's called a bucket, which is just a container for lots of objects. The semantics are a little bit different than files.

Whereas with a file, you can resize it, you can append to it, you can update, you could overwrite pieces of it. Objects are sort of targeting more static or immutable data. So if you think things like JPEG images or videos or other sort of content that you produce and you post and it fits there and then maybe eventually delete it or maybe you don't, like you're not really modifying it in place.

Because the update semantics for objects were simpler, you don't have things like rename. You don't have updates in place. It's much easier to build an object storage system that's very scalable, and sort of the limited things you can do with it also make it easier to replicate across multiple sites, and that's things like eventual consistency as supposed to cert blocking. It simplifies a lot of other components by having sort of a simpler storage interface.

[00:10:14] JM: If I'm building a block storage system or a file storage system on top of Ceph, is Ceph using the underlying object storage system in each of those cases? Is the base storage system always object storage?

[00:10:33] SW: Yes, but it's confusing because there's sort of two layers, and we use the word object for both of them. The core of Ceph, the sub-cluster is RADOS, which is basically all the storage devices and the software that makes sure that data is placed and replicated across them. RADOS stands for reliable autonomic distributed object store, and the things that we store in RADOS, we call objects. But they're very different than the objects that you would store in S3. The RADOS objects are smaller. They're a few megabytes in size and you can do lots of things to them. They behave more like files and that you can mutate them and you can truncate, you can punch holes, you can do all sorts of stuff.

RADOS provides this object storage layer. That's sort of the internal interface. Then on top of that, we built a distributed file system that sorts of everything in RADOS. We have the RADOS Block Device, which stores everything in RADOS. Then there's the RADOS Gateway, which gives you an S3-like object interface that's RESTful and has multi-terabyte objects and all the ACLs and permissions and all the stuff that S3 does. All that is backed by RADOS. It's two different layers. Two different types of objects. Sort of confusing to explain to people.

[00:11:41] JM: No. Makes sense. That bottom layer of RADOS object storage layer that you can build these other kinds of abstractions on to of, whether you want to build file storage or you want to build block storage. You want to build a user interface object storage kind of layer, why is RADOS that object storage system that was created for Ceph, why is that a useful base level of abstraction for these other systems?

[00:12:12] SW: I think it captures – It sort of solves enough of the problem that it lets the things that sit on top of it simplify the way that they think about the world. The interface that RADOS gives you is that you have pools of storage. It's a logical collection of data, I guess, and you can put billions, trillions, you can put as many objects as you can store basically inside a pool. RADOS handles all the details about which servers is a go-to. Is it going to be replicated? Where do the replicas go? If there's a failure, read from the other replica. Migrate it. When I add storage, I have to move data around between servers.

All that stuff is hidden by RADOS. So things that are consuming Librados, sort of the abstraction layer, all they have to think about is a pool, which is sort of like an infinitely-sized thing that they can put data in and the objects. As long as you know the name of the object, which is a string, then you can read the data and write the data. Then that's a very simple storage layer and having all the scalability and reliability and stuff solved for you. It's a very simple interface to code something else to.

[00:13:17] JM: It sounds like of reliability is also a potential configuration. Are there systems where you would want to configure your RADOS to not replicate the RADOS objects?

[00:13:32] SW: Yes, and you can do that. You can set the replica count to one. So there's no redundancy. People do that in certain cases. It's not very user-friendly, because when there is a data loss event because a drive fails or whatever, then it's sort of awkward to recover because of the way that RADOS is designed.

If you're sort of a Ceph power user and you understand what's going on and you understand the application using that pool can tolerate data loss, then you can poke the cluster and basically say, "That subset of the data is gone and just re-instantiate it, but empty." Then if your application can deal with that, then you can continue. But it's relatively uncommon for people to actually run in that scenario.

[00:14:15] JM: Got it. Yeah. If you have this distributed object system, or I guess if we're talking about the file system or the block storage system, one of the higher level interfaces, you have to manage both the data and the metadata of the information. What kinds of metadata does your Ceph storage system manage and how does that metadata get managed?

[00:14:41] SW: It's probably easiest to talk about it in terms of block since that's the simplest of the three. In that case, most of what you're storing is all data. You have virtual disks. You're just breaking them into chunks and storing those as objects in a system. The only real metadata is the names of the logical images or volumes that you're storing and properties about them, like when they're created, how big are they? What snapshots have you created on them? Whether they're blocked by a particular client who's accessing them. Things like that. What striping strategy are you using it for that particular volume?

In RBDs case, we just store those as attributes on like one object that just has the metadata about one particular volume. So it's sort of naturally scales and makes use of the existing RADOS infrastructure. In the case of the file system, it's more complicated, because there's a lot of file system metadata. The whole director hierarchy and all the properties around files, NI nodes, and ownership, and quotes and all that stuff. There's a whole different set of servers in Ceph that manage the Ceph metadata hierarchy and coordinate client access to data. Because in the case of file systems, you need to make sure that clients that are operating on the same file of your directory are sort of observing, have a consistent view of what the data and metadata looks like, while at the same time you want to sort of aggressively hand out leases to clients so that they can operate efficiently and they can cache things locally without talking to the server every time. Ceph FS has a pretty complicated protocol between the clients and the metadata server in order to achieve both good caching behavior and also strong consistency so that you always have a clear view, consistent view of what's going on from the client perspective.

But even though they have a dedicated set of daemons that are sort of managing that file system name space, all the actual data is stored back in RADOS. So the file data goes into regular RADOS objects, and the metadata about the file hierarchy goes into a different pool and a different set of RADOS objects. That case actually are using a slightly different interface for objects in RADOS where they're storing key value data inside the sort of logical object container in a system.

[SPONSOR MESSAGE]

[00:16:57] JM: This episode of Software Engineering Daily is sponsored by Datadog. Datadog integrates seamlessly with container technologies like Docker and Kubernetes so you can monitor your entire container cluster in real time. See across all of your servers, containers, apps and services in one place with powerful visualizations, sophisticated alerting, distributed tracing and APM. Now, Datadog has application performance monitoring for Java.

Start monitoring your microservices today with a free trial. As a bonus, Datadog will send you a free t-shirt. You can get both of things by going to softwareengineeringdaily.com/datadog. That's softwareengineeringdaily.com/datadog.

Thank you, Datadog.

[INTERVIEW CONTINUED]

[00:17:49] JM: If we think about a typical deployment of Ceph at a corporation, are there users that are interfacing directly with Ceph or are there storage administrators that are building on applications or building databases on top of Ceph? How are end user applications? What's like the layer between the Ceph cluster and the end user application?

[00:18:16] SW: I think you have the full spectrum. Sort of maybe one of the most common cases, you have like a private cloud infrastructure like OpenStack that's being used by the organization, and the only sort of people who even know that Ceph is there necessarily or operating with Ceph are the people who run the cloud infrastructure. Consumers of the infrastructure are just getting virtual machines with virtual a disk attached. That sort of sits at one end.

At the other end, you have people who are – I don't know. Maybe they're doing like big data analytics, whatever, and they're running, operating the Ceph clusters themselves alongside their Spark or whatever else, and they sort of have a more intimate knowledge of what the Ceph cluster is doing and how it's being used.

In a few cases, we also have users and customers who even code their applications to use Librados directly without going through the file system block or S3 object interface, because they can get better efficiency by sort of reaching directly into the lower level storage interface in Ceph and they have sort of one application that is very large. So the sort of scalability and parallelism that they get makes sense.

[00:19:26] JM: What are those applications? Is that like finance or trading or something?

[00:19:30] SW: Yeah. You see some FSI people doing it. There is like an ISP in Australia that built like a time series database or is it an ISP? It might have been like a power company or something that had like IoT power emitters that were logging data. I don't know. There's a bunch of sort of instances like this where you just have lots and lots of data objects. They're all sort of a regular workload and it makes sense to sort of dump it directly in RADOS. There's a group here in Wisconsin that's associated with NASA that's taking satellite imagery, and they're dumping it directly in Labrados just because it was relatively trivial to [inaudible 00:20:07] buildings and just stick it directly in there and they didn't have to worry about the intervening file system or object layers.

[00:20:15] JM: The example you gave of maybe I'm running a big open stack deployment. Let's say I'm a telecom data center and I've got a big storage cluster of Ceph machines and the interface for application developers who were deploying into that telecom data center, they're just seeing open stack, and open stack is exposing some storage interface, and if I need to set up a database for my application to MySQL database, the MySQL database just provisions interfacing with OpenStack, and OpenStack takes care of the underlying communications with Ceph.

[00:20:59] SW: Right.

[00:21:00] JM: What is that interface? If we're talking about like an interface between OpenStack and Ceph, what does OpenStack request from the underlying Ceph storage cluster?

[00:21:12] SW: OpenStack has a whole project called Cinder whose job is basically to be this intermediary by presenting a generic interface to OpenStack tenants to let you request block storage and to sort of manage the attachment of that block storage to your actual virtual machine. Then the Cinder Project has all these backend plugins by various projects and vendors that interface with the backend storage systems. In Ceph's case, there's a Ceph RBD driver in Cinder that implements all the API hooks and makes calls out to RBD to trigger all the write operations.

There's a similar interface for the Kubernetes community called CSI, container storage interface. That's sort of same that you really create volume, attach volume, delete volume, create snapshot, create a clone from a snapshot, yadi-yada. It's good for these ecosystems because everybody consuming the storage has a simple interface to consume and then the various sorts of something sort of plugin on the backside. But there's always been the sort of advantage to open source storage systems who implement these interfaces, because when you're deploying an open source, scalable infrastructure tool, you don't want to have to go buy something to run alongside it. It's nice to use the open source vision as well. In many cases, Ceph tends to get deployed by default with OpenStack and Kubernetes and so on.

[00:22:39] JM: Of course, there are these multiple underlying physical disks that actually hold the data and these different disks are managed by the Ceph object storage daemon. What are the responsibilities of that daemon process that runs across your underlying disks?

[00:23:02] SW: The OSD, it's called the object storage daemon. We run basically one per disk or SSD in a system, and its job is to be the intermediary for any data that's ever written or read from that disk. On the simplest case, if you want to read some data, request goes to OSD. It pulls it off a disk and it sends it back. Most of its – The complexity there is actually around all the reliability and redundancy. The OSDs know how to talk to other OSDs. They sort of understand what data they're storing and whether they're the primary replica or a secondary replica and where their replicas are stored. So that if you do a write operation, the data goes to the first OSD. It knows that it needs to be forwarded to others. All three have to commit that change to disk, and then before an acknowledge is sent back to the client, or if a disk is added to the cluster, or a node fails or something like that and data needs to move around, then the OSDs understand where the data was and where it's going. So they can, sort of in a peer-to-peer fashion, coordinate the migration of data to its new preferred location in the system.

Then I guess the last piece they do is they understand what the layout of the data on the disk is itself. In the past, there used to just be an XFS file system and objects will be written as files. These days, we sort of own the entire stack down to the block storage. So there's a backend inside the OSD called Blue Store that's responsible for figuring out where to actually put those bytes on a disk and how to read them and how to index them and all that stuff.

[00:24:34] JM: Tell me more about that Blue Store system. What role does Blue Store play?

[00:24:38] SW: It's essentially the file system-like thing that we use to find and manage all the data that's stored on a single device. The responsibility of Blue Store is a little bit different than a file system. With a traditional file system, you have sort of arbitrary files and directories and renames and ownership and permissions and all that stuff. In some sense, the job of the Blue Store is a bit simpler. Because it doesn't have of that, there's sort of a single consumer and there's sort of a flat organization scheme for all the data that's being stored. What's the name of the object? What pool is it in? What shard of that pool does it belong to?

At the same time, it has – There are sort of specific requirements that we have in Ceph that are different from a file system. Because we're replicating updates across multiple servers, we want to make our changes in sort of an atomic, as part of an atomic transaction when they make a change to an object on disk. So [inaudible 00:25:30] the update is. We update the version of the object. We also log a login tree. So we know that this object changed at this particular time in this order.

When all that is pushed down to the device, we have to commit that as a transaction, which is something that file system often have sort of internal to the file systems design, but not something that they expose to people who are consuming the file system. After bending over backwards for many years trying to sort of implement transactions on top of a traditional POSIX file system, we finally gave up and just wrote sort of our own storage layer that had exactly the semantics and behaviors and capabilities that we needed.

That's important because when you have all these IOs, updates are streaming across all these devices and then a device crashes or maybe it restarts and comes back or whatever it is, we want to be able to quickly look at the metadata on the devices and know at what point did they crash and what updates do they have and don't they have without having to like go and scan the entire device to find changes, like you might have to do with that write system or something like that.

[00:26:29] JM: Can you tell me other precautions you have to make in Ceph runtime to account for common failures that can occur during a read or a write? Maybe network failures or other hardware device failures. Just tell me about building the necessary fault tolerance. I realize that's a big question for such a big project.

[00:26:51] SW: There a lot of moving parts, a lot of different pieces of hardware involved. The ultimate goal is to create a reliable storage service that's constructed out of entirely unreliable components. So you have to have a lot of redundancy and you have to have a lot of safety checks. On the network side, all the data that's sent over the wire is sort of sent in messages that have CRC checksum in them in case there's some sort of error in the network or it gets flipped or whatever.

With Blue Store, any data that's ever written to the disk has a CRC that's calculated and stored separately with the metadata. Whenever read data, we can verify the checksum before we trust it at all. Something that we also really want that can get out of sort of traditional local Linux file systems. Then there's sort of the overarching failure model of Ceph itself where we don't always distinguish between if you have a disk that goes bad or a sector that goes bad or the interface between the disk the server goes bad or the server crashes. Sort of all of those things usually would just result in an OSD going down. From the rest of the cluster's perspective, the OSD is offline, I can only read and write from it and actually just rely on the other replicas or trigger a repair or whatever it is.

By sort of making it normal for things to crash in the system and for the system to recover, it makes a lot of the other air handling cases easier to think about because you already have sort of a built-in way to understand failure. The overall complexity of the system goes down and the reliability of the system goes up.

Then the last thing is probably the way that Ceph encourages you to structure the organization of the cluster so that you deal with failures in the best way possible. We use an algorithm called crush for data placement. The basic idea being that if you know the name of the object foo that you're going to read or write and you can do a calculation and that calculation will spit out which servers it should be stored on based on the name of the object and the current state of the cluster. Which OSDs are up and down? Any client can know how to find any piece of data by doing this quick calculation and you don't have to go consult some metadata server to find out where everything is.

But the way that that algorithm describes the structure of the cluster is via a hierarchy. Usually you would have OSDs at the least, individual disks. You'd group those into hosts and then you'd group hosts into racks. You'd group racks into rows. You might group rows into rooms of a data center, whatever, all the way up into some root of the hierarchy.

Then you can write your data placement policy in terms of that hierarchy. You might say I want three replicas in my data all in different hosts, or maybe I want them in different racks, or maybe I want the first two replicas in the first rack, and the third rack and the second rack, or whatever it is." But aligning that data placement policy to the structure of the system and aligning that hierarchy to the way that the physical infrastructure is built, the way that switches are connected or power supplies are hooked up. You can increase the overall reliability so that if you have an entire rack go offline because of a top rack switch or a PDU [inaudible 00:29:56] or whatever it is, you can be confident that you're only going to take out one replica. Every data in the system can continue to function without any interruption until repair is made or you migrate the data somewhere else.

[00:30:08] JM: Now, you both built a hosting company and gone through academic distributed systems work. I find there are a lot of people who have built or worked with distributed systems who have not gone through an academic training. I also feel – My experience with this I a little bit limited. I took this one distributed systems class in college and it was like traumatic for me, because it was just so hard. Just these proofs and things. I was like, "I guess I wasn't cut out to be a software engineering," because that's really how I felt, because I couldn't solve these proofs.

I just wonder, as you have built Ceph through the years, how much does your academic training come into play there, or does it just become an intuitive muscle that plays maybe a small but subtle role in how you create Ceph?

[00:31:13] SW: There might be some of that, but it's not something that you're really aware of, I guess, if it is intuitive. I think that the thing that struck me about going through the storage research graduate track was that it's the PhD. It's called computer science, but it's not science in the sense of other hard sciences where you're studying something that already exists or you're writing – Even proofs, like wouldn't do a lot of proofs or anything. It's a really a systems discipline where it feels to me more like a craft.

The thing that I remember about grad school was we would take these seminar classes where we would read papers. Like every session, we would read like one or two papers about some system that somebody designed. Then in class, we would just discuss it aside about what's interesting. Why it worked? Why it doesn't work? How it compared to other systems? Mostly it was just learning about everything that came before and how they built things and all these like sort of clever ideas that people used to sort of advance the state-of-the-art.

Then when it comes time to actually build something, you're sort of drawing on this [inaudible 00:32:21] menu, whatever, of all these different tricks in your pocket and figuring out which ones sort of help you on this particular situation and which ones don't. You build a prototype and then you have to do some evaluation, which is like run some performance tests and try to demonstrate that they solve a scalability problem or a throughout problem or whatever it is.

I think my main regret after leaving grad school was that I haven't really had time or maybe the motivation to like continue reading the literature to read about new systems. But I still find that I'm frequently are referencing in my own thinking these papers that I read 15 years ago now, because it's the same bag of tricks that everybody is using in all these systems, whether it's

Ceph, whether it's Kubernetes, whether it's IPFS or whatever it is. All these scalable systems are sort of drawing on a lot of the same concepts.

[00:33:16] JM: The systems question seem to be a little bit different these days. At least the ones that need to be answered by people solving business problems. It's less a matter of like am I going to lose data here and more a question of like, "Oh, should I use Redis and have like a faster access, or should I use some distributed NoSQL thing, or which distributed NoSQL thing should I use? What are the subtle tradeoffs between these things?"

It's more about like memorizing the little check boxes of like this one has this attribute, versus this one has this attribute. That matches up with my problem rather than knowing like what is the – Like do you use two-phase commit , or three-phase commit, or Paxos, or Raft, or whatever else you might have meant by your bag of tricks.

[00:34:10] SW: Right. Exactly. I think for the consumer, it really comes down what are the semantics out of the system that you really need? What consistency and what performance? What kind of data are you storing and how will that data mutate or not mutate overtime? That really determines what type of system is sort of the best choice for that particular application.

[00:34:28] JM: You started DreamHost when you were like 18? 18 or 19?

[00:34:35] SW: Sort of. Yeah, I was one of four founders for DreamHost. Actually, joined after the other three had already started the company, but at that point, it wasn't DreamHost. It was like a web design consortium thing called New Dream Network. DreamHost was launched I think in 1997 or 1998 when we realized that it was really easy to just charge people money for an account on our server and make money. I worked on that until 2003, I guess, when I went back to grad school.

[00:35:05] JM: Eventually, DreamHost spun out a company based on Ceph, though, right? There was some kind of parallel track where like you were running DreamHost but you were also doing your PhD and you're also like applying your PhD stuff to DreamHost or something?

[00:35:21] SW: Yeah. When I finished my PhD, we had open sourced Ceph when we sort of published the first paper on it. At that point, it was clear to me that like this was filling a big gap in the open source community. So I wanted to keep working on it. The most expedient way to do that was just to go back to DreamHost and continue packing on this project. Hopefully it was something that DreamHost would be able to use. I could just use existing infrastructure and had a whole – All the access to the data center and all the hard drives and whatever else to continue to working on it.

We hired a couple of people over the next couple of years, I guess. So, by around 2011, we decided that in order for the project really to become successful and get adaption, what it needed was a company that was standing behind it to offer enterprise support. Until that happen, nobody would really trust it. We spun out Inktank. Started hiring like crazy. Got some convertible debt from a number of different sources and Inktank ran for I guess two and a half years before we were bought by Red Hat in 2014 or 2015.

[00:36:31] JM: Well, this is kind of a random question, but why in that situation when you're spinning out, you're spinning an infrastructure company out of a hosting company, why does it make sense to raise convertible debt?

[00:36:43] SW: It was just the easiest thing. DreamHost just gave us a convertible note, because we didn't know how to value the company and we just needed money to grow. We went out for I guess twice to raise a venture capital. In both cases, we were unsuccessful. The first time, we got a strategic investor came along and gave us another convertible note to sort of get us through another year. Then the second time around, we were about to raise our first actual venture round when Red Hat came along and was willing to just buy us.

[00:37:16] JM: Amazing.

[00:37:16] SW: Yeah. We got lucky, I guess, in that sense, or unlucky. I don't know. I got to do all the like soul sucking pitches to venture capitalists and then didn't actually have to take or get to take any other money. I'm not sure which way it goes, but yeah.

[00:37:32] JM: When Inktank was acquired by Red Hat, was this around the time that OpenStack was really popular? Like those kinds of OpenStack installations were kind of the – Like that was the Kubernetes of its day?

[00:37:47] SW: Exactly. Yeah. That was really why they bought us. We were the dominant storage system in the open sec community and there was no realistic way that Red Hat was going to – Red Hat had bought Gluster a couple of years before and was sort of trying to push Gluster for open stack, and it just wasn't working. Ceph also did object storage and they didn't have an object storage solution and so on. It was really I think [inaudible 00:38:12] OpenStack what's really driving that whole acquisition.

[00:38:15] JM: I do always hear these two storage systems in the same sense, Ceph and Gluster. Were there some key design differences between the two?

[00:38:25] SW: Yes. Gluster was sort of – The way it's put together is as sort of an abstraction, a layering sort of a file system API level. You would take all the local file systems and you would have a layer that would basically replicate across two of them and then you have another one that would like shard or fragment up your namespace across and you sort of layer these together in order to build a whole cluster that had redundancy and so on.

But it was all through a file API, which worked well up to a certain scale, but didn't work when you had a lot of directories across like a hundred servers, and when you do a maketour, it has to go create that same directory in all the different servers. There's sort of some architecture limitations that made it only scale to a certain point.

In contrast, Ceph sort of as its unifying layer was this RADOS object layer that didn't have that same abstraction. So sort of that core piece that handles the replication and scalability and recovery and so on just behaved much better when it's being added or removed and resized and things failing and so on.

[00:39:29] JM: That's amazing. That insight that you had pretty early on that the file system is not the right abstraction as the base layer in contract to the object abstraction, well, the RADOS object abstraction. That was really what made the difference for Ceph.

[00:39:47] SW: Yeah, that's certainly my opinion at least. Yeah, some of the Gluster developers might disagree, but yes.

[00:39:52] JM: Why is it the file abstraction is so much harder to manage than the object abstraction?

[00:40:00] SW: It's more complicated, because you do a lot of things to files.

[00:40:04] JM: It's maybe too complicated for a podcaster.

[00:40:06] SW: Yeah. Well, no. No. No. The problem is that the file interface itself is more complicated, because you can do things to files that you can't do to objects. The biggest one is rename. In Ceph, if you write a RADOS object with name foo, that's it. It's called foo and it goes in a certain place in a cluster and that's fine. If it were a file, you could rename it into a different directory. You can change its name. Because of the way that placement works, we're calculating placement based on the name. The act of renaming an object file, whatever it is, would actually move it to a different new location.

In Gluster, to deal with this, they have this sort of like redirects. When something gets renamed, you sort of use the hash function to figure out where it should go. If it got renamed, then you have to sort of leave these breadcrumbs behind or ahead of you, I guess, so the data doesn't have to move every time you rename things. That's just sort of one example. The other one that Gluster struggled with was the way that the directory hierarchy was managed, because it ended up basically marrying the total system file hierarchy across every node, and then certain files would or wouldn't appear on different nodes because you had this sort of arbitrarily deep hierarchy. There's all these complexity around making sure that parent directories exist and how do you deal with permissions and things like reader have to like query all the different nodes and aggregate the results and so on.

The way that Ceph dealt with this is sort of at an entirely different layer above this. The object layer doesn't have anything like reader and it doesn't have rename, and instead the metadata server handles that in a way that's sort of written specifically for that set of semantics. So it can to it in a more efficient way. Only that file data gets stored in this sort of free-for-all that is the RADOS object store.

[SPONSOR MESSAGE]

[00:41:54] JM: Vettery makes it easier to find a job. If you are listening to this podcast, you are probably serious about software. You are continually learning and updating your skills, which means you are staying competitive in the job market. Vettery is for people like you. Vettery is an online hiring marketplace that connects highly qualified workers with top companies. Workers and companies on the platform are vetted, and this vetting process keeps the whole market high-quality.

Access is exclusive and you can apply to find a job through Vettery by going to vettery.com/sedaily. That's V-E-T-T-E-R-Y.com/sedaily. Once you are accepted to Vettery, you have access to a modern hiring process. You can set preferences for location, experience level, salary requirements and other parameters so that you only get job opportunities that appeal to you. If you have the right skills, you have access to a better hiring process. You have access to Vettery. So check out vetter.com/sedaily and get $300 signup bonus if you accept a job through Vettery. Vettery is changing the way that people hire and the way that people get hired. Check out vetter.com/sedaily an get a $300 signup bonus if you accept a job through Vettery.

Thanks to Vettery for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:43:31] JM: The initial motivation for this episode was I saw a paper from SOSP. I don't remember what that one stands for. It was about lessons from 10 years of Ceph evolution. Can you give a few other pieces of background that you have learned over the years of managing Ceph?

[00:43:55] SW: Sure. I mean, the paper was specifically about the storage backend, so Blue Store was mentioned earlier. Originally, we stored all that data on sort of an XFS file system on each disk and then that evolved to Blue Store. It's interesting, because the history is actually sort of more nuanced than that, because we started out writing a custom file backend called [inaudible 00:44:15] that was sort of an object-based file system. Then it was just more code to maintain. We're trying to make everything else work and btrfs appeared to have sort of all the features that we needed. So we pivoted to btrfs, and btrfs didn't stabilize, so we used XFS. Then we realized that we really did need our own custom thing and not a file system. So we wrote Blue Store. It was sort of back to the beginning again.

That's what that paper was about. Really, that using an existing local file system is a very expedient way to get something else the ground quickly, but once you get a little bit further down the road and you hire performance requirements and if the semantics don't quite line up, it's worth the investment to sort of own the entire stack, I guess.

I think that's sort of one blessing. I think the other one, the other big one that comes out is that open source communities are hard maybe. When we first open sourced Ceph, in my mind it was like the greatest thing since life spread. It's such a better design than all the other stuff. People are just going to start using it and submitting code and it's going to be great, and then the reality is that took years to build a user and developer community of people who are actually actively contributing to the project, and it's still sort of an ongoing challenge to cultivate a community of engineers who have time to invest in the project that are paid to do it and are working on it long enough where they can really understand what's a pretty complex system so that they can sort of meaningfully contribute to it. Yeah, open source is hard, but it's fantastic when it works.

[00:45:43] JM: You've really worked on a lot of projects, and it was really interesting reading up on your background, and I was pleasantly surprised to learn that you created the WebRing. This is a classic piece of internet lore. A friend showed me WebRing, it's was like digging through an old record collection. Because it was before my time, and I didn't even really start looking at the web history until I was like 22 or something, and I'm sure a lot of people discovered this in high school or whatever. But explain what WebRing is.

[00:46:23] SW: Right. Right. It's something that you read about in XKCD Comics that are referencing like the big old internet. No. I think the WebRing thing is funny because I'm credited with creating WebRing the same way I'm credited with creating Ceph. But in both cases, it wasn't my idea and it wasn't really something that I created. It was more something that I like managed to build and make success, I guess.

There was a website that I ran across in like 1994, maybe, in high school, that was called Europa. It was the ever-expanding ring of pages or something like that. Basically, the idea was that there's these webpages and there'd be a next site link and it was just link to the next person. Then if you wanted to join the ring, you'd just email one of them and tell them to link to you and you'd link to their next person. Just insert yourself in the list.

I thought that was kind of clever and decided, realized that you can make a CGI script to do it, or maybe somebody told me that you should make a CGI script, and then I read – I can't quite remember. But that's how it started, and then realized that you can have different rings, different topics. You could have ones that are on about cars, about whatever it was. A whole sort of community in what site and so on, built up around it.

Yeah, that was a roller coaster, I have to say, that whole process. Because it was – I started it in high school and was working on it through my first couple of years of college and was learning how to like make something that was on a server that wouldn't crash and would know if things kept scaling and having to deal with all these stuff, and I had to carry a pager and whatever [inaudible 00:47:58]. It was sort of a bizarre college experience, but it was pretty fun. It was pretty fun.

In the end, it sort of – I eventually sold it to GeoCities in Yahoo and that put me in a reasonable financial situation where I could sort of focus on other things that were more exciting. Certainly, I can't complain.

[00:48:17] JM: Do you think it could have become something so much more if you would have kept working on it?

[00:48:23] SW: I don't know. It depends on what the so much more was. I think that at the time, the rings were – They've really clicked with people because you didn't have Google, really. The search engine [inaudible 00:48:33] but they didn't work very well. So if you were trying to find content or community, you had to go through like the Yahoo directory hierarchical way to find stuff. It was just hard to find stuff on the web. WebRings basically allowed to people to create all these little communities of like sites that all related to the same topic and you'd have the ring master who would sort of manage that community. So it was like once you sort of found your little pocket, you could find this whole little sub-world, or whatever.

I think that worked really well for creating community, but it wasn't like a recipe for a .com company to make money. I think that's ultimately what Yahoo discovered. They went through several years of trying to figure out how to monetize it. Could they put ads on these pages? You end up putting ads on like random websites. People didn't really want to do that. They couldn't really figure out how to turn that into revenue.

Ironically, in the end, they actually sold WebRing to one of the engineers who worked on it and they continued to operate it, but as a bunch of smaller sort of entity. I think that still exists today. I'm not even sure. Then when, now, search is [inaudible 00:49:38] you can always find sort of what you want. I don't know that there's really a substitute for that same thing where you get these little sub-mini communities, pockets of related content. But at the time, it's certainly sort of filled a gap in the web space.

[00:49:54] JM: All right. Well, last question, and I've really enjoyed talking to you and enjoyed learning about the various projects you've worked on. After starting these various enterprises that you've started, you've stayed at Red Hat. I wonder, if you weren't at Red hat, what would you be doing? Do you have any other ambitious, crazy distributed systems, or web social networking, or hosting infrastructure company idea or have you dropped the mic while you're on top of your game?

[00:50:27] SW: Yeah. I mean, what I normally tell people when they ask this me this question is that I feel like I'm not done with Ceph yet. It feels like the state-of-the-art storage technology that you reach for should be open source, and yet still people spend billions of dollars a year on these proprietary systems, that if they're better, they shouldn't be, because you should be able to build an open source solution that's better than proprietary one.

I think Ceph has been hugely successful, and so I'm certainly not complaining, but it feels like we're not done yet and not really just walk away. If I am going to be working on Ceph, then Red Hat is the easiest and best place to do it, because they're huge supporters of Ceph. We have tons of engineers working on it. It's a great open source friendly company and a great place to work.

But it's funny that you should ask that question, because I am actually taking a leave of absence from Red Hat starting next week through the end of the year to work on something entirely different, and that is just putting all the Ceph stuff on hold for a little bit in order to see if there's something I can do to help out with the political situation here in the US. I'm going to be working with an organization trying to get voters registered, particularly underrepresented communities, young people, minorities and people who are sort of in states where there's voter suppression and related tactics so see what we can do about that. It will be a little bit of a change of pace.

[00:51:54] JM: That's awesome. I mean, the world could use an open source WebRing also, just to let you know. Facebook – We need the open source. As badly as we need the open source answer to S3 and EBS, we also need the open source WebRing. I do hope you solve voter suppression or alleviate the political problems of the United States also. But just letting you know, you got one prospective social WebRing user here, open source WebRing.

[00:52:22] SW: Cool. Good to know. Thanks.

[00:52:24] JM: Sage, thanks a lot for coming on the show. I'm really inspired by your work. It's quite amazing. Appreciate your time.

[00:52:29] SW: Thanks for having me. It was good talking.

[END OF INTERVIEW]

[00:52:40] JM: Apache Cassandra is an open source distributed database that was first created to meet the scalability and availability needs of Facebook, Amazon and Google. In previous episodes of Software Engineering Daily we have covered Cassandra's architecture and its benefits, and we're happy to have DataStax, the largest contributed to the Cassandra project since day one as a sponsor of Software Engineering Daily.

DataStax provides DataStax Enterprise, a powerful distribution of Cassandra created by the team that has contributed the most to Cassandra. DataStax Enterprise enables teams to develop faster, scale further, achieve operational simplicity, ensure enterprise security and run mixed workloads that work with the latest graph, search and analytics technology all running across hybrid and multi-cloud infrastructure.

More than 400 companies including Cisco, Capital One, and eBay run DataStax to modernize their database infrastructure, improve scalability and security, and deliver on projects such as customer analytics, IoT and e-commerce. To learn more about Apache Cassandra and DataStax Enterprise, go to datastax.com/sedaily. That's DataStax with an X, D-A-T-A-S-T-A-X, @datastax.com/sedaily.

Thank you to DataStax for being a sponsor of Software Engineering Daily. It’s a great honor to have DataStax as a sponsor, and you can go to datastax.com/sedaily to learn more.

[END]