EPISODE 571

[INTRODUCTION]

[0:00:00.3] JM: Google's central system for managing compute resources is called Borg. On Borg, millions of Linux containers process a wide variety of workloads. When a new application is spun up, Borg provides that application with the resources that it needs. Workloads at Google usually fall into one of two distinct categories; long-running application workloads such as Gmail and batch workloads, such as a MapReduce job.

In the early days of Google, the long-lived workloads were scheduled onto a system called babysitter. The batch workloads were scheduled onto a system called global work queue. You can imagine those names being quite apt, because babysitter it's monitoring your long-running applications like a Gmail back-end application. Global work queue sounds like something where jobs are just processed and then taken off of the queue.

Borg was the first cluster manager at Google designed to service both long-running and batch workloads from a single scheduler system. The second cluster manager at Google was Omega. Omega was a project that was created to improve the engineering behind Borg, so Borg is still running. The innovations of Omega improved the efficiency and the architecture of Borg.

More recently, Kubernetes was created as an open-source implementation of the ideas pioneered in Borg and Omega. Google has also built a Kubernetes as a service offering that companies use to run their infrastructure in the same way that Google does. Brian Grant is an engineer at Google who has seen the iteration of all three cluster management systems that have come out of Google.

As a principal software engineer with more than 10 years spent at Google, Brian has so much wisdom to offer around cluster management, scheduling, orchestration. He's the Kubernetes lead architect today and it's really great to have had him on the show. He joins the show to discuss how the workloads at Google have changed over time, and how his perspective on how to build and architect distributed systems has evolved.

Full disclosure, Google is a sponsor of Software Engineering Daily. If you are looking for older episodes about Google infrastructure, we've told many of the stories of Google's early days and some of the more recent days as well. We have all of those episodes in the Software Engineering Daily apps for iOS or Android. We've got lots of episodes about blockchains and distributed systems and tons of other topics. You can listen to all of those episodes in the apps.

You can also find them on softwaredaily.com, where we've got a community of software engineers that are posting projects and discussing the episodes. If you want to become a paid subscriber to Software Engineering Daily, you can hear all of our episodes without ads, you can subscribe at softwaredaily.com. Also all of the code for our apps is open source. If you're looking for an open source community to be a part of, come check us out at github.com/softwareengineeringdaily.

With that, let's get to this awesome episode with Brian Grant.

[SPONSOR MESSAGE]

[0:03:26.2] JM: This episode of Software Engineering Daily is sponsored by Datadog. Datadog integrates seamlessly with more than 200 technologies, including Kubernetes and Docker, so you can monitor your entire container cluster in one place. Datadog's new live container view provides insights into your container's health, resource consumption and deployment in real-time.

Filter to a specific Docker image or drill down by Kubernetes service to get fine-grained visibility into your container infrastructure. Start monitoring your container workload today with a 14-day free trial and Datadog will send you a free t-shirt. Go to softwareengineeringdaily.com/datadog to try it out. That's softwareengineeringdaily.com/datadog to try it out and get a free t-shirt.

Thank you, Datadog.

[INTERVIEW]

[0:04:23.0] JM: I'm here with Brian Grant, Principal Software Engineer at Google. Brian, welcome Software Engineering Daily.

[0:04:28.4] BG: Thank you. I'm happy to be here.

[0:04:30.3] JM: At Google you've worked on three different resource management systems. You've worked on Borg and Omega and Kubernetes. How have the requirements of Google's cluster management evolved over time?

[0:04:45.6] BG: Yeah, so we described a little bit of this in I think a couple of the papers. Google started in the late 90s, which was actually just before the multi-core era. That's when single-core frequency increases was plateauing. The early cluster management systems at Google focused on just finding a machine for an application. There were two systems; one was babysitter, and that was for continuously running services and the other was a global work queue, which was for batch applications.

As Google started to evolve and get more diverse types of workloads that it need to run, and as platforms and our data centers started to increase in number, of course it became clear that the existing find a machine approach wasn't going to be sufficient. Borg was really born out of that when new applications were being created and when more resources were available on each machine to try to create a system that could pack multiple applications into each machine.

Since then that was around 2003, 2004. Since then, Borg has continued to be developed and evolve continuously. Since then, the later efforts came out of more specific needs, which I guess we're going to talk about later.

[0:06:06.8] JM: Yeah. I interviewed Louis Ryan about API infrastructure a while ago. He is now working on Istio at Google, and he talked about a few of the major milestones in how the API traffic changed over time. Namely there was the growth of mobile traffic, there was the growth of the public cloud. Did these kinds of upticks in traffic, did they also affect how cluster management needed to proceed?

[0:06:35.0] BG: I think mobile and Internet of Things affected certain aspects such as networking. For example IPv6 is a necessity for communicating with vast numbers of client devices. For the cluster management itself, I think it was much more driven just by Google's scale more than anything else, so we had a need for larger and larger data centers and larger and larger clusters to run more and more applications as a set of things Google was doing was expanding over time.

Throughout I think the whole first decade of Borg's existence, for example, just that scaling was a constant pressure, exponential scaling. As the number of workloads really started to explode more in more recent years, there became more demand for automating the management of the applications themselves as opposed to just managing the resources. I think that's a big shift that's happened someone concurrently with development of public cloud. but I don't think it's a direct consequence of it.

[0:07:39.3] JM: It sounds there's not any particular key event that sticks out in Google's timeline as having changed the requirements for that core infrastructure. It's more a steady exponential uptick in the demands.

[0:07:53.6] BG: I think so. Yes.

[0:07:54.8] JM: Why have you focused on cluster management? Why not something else machine learning systems, or databases? Why is cluster management been your curiosity?

[0:08:03.5] BG: Prior to cluster management, I worked on high-performance computing and actually compilers, dynamic compilation in particular. There are a couple of interesting ties with cluster management. One is, well first of all, it was just a super important problem to Google. All of Google's businesses depended on this infrastructure working really, really well.

Then the connection with my prior background was that it was a very infrastructure-centric project and that's the type of project that I've always gravitated towards. Actually growing up, I was lucky enough to build a computer as a child. From that point, I was fascinated by understanding how they actually worked and how programs actually ran on them. I've gravitated towards lower level systems, like compilers and operating systems and high-performance computing.

These systems like machine learning systems, or databases, high-performance, high-volume databases, these are systems that ultimately run on top of the type of cluster management systems that you have been working on for the last decade or so. Do you have any interesting stories of deploying these kinds of systems on top of a cluster manager, where because machine learning systems and databases, I think of these as applications that they could put unique pressures on a cluster management system.
Have you ever seen one of these things deployed and have it expose a problem with the cluster manager that you didn't previously see with just generic services running on top of the cluster manager?

[0:09:39.4] BG: Yeah, definitely. There are actually a number of kinds of applications that have had unique demands. Machine learning, one of the first things that we stumbled on was actually how to expose the various kinds of hardware accelerators, like GPUs in the environment, especially in the early stages.

One of the things I did prior to Google was actually GPGPU, work high-performance computing on GPUs. In the very early stages, they were not shareable for example, which is contrary to one of the driving motivations behind Borg is to be able to share resources on a machine.

Also Borg treats tasks mostly independently, but multi-machine learning workloads needed to pay attention to the actual – the network topology between the instances, in order to get optimal performance, because the whole value proposition of doing these workloads on GPUs for example is to deliver extremely high-performance, but there are a lot of tricky performance, very steep performance cliffs that needed to be avoided.

Those definitely placed new demands. Storage systems, actually all of our storage systems in Google run on Borg, especially the ones that are very, very widely distributed have a number of special requirements, but Borg wasn't really designed to run those types of workloads. Things where the lifecycle management of the underlying infrastructure, for example storage devices becomes intertwined with application level concerns and vice versa were a bit tricky to model inside a Borg.

I'd say there have been several of these. Just another simple example there, they all have this flavor of requiring some special access to devices that was not really anticipated like tape backups. Tape drives only on particular systems, for example. Yeah, they're a bunch of – been a bunch of different examples of that over the years.

[0:11:41.0] JM: Do you find yourself having to talk to hardware teams, or data center teams in order to figure out how to implement this stuff correctly, or does your –

[0:11:50.1] BG: Absolutely. There have been some proposals for new hardware systems that have been changed, because they would be too hard to fit into the board model. We actually engage with our platform teams that are developing new systems at very early stages, to make sure that there are mismatches, like things that would be almost impossible to implement in Borg, for example.

[SPONSOR MESSAGE]

[0:12:22.3] JM: Software workflows are different at every company. Product development, design and engineering teams each see things differently. These different teams need to collaborate with each other, but they also need to be able to be creative and productive on their own terms.

Airtable allows software teams to design their own unique workflows. Airtable enables the creativity and engineering at companies like Tesla, Slack, Airbnb and Medium. Airtable is hiring creative engineers who believe in the importance of open-ended platforms that empower human creativity.

The mission of Airtable is to give everyone the power to create their own software workflows; from magazine editors building out their own content planning systems, to product managers building feature roadmaps, to managers managing livestock and inventory. Teams at companies like Conde Nast, Airbnb and WeWork can build their own custom database applications with the ease of using a spreadsheet.

If you haven't used Airtable before, try it out. If you have used it, you will understand why it is so popular. I'm sure you have a workflow that would be easier to manage if it were on Airtable. It's easy to get started with Airtable, but as you get more experience with it, you will see how flexible and powerful it is.

Check out jobs at Airtable by going to airtable.com/sedaily. Airtable is a uniquely challenging product to build, and they are looking for creative front-end and back-end engineers to design systems on first principles, like a real-time sync layer, collaborative undo model, formulas engine, visual revision history and more.

On the outside, you'll build user interfaces that are elegant and highly customizable that encourage exploration and that earn the trust of users through intuitive thoughtful interactions. Learn more about Airtable opportunities at airtable.com/sedaily.

Thanks to Airtable for being a new sponsor of Software Engineering Daily and for building an innovative new product that enables all kinds of industries to be more creative.

[INTERVIEW CONTINUED]

[0:14:42.2] JM: As you have alluded to the first unified container management system at Google was Borg, what was going on at Google around the time Borg was created? Can you set some more context for just what was happening when the first, when a widespread cluster management system was defined and implemented?

[0:15:05.5] BG: Yeah, I touched on that a little bit earlier on the hardware side. It was the start of exponential number of growth in the number of cores per box, so that was one thing that was happening. On the Google side, it was MapReduce was being developed around that same time. It was clear that the existing global work queue base solution was not going to be adequate, especially for the newer hardware.

Our web search platform was being redesigned, again to take advantage for optimal advantage of the newer hardware. There is these big new systems, applications really that were being developed around the same time as Borg was developed and it has actually made a goal to ship Borg able to run these key new applications.

[0:15:54.6] JM: If I recalled the paper correctly and you touched on this earlier also, but the babysitter and the global work queue, these were the early, early, early systems that were for processing. Babysitter was if you've got some service that you want to deploy and it's a long-running job, it's like you want you want your Gmail service to be up. It's not a periodic bursty workload, it just sits there and serves traffic.

Whereas, if you have something a nightly MapReduce job, this would be something that would go in the global work queue. Is that the right definition for those different schedulers that you had pre Borg?

[0:16:30.7] BG: Yes, that's right.

[0:16:32.3] JM: Borg was the answer to the question. What happens if we just combine all of our scheduling requirements into one system that can handle all of those things, right?

[0:16:44.1] BG: Yes, although I don't know that that was originally the main goal. We wanted to improve utilization of these newer systems that had more resources available. We knew we'd have to carve up the resources and divide them between different applications. I guess there were two – it became clear that there are two reasons why we'd want to address both categories of workloads with the same system.

One, was it was going to be hard enough to build one such system, much less two such systems. Just amortizing the work and complexity and the number of different systems users inside of Google would have to deal with and things like that. The other was we actually use batch jobs to fill unused, or underutilized resources that are reserved by services.

Because there are different or more relaxed requirements with respect to things like latency and availability of the batch tasks, for example they can be killed and restarted with less user-facing impact, or no user-facing impact. We actually run those workloads at lower priority than the user-facing production service workloads, and were able to push utilization higher by filling the gaps effectively with those workloads.

[0:17:59.5] JM: Borg was created at a time when Google had invested significantly in hardware that did not support virtualization. I think this was in one of the papers.

[0:18:11.5] BG: I think no hardware supported virtualization when Google started.

[0:18:15.1] JM: Right. Okay, fair enough. Of course, in today's Google infrastructure, the containers – well, do the containers in Borg run on VMs or directly on both host machine? Or maybe you could talk a little bit on about Google's evolution towards virtualization, purchases of virtualization supporting hardware and deprecating that old hardware, what that trial and tribulation looked like?

[0:18:38.8] BG: Well, in general Google updates its hardware as it needs to in order to provide significant new value, or lower cost, or new capabilities, or things like that. Virtualization I think mostly did not come into equation when we replaced all the non-virtualization-friendly hardware actually.

The development of key kernel features that support containers like C groups were actually motivated by Borg. The need for stronger resource and performance isolation of the workloads we had running on bare metal effectively in Borg. I think virtualization doesn't actually fully solve that problem. If you need to partition the use of CPUs, or memory especially at a fractional CPU level, virtualization alone doesn't get there.

We actually invested a lot in the kernel mechanisms, including those that are underpinnings of all kinds of containers today for Linux, in order to – so that we could pack applications more densely in Borg. We do have VMs running on Borg, like Google compute engine runs on Borg. Those are VMs onboard, but Borg jobs do not run on VMs.

[0:19:52.1] JM: Okay. I guess, I learned something new today. You really don't even need to have VMs running within your organization if containers do the job, unless you have a cloud service that serves customers that require VMs.

[0:20:11.2] BG: Right. There are some special concerns, like security concerns about untrusted applications and cloud customer applications would be one example of that, but we developed a variety of sandboxing technology over the years to address those.

[0:20:27.6] JM: Yeah. I think about that even on the client-facing side, if you think about a Chromebook. I just think of this very lightweight sandboxed client side browser tab that's probably interfacing with a server-side container, and I don't see where a VM would need to fit into that equation.

Borg evolved with Omega. Omega is described as an offspring of Borg and there's several papers about Omega. What were the key evolutionary developments that occurred around the time of Omega?

[0:21:04.7] BG: Yeah. Omega came at a time where we were facing challenges with the evolution of Borg. There were a number of new use cases that were hard to support and cluster management scenarios that were becoming fairly complex, and like we described the interaction of all the different systems involved as spaghetti or Rube Goldberg machines or things like that.

We're looking at trying to come up with a cleaner design that was more extensible, that could accommodate steadily increasing number of scenarios and systems that needed to be supported. It was really targeting the Borg control plane, the very lowest levels of the Borg control plane. Not so much the user-facing pieces, but looking at questions like how can we support multiple kinds of schedulers for different sorts of different demands being placed on resources?

That's what EuroSys 2013 paper mostly focused on is the multi-scheduler architecture. Really we were looking for how to come up with a cleaner, more extensible less monolithic design that would make it easier for us to add more functionality over time.

[0:22:17.1] JM: When you have such a wide variety of users that are going to schedule jobs on to Borg, or on to Omega, whatever your cluster scheduler is, what are the different axis of scheduling priority that you want to enable them to specify? Because that sounds like, what you're suggesting here is at least one side of the evolution towards Omega is you've got all these different types of jobs and how can we give all of our developers the right tunable priority for how they want to schedule jobs? Am I hearing you correctly?

[0:22:54.7] BG: Yeah, it wasn't even just a matter of priority. It's more a matter of criteria. Borg actually has a priority and preemption scheme built-in that has arbitrarily fine granularity. You could make decisions in a pretty straightforward way about which individual task should displace some other individual task.

If you needed to change what criteria were used to make a placement decision, or change how the resource reservation was modeled, or change how underlying hardware resources were accessed, or changed how Borg's application level scheduler interacted with other maintenance activities at the system level, or at the hardware level, for example where you might need to model the actual underlying resources in a different way. Those use cases where it was becoming tricky.

[0:23:54.8] JM: The other thing you mentioned was that there was some spaghetti code associated with people who had to interface with Borg sometimes. I don't know the history of the setting up the configuration of your cluster, but I know that with Kubernetes it's a declarative syntax, where you just define in declarative syntax what you want your cluster to look like at any given time. If something gets perturbed, then the cluster will self-heal.

I can imagine different imperative language definitions that would probably be more prone to developing spaghetti code. Is that what happened? Would the previous language definitions for creating your cluster, were they imperative rather than declarative?

[0:24:42.3] BG: Yeah. Just to clarify, the spaghetti came from systems that were trying to do things that weren't directly modeled by the board and API at all. There were kluge's in place to try to interoperate in a reasonable way with Borg. That's where a lot of the spaghetti came from.

As far as the Borg's API, it's described a little bit in the Borg paper in EuroSys 2015, but the primary abstraction was job. Job was a parallel array of tasks, where each task executed as an individual container that was scheduled on to some machine. That model is not fully declarative. It did have mostly create read, update, delete kind of a comparative API. It was fairly restrictive, as somewhat high-level, there was just one API for all types of workloads.

Because the tasks, there was no task API really. Things like update strategies and the like had to be baked in to the business logic of that API. That's a again something that drove a lot of complexity as the API needed to be expanded to run all different kinds of workloads.

[0:25:55.1] JM: A component of any distributed systems architecture, well at least most distributed systems architectures, maybe not blockchains, I guess, but there's often this centralized transaction-oriented store. With Kubernetes, you have ETCD, with previous systems you had Paxos-based stores. I think ETCD is raft-based.

Maybe you could describe the evolution of the transaction-oriented store if there were any mistakes that were made in Borg, or how it evolved towards Omega and eventually Kubernetes?

[0:26:33.8] BG: Yeah. I think the main Borg started with its own transaction log that was not even run as a separate service. It was just embedded into the control plane, and I think from my perspective disadvantages of that were in terms of scenarios, like disaster recovery and such.

Getting direct access to the storage system without the control plane and a healthy state was basically impossible. Splitting that out into a separate service that's actually just focused on maintaining that state decoupled from the business logic, in my opinion, makes it easier to operate.

Having it be centralized instead of decentralized, I think also tends to make it easier to operate. Some people have suggested, well what about peer-to-peer, or other approaches? But then you have to question about well, how do you do things backups of the state, or – also just expands the operational surface area for the number of instances you need to manage and things like that.

If you can scale with a smaller number of instances, that tends to be simpler operationally. Whether it's Raft or Paxos or whatnot doesn't so much matter. Most of those all work similarly enough, at least with additional optimizations on top of them; some leader elected transaction store.

Omega's model was a little bit different than at least ETCDV2 and that it was – it did allow multi-record transactions, atomic transactions across multiple records. That was mostly used to be able to factor resources into more granular records in the transaction store, as opposed to being used to mutate lots of unrelated things together in the same transaction.

I think one thing we changed in Kubernetes compared to Omega was that we didn't allow direct access to the store to all of the control plane components, but basically the only component that interacts with it is our API server. That allows us to impose higher-level semantics on top of the raw store, like authorization policy, validation, translation from one representation to another, things like that.

That is a better model we had found for supporting a larger, more diverse set of control plane components, including components that users may build and want to run on their cluster. Providing direct access to the storage system could be very dangerous, things to do without adequate mechanisms for ensuring that they're not corrupting the state, for example.

[0:29:18.9] JM: The third container management system developed at Google after Borg and Omega was Kubernetes. I've done some interviews with the other early contributors to Kubernetes; Brendan Burns, Craig McLuckie and Joe Beda. When did you get involved with Kubernetes?

[0:29:38.0] BG: Yeah, so I was involved with Kubernetes from very early on. Actually back in 2012, there was a mandate to have our internal infrastructure team start to work together more closely with their cloud teams. That work started then. That was around the same time also that in for Borg and Omega, some of these concepts like pods and labels watch API, things like that we're actually being developed for our internal systems.

Then the next year 2013 is when Docker was starting to become better known. That's when Craig, Brendan and Joe started to come up with an idea to build a project around Docker. It came out of that marriage of taking the expertise from our internal infrastructure systems and we have several engineers who worked on various parts of Borg, working on Kubernetes today here at Google still, and working with our cloud teams to elevate the abstraction above VMs to that container, application and service level abstraction, where we can provide portability, more automation and more the kinds of capabilities that we have internally.

[0:30:53.2] JM: As Kubernetes was getting off the ground, you saw the other container orchestrators cropping up outside of Google, the open source ones like Docker Swarm and Mesos. Did the external evolution of those different container orchestration frameworks, did those serve as any inspiration for what you were building internally with Kubernetes?

[0:31:20.6] BG: In a way. 2013 when the actual coding on the project started; there weren't a lot of other systems of that nature. There was no Docker Swarm. In fact, even what was announced at the same DockerCon in 2014 when we launched Kubernetes, that was Lip Swarm, which was basically a library, a simple library client that could start and kill containers on multiple hosts if you told it which containers you wanted to run a multiple host is a fairly simple thing.

That's one thing we saw a lot of people doing is these very simple orchestrators, they would effectively be automating scripts that you would write by hand to do this on a handful of machines, in a very imperative and brittle way.

We saw a big window of opportunity, where we had a very clear idea of what was possible and where we saw the space going. We did consider Mesos. Actually, some of the folks who worked on Mesos interned at Google, and that's where one of the Omega papers came from, for example. They weren't yet targeting Docker, and we felt like some of the design choices that had been made weren't a perfect fit for what we wanted to do, especially for running in a public cloud environment, where we felt the additional resource management layer wasn't strictly necessary.

We decided to start from scratch, which had also the benefit of being simpler and enabling us to choose Go as the language, for example, which others in the space, like Docker and ETCD had chosen. There is some evidence that developers in the space seem to like it. It gave us a clean slate, where we could try out a bunch of these ideas that we had been developing internally over the previous few years.

[SPONSOR MESSAGE]

[0:33:16.8] JM: The octopus, a sea creature known for its intelligence and flexibility. Octopus Deploy, a friendly deployment automation tool for deploying applications like .NET apps, Java apps and more. Ask any developer and they'll tell you that it's never fun pushing code at 5 p.m. on a Friday and then crossing your fingers hoping for the best. We've all been there. We've all done that, and that's where Octopus Deploy comes into the picture.

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

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

[INTERVIEW CONTINUED]

[0:34:48.3] JM: Since then, Kubernetes has been widely adopted. The adoption of it has brought a lot of changes to how people are thinking about the architecture of their back-end infrastructure, as well as the discussions around cloud providers and how you should think about your cloud provider strategy.

When I started covering the space, I got the impression that, okay it's great to have a container orchestrator, because it puts a partition around your infrastructure where you could lift and shift it from one cloud provider to another and you wouldn't be locked in. That's true; people can do that, but what I have seen more of in practice, in talking to people is that they want to use Kubernetes as a vessel for multi-cloud.

It's a way of, if you're on one cloud provider and you want to have access to the APIs, or the cost, the lower cost of infrastructure on another cloud, you can set up a Kubernetes cluster and it gives you a home base to define your APIs and interact with other pieces of infrastructure. Now I know you've been at Google for a while and you probably haven't really seen – Google doesn't need to be multi-cloud, because it is a cloud.

I'm sure you've talked to a lot of people. Are you seeing more of the lifting and shifting, or are you seeing more of the multi-cloud? When you're talking to people, how are you seeing their cloud strategies evolve?

[0:36:25.7] BG: Yeah, one thing I'm definitely seeing is people want to take advantage of the capabilities that Kubernetes have for managing their workloads. They want to run all of their workloads on it; databases, data processing, machine learning, CI, whatever it may be to take advantage of those capabilities, and even simple things like a consistent way of specifying their command line, or a consistent way of probing whether their applications are still behaving correctly, doing health checks, those kinds of things getting that consistent operational experience, where they can affect – we do an API query to see what's running and what state it's in, users have really embraced that.

Having that consistent environment be everywhere they want to run is something they naturally then also want. That's definitely one of our goals with Kubernetes was to make it portable and make it ubiquitous. I'm happy to see that it is now. I think that's great for users and it does really enable multi-cloud in a way that wasn't really possible before, because not that you necessarily want to run the same workload across multiple clouds, but you can run workloads in the same way using the same tools using concepts that you're familiar with using the same workflows anywhere that you want to run, whether it's on Google's cloud, or another public cloud, or on-premise.

It might be a private cloud, or it might even bare metal, it might even be your laptop. Recently, a user told me a story where one of the things, capabilities that Kubernetes really unlocked for them was on-demand provisioning of integration tests environments, for example, that previously they had to be set up by hand, potentially had to get repaired by hand and then people have to schedule time to run their tests.

With Kubernetes, they can just use a tool to spin up the application and all of the components it depends on, run their tests and then tear it all down all automatically. It's those kinds of scenarios I think that really resonate to me. Yeah, I do see people using it multi-cloud, but I think the generalization of that is people just want that consistent foundation everywhere.

[0:38:41.9] JM: Yeah, or you see a lot of companies where they have a single QA environment, where it's if you want to do your QA testing, you've got to wait in line to deploy your new version of the software to that QA environment, and then run your tests there and then somebody else gets to run their tests there. As opposed to if your environments are so easy to set up and destroy, because they're defined as just easy Kubernetes commands, then it lowers the barrier to people testing individually in parallel and so on.

[0:39:13.7] BG: Absolutely. You could potentially do something like that with terraform and virtual machines, but a problem you quickly get into you is that terraform doesn't really attempt to abstract away the differences between the different cloud providers. Virtual machine images are not nearly as portable as container images. Really it would have to be handcrafted for each environment you want to run in.

Then if your test environment is different in any way than your production environment, then the benefits that the coverage you're actually getting from that testing is not nearly as good as if the environments actually had matched with higher fidelity.

[0:39:50.8] JM: Do you think Kubernetes is going to last container management system that you'll have to build? Or do you think there's going to be another one?

[0:40:00.1] BG: I think it would be a bold statement to say it would be the last one. In general, what we see is that these things get rebuilt every decade, or two as there are new ideas as requirements change. I'm hoping that Kubernetes will have a lifespan of at least 20 years, and I think that's easily possible.

Linux, I guess is what? Almost 30-years-old, so something in that realm would be good. Yeah, I think to say – to generalize that even a little bit, the more general space I think is just automation of applications, of operations, of infrastructure. I think we'll continue to see innovation in that space of various kinds and we're seeing that with service mesh and SDO, for example.

Where now, there's a whole new category of infrastructure and application level concepts being orchestrated. We're seeing it with functions. As people try to automate everything in order to be able to make life easier for users, but also scale to much higher levels, like we're seeing – we mentioned this explosion in mobile, in IoT on the number of clients before. Now we're seeing this explosion in the management space in data centers and in public clouds.

People are going to need more and more tools and more and more automation in order to automate that without hiring more and more people to manage it. We actually have a policy about that inside of Google, or at least aspirations or a goal. I think we probably talked about it [inaudible 0:41:31.2] e-book, where we try to ensure that our operations, staff scales sub-linearly with the scale of the applications being managed, right? Because if the applications are growing exponentially, you don't want to have to hire an exponentially growing set of people to manage them. Part of the job description for managing applications at Google is also building additional automation, so that we don't need to hire more and more people at an exponential rate.

[0:42:01.8] JM: Has anybody charted that out to see if it's actually sub-linear?

[0:42:06.3] BG: Almost certainly.

[0:42:08.3] JM: Okay. I hope the results were good. Otherwise, perhaps it doesn't bode well for those operating large-scale infrastructure.

[0:42:16.6] BG: Well, definitely we see that users want to offload more and more services onto cloud providers and other Sass providers, for example, right? They want their services managed so they don't have to manage it. It's the same goal, but they're just shifting the problem to the people who are running those services. Yes, people running those services will not be successful if they're doing just as much work as the union of all their customers would have done.

[0:42:44.9] JM: You mentioned Linux's dominance for the last 30 years, or it's arisen to be so dominant. Are there every things in the host OS, in that Linux, or that I guess – yeah, the host OS that the Linux substrate that you question, where you think, "I wish we just had a better operating system." Is there something at the lower level that you ever think could be changed that would be better for what has been built on top of it?

[0:43:14.0] BG: Oh, absolutely. I think there are a couple of different answers to that. One is that Google has been contributing to Linux for quite a long time to build some of those things that are actually needed. I mentioned C groups, for example. Not all operating system concepts and resources are containerized. Some things are still managed per process.

If we can get more of those sorts of things isolated into container level primitives, either C groups or namespaces, I think that will just improve the kinds of management that we can do on top. That's an ongoing process. Even user namespaces haven't completely matured yet and rolled out everywhere. I think that will be an ongoing process.

We still have some things that we proposed to the kernel way back before Docker several years ago, that didn't get merged. Some of those things we may go back and try again, now that a lot of more people are using containers than just Google. On the other side, we're seeing a lot more container-optimized operating systems.

In operating systems, like Red Hat Enterprise, Linux or, Ubuntu users are trained to install a lot of packages directly into the host operating system, to run the kinds of applications that they want to run on those systems. We're seeing movement away from that to much more stripped-down distributions, lighter weight.

In many cases, they don't even have a package manager associated with them. They just have images with based services that can run containers and then applications are expected to bring all of their own dependencies with them. I think that general trend will continue and I think it's a good one. Honestly, I wish I had containers on my Mac. It would make things so much easier. I think we'll see that model really start to take off.

The surface area of what you need to vet in your compliance and security audits, for example, just becomes so much less on the host OS if you just strip away all the things that are only needed by applications, or are only needed for certain administrative actions, things of that nature. You can isolate all those things and then really start to understand what things are needed and why and where.

[0:45:37.5] JM: When you start to talk about running your host, your client operating system, like your MacBook, or your phone in terms of containers, is that to say that you would want to have something like a Kubernetes on your client device?

[0:45:54.5] BG: Well, I don't know that I would need something like Kubernetes per se, to run my laptop. It might be fun, but it's more – since I started using computers, I don't know, I was nine-years-old, one problem has been installing multiple pieces of software on the same system; has always created some problems or some challenges, whether it's DLL conflicts and Windows, or what-have-you.

I think containers provide both application, file, dependency, isolation that is super beneficial to just cleanly divide, keep each application completely independent from the other so you can easily install them and uninstall them and upgrade them and all independently. Also, the resource isolation so that you can manage more cleanly how much resources to devote to different applications, like I would love to be able to just take away resources from my browser in order to do some development, for example.

Another thing that I think containers do can do really well is package up all the dependencies needed for a development environment, right? One pain point for me if I download code and install it and try to develop it, is that I may not have all the tools necessary to build it on my local machine, for example. I can just package all those into a container image, then my job is simpler. I don't need to figure out what version of the compiler do I need, what version of the – which build system do I need, and what version of it do I need, and what environment does it expect and how should pass be set up and all that crazy stuff that every developer has to go through.

[0:47:37.1] JM: I know we're at the end of our time. Just to close off, you've been building these kinds of systems for more than a decade. Do you have any parting thoughts for people in the audience who are building some distributed system, something that involves a scheduler, or large volumes of infrastructure?

I know there's a lot of people out there building blockchain-related applications based on the listener responses from the most recent episodes, but it can or cannot encompass the design of blockchain applications. Just any words of wisdom.

[0:48:11.2] BG: Yeah, actually one thing that came up in a conversation I had earlier today with someone; understand what use case you're optimizing for. Many of these systems have many engineering trade-offs that need to be made. Presumably the reason you're building a new system is that the existing ones didn't address some use case as well as you would like. Be really clear about what use cases you're optimizing for, because you're not going to be able to address all use cases equally well.

I think the other thing, I feel like sometimes it's my job just to chant is pay attention to separation of concerns. One system, one tool doesn't necessarily have to do everything. Sometimes it's better to draw the bounds of scope of your system and define the integration points with other systems, so you don't have to make a set of trade-offs across a much broader space than you intended. You can allow people to make different trade-offs in the systems that they would integrate with yours. I think those are the two main things I would say.

[0:49:16.6] JM: Brian Grant, thanks for coming on Software Engineering Daily.

[0:49:18.9] BG: Thanks. It’s been great. I really enjoyed it.

[END OF INTERVIEW]

[0:49:24.5] JM: Users have come to expect real-time. They crave alerts that their payment is received. They crave little cars zooming around on the map. They crave locking their doors at home when they're not at home. There is no need to reinvent the wheel when it comes to making your app real-time.

PubNub makes it simple, enabling you to build immersive and interactive experiences on the web, on mobile phones, embedded into hardware and any other device connected to the internet. With powerful APIs and a robust global infrastructure, you can stream geo-location data, you can send chat messages, you can turn on sprinklers, or you can rock your baby's crib when they start crying. PubNub literally powers IoT cribs. 70 SDKs for web, mobile, IoT and more means that you can start streaming data in real-time without a ton of compatibility headaches. No need to build your own SDKs from scratch.

Lastly, PubNub includes a ton of other real-time features beyond real-time messaging, like presence for online or offline detection and access manager to thwart trolls and hackers. Go to pubnub.com/sedaily to get started. They offer a generous Sandbox to you that's free forever until your app takes off, that is. Pubnub.com/sedaily, that's P-U-B-N-U-B.com/sedaily.

Thank you PubNub for being a sponsor of Software Engineering Daily.

[END]