EPISODE 597

[INTRODUCTION]

[0:00:00.3] JM: Containers have improved deployments and resource utilization. Kubernetes created a platform to manage those containers and orchestrate them into distributed applications. Today's episode we explore tools that improve the workflow of the application developer who is working with Kubernetes. Tools include Helm, Draft and Brigade. Helm is a package manager for Kubernetes, which allows users to find, share and use software that is built for Kubernetes.

Unit of installation for Helm as a Helm Chart. [inaudible 0:00:31.4] a Home chart can simplify the deployment of a database, a load balancer or continuous integration. Draft is a tool for simplifying the containerization process. Developer runs Draft, a Docker file is created to containerize the application and a Helm Chart is created to enable the application to be easily deployed.

Brigade is a tool for creating and running Kubernetes workflows. It allows for event driven scripting on top of Kubernetes. [inaudible 0:00:58.8] ops continuous integration systems and complex big data pipelines can all be defined with Brigade.

It's an exciting topic of discussion because it's a higher level tool on top of Kubernetes. In dumb ways, it's similar to the serverless on Kubernetes systems that we have covered in the past.

Ralph Squillace is a principal program manager with Microsoft and he works on containers and Linux and cloud products. Ralph joins the show to talk about how developing with containers has changed in the last few years and how it will continue to evolve in the near future. Full disclosure, Ralph works at Microsoft, which is a sponsor of Software Engineering Daily.

[SPONSOR MESSAGE]

[0:01:48.4] JM: Citus Data can scale your PostgreS database horizontally. For many of you, your PostgreS database is the heart of your application. You chose PostgreS because you trust it. After all, PostgreS is battle tested, trustworthy database software, but are you spending more and more time dealing with scalability issues? Citus distributes your data and your queries across multiple nodes. Are your queries getting slow? Citus can parallelize your SQL queries across multiple nodes dramatically speeding them up and giving you much lower latency.

Are you worried about hitting the limits of single node PostgreS and not being able to grow your app or having to spend your time on database infrastructure instead of creating new features for you application? Available as open source as a database as a service and as enterprise software, Citus makes it simple to shard PostgreS. Go to citusdata.com/sedaily to learn more about how Citus transforms PostgreS into a distributed database. That's citusdata.com/sedaily, citusdata.com/sedaily.

Get back the time that you're spending on database operations. Companies like Algolia, Prosperworks and Cisco are all using Citus so they no longer have to worry about scaling their database. Try it yourself at citusdata.com/sedaily. That's citusdata.com/sedaily. Thank you to Citus Data for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[0:03:33.1] JM: Ralph Squillace is a principal program manager at Microsoft. Ralph, welcome to Software Engineering Daily.

[0:03:38.3] RS: Thank you.

[0:03:39.6] JM: It's been a few years since containers have become popular. Containers have improved in a lot of ways and they've helped developers improve their deployment workflows. They’ve improved many other aspects of development, but what are the shortcomings of the development process that still exist for an engineer working with containers today?

[0:04:03.3] RS: Oh, there's still plenty. Naturally, containers give developers and system admins and operators a tremendous amount of flexibility, speed and agility and so forth. Unfortunately, with every step in every technological leap that allows us to do more faster, it turns out the next thing you do is more faster, and that means you need a whole bunch of extra tooling to handle the additional complexity that doesn't exist yet, and that's the first thing. The second thing is that until the tooling helps you do your work in a way that is coherent, you now need to master containment service composition, other things that have nothing to do with software development per se, but really have to do with getting your container of software running somewhere.

Whether you're a developer in a small team, or a startup or whether you're in an organization, those two areas are the focus of work right now because the containment itself makes sense. We understand how to do it. It works really, really well, but those two areas need to catch up.

[0:05:04.8] JM: I think a lot of this will come with improvements in tooling, and you are familiar with a lot of the tools, you work on a lot of the tools in the Kubernetes landscape. Let's talk about some of those tools that the audience may or may not be familiar with. I'd like to start with Helm. Helm is a package manager that allows users to manage groups of resources.

Before we talk about Helm specifically, let's just talk about the idea of a package manager. What do we want out of a package manager? What are the requirements?

[0:05:38.0] RS: Oh boy! There’s a can of worms. Generally speaking, in fact I’d go back and say that the requirements of package management do differ by what you want to do, but generally speaking, you got to be able to bundle up something that is logical. You got to be able to have that deployable in the logical verbs. You got to be able to upgrade it. You got to be able to delete it. You got to be able to do those kinds of things.

Typically, and one of the things that Helm didn't necessarily specify, you’ve got to be able to store those artifacts somewhere in a way that you trust and you can sign them, making sure that you're doing the right thing, have some sort of dependency management, all of those kinds of things are involved.

[0:06:18.1] JM: And then Helm specifically, what does Helm do?

[0:06:20.8] RS: Helm is – The easiest way to explain Helm, I find, for people who aren't familiar with that is that is to take one step back into Kubernetes, and Kubernetes is not actually, at least the way I look at it, it’s not really an application deployment system. It’s a series of objects that implement particular features that you can use by configuring them “correctly”, and correctly here is in quotes.

So if you know which objects enable which features, you can create a manifest in YAML and you can turn on or off and configure various features with certain artifacts like containers that allow them one or more containers to function coherently as a logical unit. But Kubernetes itself doesn't understand the logical unit. It just does what you tell it to do, right?

So Helm then is a way of breaking up those manifests that Kubernetes, the raw Kubernetes manifest into logical chunks, first of all. So a deployment has one file instead of the deployment, the service and pods all being in the same file. A deployment has one file. Ingresses have one file, potentially secrets and so forth. That helps you sort of logically understand the composition of your service or services. So that's the first thing.

The second thing, and we’re talking about a Helm Chart here, which is really sort of the set of these files. The second thing it does is it uses go templating. Although it's possible to plug in any other kind of templating so that the core values that are important to you are in the top level values .yaml file and those values are then reused and inserted in the templates in the lower level files. Ultimately, Helm just converts all those files into a manifest and gives that to Kubernetes and says, “Please make this happen,” but it also stores logical information about your chart, your set of components that you wish to deploy into Kubernetes as release information. So with Helm you can do things like roll back into a different release and roll forward and do upgrades at a logical level rather than at a Kubernetes object level. Now, did that make sense?

[0:08:25.3] JM: It does make sense. So a Helm Chart consists of metadata and Kubernetes resource definitions, I think. You would agree?

[0:08:35.2] RS: Yes, it should not be imagined that Helm Charts are anything different than ultimately Kubernetes objects and metadata about Kubernetes objects. So it's not as if you're doing something different. Helm just turns around, takes the information from the Helm Chart and configures the appropriate objects. That's all it's doing. So that's exactly right.

[0:08:55.3] JM: Helm has three core concepts. There's the chart, which is a recipe that describes the metadata and the resource definitions, which you just said these are the things that a Helm Chart consists of. There's also the values, which are the user supplied configuration. Then there is the release, which is the chart together with the values that you supply. Say a little bit more about these core concepts.

[0:09:27.7] RS: Well, so the chart itself is the package of files in a particular format. You can go to Kubernetes Helm and look at the chart documentation there if you'd like. The core files are in fact what I described earlier. They are the descriptions of the raw Kubernetes objects that will be configured. However, they have Go templating in there into which the values that you set, the user settings as you said, such as things like how many copies of this container do I want to run. Do I want two to be running or do I want seven? Those kinds of things. What is the name of the service? What are the names of the pods and so forth? Those kinds of things, whether ingress is enabled and so forth.

All those kinds of things will be injected into the charts when you invoke Helm and install. So that's what the values portion are. The releases are the logical structure for all of these working together, and so you can – For example, let's take a particular Helm Chart for, say, PostgreSQL. So you can say Helm install PostgreSQL chart and what'll happen is that chart knows how to install PostgreSQL in a Kubernetes cluster wherever it is. It's not dependent on anything else other than having access to a cluster. It works overview kube control.

So that releases a logical entity, and if you delete the release, the release history is still there. So, for example, you could try and install that same release again and it will say, "I'm sorry. We already have one," in which case you just re-created. You just re-created the history. The nice thing about that is it allows you to go and do rollbacks to any particular previous release and you can install multiple releases side-by-side with naming, differences and so forth.

There are wonderful projects out there and companies doing Helm Chart management now to track dependencies and to do tests and automatic deployments. I can mention a few. Bitnami has a bunch of stuff that they've just – They're sort of the Helm Chart Masters in many ways. JFrog, and Codefresh and Weave, many others. So they're a bunch of tools to use to manage the kind of chart complexity. You could imagine, if you think of Helm Charts as being sort of the micro services approach or version of yum or apt or something like that, you're going to need some repositories in some ways to manage what gets install, which doesn't, and those repositories along with Helm allow you to do that.

[0:11:57.5] JM: If I deploy Helm to my Kubernetes cluster, if I've got Kubernetes running and I decide I want this package manager on my Kubernetes cluster, because why wouldn't you? What happens if I deploy Helm? When I deploy it, what does it need to add to my Kubernetes cluster to be able to deploy to it?

[0:12:18.1] RS: Well, Helm actually is a client/server architecture, and so when you deploy Helm, you do a Helm in it, by default what that does is it installs a server-side component called tiller, and the default installation is insecure in the sense that it's global. The assumption was made when Helm was started, Kubernetes didn't have a lot of complex security features such as RBAC and things like this, and even RBAC isn't mostly enabled by default, even though it should be and it's definitely the way to go.

So at the time the two X – So I think we're at 2.9, 2.91, something like that, those versions of Helm install globally, which is great for developers getting their work done and needing to install, say, PostgreSQL or any number of other services very, very rapidly so that they can do development against them. That works perfectly, but it works in a developer environment in which there is no particular threat to the network that you might suspect.

When you get the production, you don't really want to do that. You want to install Helm per namespace and you want to secure it with TLS and configure it to respect RBACK policies and you can do that with the 2X version, but making that easier and turning it on by default is one of the main goals of Helm 3.0, which is just kicked off.

[0:13:36.3] JM: So there's a Helm client, as you said. Again, what are the responsibilities the Helm client?

[0:13:42.0] RS: Will, the Helm client actually just connects and calls and passes payloads to tiller server component right now, and it passes those payloads through kube control. So it's all secured. So it’s a very simple way of going about it. There's also libraries, you can go ahead and invoke the Helm API and build tools on top of Helm that do the work directly with Helm Charts. There are several that do that including things like binoculars and, I think, landscaper and a couple of others. So you can use it either way.

Most developers just use Helm because they just want to get something installed. So it's basically pseudo-app install and all of a sudden they’ve got PostgreS working so they can do their development or something like that. But in a professional environment you are going to need other approaches to make sure things are secured and that the developers get access only to the namespace they really want.

[0:14:33.9] JM: As you said, there is this pod you get called a tiller when you deploy Helm to your Kubernetes cluster. What is the interaction between the Helm client and the tiller?

[0:14:46.8] RS: So tiller has basically two ways of going about it. Tiller has a GRPC endpoint that the Helm client interacts with, and it's basically a client/server. It just goes ahead and connects and passes the payloads with the appropriate verbs. It’s very simple.

[0:15:05.7] JM: Okay. So now that we've given a little bit of an outline of the architecture, when I install a Helm Chart, what is going on across my Kubernetes cluster?

[0:15:16.4] RS: What’s going on across your Kubernetes cluster is that the Helm client, let's say you're going to use the Helm client. The Helm client is going to pass the artifacts to tiller and tiller grinds that out into the Kubernetes manifest. That is, it applies the Go templating to the files that you’d pass it and goes ahead and converts it into native Kubernetes objects and then it calls Kubernetes and says, “Please do these things. Create this service. Open up this ingress. Grab this container. Give me that many copies,” and so forth.

If you're familiar with how Kubernetes works, it should be very easily understandable that it just turns around and gives Kubernetes the same commands you would do with kube control if you are ere doing everything manually. If you're not familiar with Kubernetes, it's basically doing – It’s taking configuration information and telling Kubernetes, “Make this stuff happen.” Again, there are a couple of things that it does that as acting as tiller. So one of the goals, again, of 3.0

is to – Will end up removing the server-side, as I understand the architecture now, so that the work is all done on the client side, and then when you do Helm in 3.0, you'll be acting as you in an RBAC system and that will be one of the big improvements that we're looking forward to.

SPONSOR MESSAGE]

[0:16:40.1] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It's open source and free to use, and GoCD has all the features you need for continuous delivery. Model your deployment pipelines without installing any plug-ins. Use the value stream map to visualize your end-to-end workflow, and if you use Kubernetes, GoCD is a natural fit to add continuous delivery to your project.

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

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

[INTERVIEW CONTINUED]

[0:18:01.5] JM: Let's say I deploy WordPress using Helm. I use WordPress. It has a database. If I wanted to deploy that to Kubernetes, I would get a database with my Helm install. So if I write to that database, if I'd create this WordPress instance and then I create a blog post on my WordPress instance running on Kubernetes and then I do a Helm delete. Let's say I did Helm delete to delete this WordPress instance. Does that clear away all the resources? Does it clear away the database that I've created and written to?

[0:18:38.2] RS: Sure. If you're going to use delete, it absolutely would. So if you think about it like in a micro service environment – I mean, WordPress is a fun example because sort of like the hello world of micro services. You obviously have a web frontend and you have a database in the backend. So there are multiple services involved in that installation, right?

So if you think of the multiple services as "the logical application", they'll be a Helm chart that installs multiple services where one of them is a database, usually MySQL, could be anything, on the backend as a container in your application. If the Helm Chart wants to deploy it in a different form, it can, but the basic default WordPress installation, which is at –If you're new to this, you might look at kubeapps.com, which is a wonderful collection of curated charts to install, things like WordPress, and it's a great place to sort of dork around with Helm and Kubernetes. So in those cases, the logical release, the Helm Chart, actually deploys more than one service. If you delete that chart, you're going to delete all the services related to it.

Because, of course, it's a micro service application, that means if you wish to, you can drop down to kube control and delete one of the services out from underneath the Helm chart. Obviously, your larger logical application of WordPress instance would stop working properly, but the Helm chart is still deployed as a release. Then if you wanted to get rid of everything, you would do a Helm delete and pass the chart and the name – Excuse me, just the name, and that, and then if you want to delete the history, you'd pass –purge, which will get rid of any evidence in the Helm release history that it was deployed in the first place.

[0:20:25.6] JM: Once I've have installed something to my Kubernetes cluster, like if I did install WordPress with Helm, how does Helm keep WordPress up-to-date?

[0:20:39.1] RS: Actually, Helm itself doesn't keep WordPress up to date. It is assumes that the release you gave it is the release you wanted in the cluster. So in that sense, it's much like the sort of standard yums, apps and so forth that install what you want, but it's entirely possible that you can receive notifications or that you can go and check. Again, some of these Helm tools will do that for you and detect changes in shared Helm charts that you might use. For example, the WordPress one.

If you go and use a newer version of the Wordpress Helm Chart, that will be a new release and that you can install both side-by-side version 1 version 1.1 as you see fit since the charts themselves, the logical versions will be different. So you can install many different versions side-by-side if you want it. That's totally part of the way that it works. But you're going to have to get another tool to notify you and/or do automatic upgrading if that’s something that you want.

Some people like that, and especially in the developer space and in the small business space, some things like those kinds of things are used commonly. Large organizations tend to not like surprising updates, since as you get more complex, updates have interesting side effects that are sometimes not predicted. So it really is a choice of how you want to do your operations there.

[0:22:07.8] JM: What are some applications of creating Helm Charts within a company? If I'm running a startup, when would I want to create my own Helm Charts? How would that be useful?

[0:22:19.3] RS: So that's an interesting question. I like it. Most people don't –

[0:22:23.1] JM: Or is that not an application? Is that not something I would want to be doing?

[0:22:27.1] RS: No. I actually like the question. Most people don't ask when you want Helm. So we’ll start with that first and then go into sort of the details. You want Helm when, A, your developers are not terribly familiar with the Kubernetes object yet and you want to start developing against Kubernetes, against other things that can be installed in Kubernetes like, let's say, databases and other applications, right? But Helm will get you started quickly by deploying those for you with public versions of those charts. That's a good example of when you want to do it.

But the other time you want to use Helm is when you have mastered some degree of Kubernetes stuff. Let's imagine you’re building original material now. So you’ve used Helm to install a database and a couple of messaging queue services and various other things that allow you to really focus on the application you want to build. So now you’ve been building it, but you didn’t really use Helm except to get those things up and running real quickly.

So at some point, you get to a level where you think your service is actually running properly and you've been tweaking the knobs and so forth and now what you want to do is allow other people to install it easily and quickly. What you would want to do there is convert your manifests, assuming you were originally tweaking YAML, manifest YAML. Convert your manifest for your service into a Helm Chart and then add the other services as chart dependencies.

So for example, a Helm Chart can specify other charts as dependencies. Once you do that, you can then create a Helm repository in public or in private. Say, if you're in a private environment, a large corporation or governmental environment or whatever you might be true of your case, but whether public or private, you can now give the URL to that repository, Helm Chart repository, to other people and they can do a Helm repo add with the URL of that repository, and now they can Helm Install your work in one command, including the dependent charts for the other services that your service requires in order to run correctly.

[0:24:43.0] JM: So it'd be useful for re-creating complex Kubernetes installations that are already deployed in one environment and you want to deploy them to a different environment, whether that's a staging environment or a sandbox that's just for another developer.

[0:24:59.4] RS: Yes, absolutely. In fact, any environment, because Helm Charts are used in precisely this way to deploy live to production as well because of the relatively robust verbs or rollback, update, upgrade release version differences and so forth. In addition to which the simple ability to do a Helm install, one line install, which pretty much anybody working in Linux or if you're on Mac, like brew install, for example, and things like this, that simplicity is just when you want people to use your software, you need to offer them that simplicity. The problem with kube control, which is not a problem of Kubernetes is just the way it works. The problem with kube control is there's too many magic strings going on. There's no particular logical entity tracking that Helm repositories give you along with the tooling that goes along with them. So Helm becomes useful as soon as you want to make things easy for everybody to move faster.

[0:25:57.4] JM: I want to move on to talking about Draft. Draft is a tool that simplifies the containerization of an application. Explain what draft does and how I can use it.

[0:26:08.2] RS: Okay. So we talk a lot about Helm and manifest. Draft is interesting because it fits another niche in the developer space. We talked a lot about you have to have containers in Helm and Kubernetes needs service composition for those containers and there's a bunch of variables that have to do with how many of them you want, so forth. The problem there is something you asked about in the beginning, which was what are the kind of the difficulties for developers.

One is, really, most development is not done in containers, and this container thing is very, very hot and it’s hot for a good reason, but you got to remember that most people are still not doing containers yet, and most applications are not built to a micro services approach. As we start embracing whether you’re a younger developers trying to move quickly or whether you participate in a larger team that has applications you want to migrate to containers to take advantage of the benefits, you don't really want to spend all your time learning how to build containers, or even worse, learning how to configure Kubernetes. It's not that you shouldn't. You should. Knowledge is good. These are just tools. But it turns out the getting started is every bit as important as it is mastering the toolset.

Draft is a tool that does two things mainly. It's designed to detect the language you’re working in and give you a basic container, a Docker file, container image description, such that you don't have to do what most people do. You don't have to go out and Google the world after you built your app and go, “Oh my God! I'm using F sharp in asp.net core on Linux, and so how do I contain that?” Most people don't actually know, and they just trust some image that they’ve found on Google, and if it works, they think they’ve contained.

Draft helps find a basic best practice for most of the languages and applies that and a Helm Chart for that basic application to get you started right away and then allow you to augment those artifacts and check them in to your GitHub repo properly so that they can be used by everybody and be tracked according to the git commit hash along with your code, and that gives you a high degree of consistency. It lets you start doing container native development right away against Kubernetes, and that's the first thing it does. Lot of words, I know, but it's worth going on and having a look at some of the demo videos and you’d get a feel for how easy that is.

[0:28:38.6] JM: Indeed. Please continue.

[0:28:39.7] RS: Okay. So the problem with that is that those are only basic. That's getting started applications. In hello world, we've got like hello 10 language versions of the hello world applications in the draft repo and you can use those to get started building an application as a service. There is a larger problem, and that is as your service development increases complexity, that is as you master what's going on and you connect to other services and things like this, you're going to be modifying your Helm Charts. You're going to be modifying the container image to, for example, add environment variables and things like this.

As you do that, those are things that we can't really detect, because now you're building your custom application, right? Your special thing. In addition to which, it might be a private special thing about which we can know nothing and you don't want anybody else to know about it. So the question becomes; how do you do that? How do you handle that environment? Because it's not just getting started, and the Draft steps up here by giving you the concept of custom Draft packs. You can create a repository with your – Let's say you iterated on your service and you worked on it and then installed it in a custom development or staging environment. There are some secrets in there that the real world should know. What you might do is you'd take your application and you check it into a GitHub repo that is private, for example. The chart and docker file that you build for that application can now be extracted into a private GitHub repo that it follows the draft pack format, and if it does, you can do the same thing with Draft that you did with Helm. You can do Draft, a repo pack add, specify your private repo or public repo if you wish to share your configuration with other people.

Once you brought in that repo, you can now do a Draft create with the pack option and it'll bring in this precise configuration for that specific custom application so that you can actually blow away your machine or you can have some other developer come in to fix a bug may be on a service that you left, and they need to pick up with that precise service and they can do it securely and instantly without having to master any configuration at all. They just do draft create and it brings in the right pack and they can start working by doing Draft up. Draft up does an interesting thing. It builds the container right there and deploys it into whatever Kubernetes cluster so that you can begin testing it immediately as the artifact would ship, rather than testing the code and then hoping you packaged it correctly.

[0:31:14.8] JM: Right. So this is constantly sinking your laptops code with the code that's deployed to your Kubernetes cluster.

[0:31:22.0] RS: In a way. It's not really syncing. It is doing a forward motion. So, for example, your local code is going to be built into an image. That image that your code built is going to be the one that's deployed by Draft for you to test. If that test works correctly, then when you do git push, your CICD can actually pull that precise image, precise image and deploy using the Helm Chart into whether it's a staging service, a staging environment or a production environment.

So it's not really a synchronization activity. It's actually using precisely the container that you had been testing against and developing against, and that's exactly what you want to do in Container Land.

[SPONSOR MESSAGE]

0:32:09.3] JM: If you are on call and you get paged at 2 a.m., are you sure you have all the data you need at your fingertips? Are you worried that you're going to be surprised by things that you missed, errors or even security vulnerabilities because you don't have the right visibility into your application? You shouldn't be worried. You have worked hard to build an amazing modern application for your customers. You’ve been worrying over the details and dotting every I and crossing every T. You deserve an analytics tool that was built to those same standards, an analytics tool that will be there for you when you need it the most.

Sumo Logic is a cloud native machine data analytics service that helps you run and secure your modern application. If you are feeling the pain of managing your own log, event and performance metrics data, check out sumologic.com/sedaily.

Even if you have your tools already, it's worth checking out Sumo Logic and seeing if you can leverage your data even more effectively with real-time dashboards and monitoring and improved observability. To improve the uptime of your application and keep your day-to-day run time more secure, check out sumologic.com/sedaily for a free 30-day trial of Sumo Logic.

Find out how Sumo Logic can improve your productivity and your application observability whenever you run your applications. That's sumologic.com/sedaily.

Thank you to Sumo Logic for being a sponsor of software Engineering Daily.

[INTERVIEW CONTINUED]

[0:33:54.0] JM: I'd love to keep talking about Draft, but I want to get to Brigade because I think there's a lot of depth there. So Kubernetes was built to manage resources for long-running applications, which I think is mostly what we've been skirting around the edges up here. When we're talking about developing applications, I think for the most part we're talking about long-running applications if we're thinking about things like WordPress, for example, but it's also supposed to be used for short-term tasks.

So, I mean, if you'd go back to the legacy of Borg being developed, this was originally a system to be able to manage both Babysitter, which was the long-running applications, and global work queue, which was though short-term tasks, and brigade is focused on being able to manage these short-term tasks. Give a few examples of short-term tasks that you would want to run on a Kubernetes cluster.

[0:34:53.0] RS: Actually, it's interesting. Brigade is one of my favorite tools. We built it because we needed it ourselves. Brigade does both short-term tasks and long-term tasks, but the interesting thing is people tend to focus – Brigade isn't a tool for developers and dev ops. The easiest way to talk about it is if a batch script is really a series of commands the pipe a standard out to standard in, where the next command is its own process and works on the previous process as data, Brigade is the same thing, but using containers for the process and shared storage for the pipe.

As a result you can actually use Brigade to build set up and run complex structured pipelines, whether they're parallel and you wait for all of them to complete or some portion of them to complete, or whether they're completely in series, one after the other, one after the other reusing the data that had come before or bringing in new data. Brigade will do any of those things, because it's basically scripted container pipelines in your cluster. Now that's a very open-ended possibility.

Here is where we get back to your question. In fact, the easiest demo for something like this is something like a git push occurs and I need to build the image and I need to stage it, for example. Standard CICD kind of activity. The only problem is that, that that's one example of the kind of thing that you were talking about.

Brigade has a container that acts as a GitHub web hook listener, and so you can configure git web hook events to trigger a request to the Brigade gateway and Brigade will then kickoff the scripted container pipeline that you asked it to kickoff off. Now the funky thing about that is that's an easy way to demo it, but it turns out it can do anything. So people often think of it as a CICD solution. That is not what it is at all. It just turns out to be very useful for that.

I'll give you a simple example. I was – Let’s see. We were at KubeCon in Austin in December and we had a customer who came up who was working with satellite telemetry, and what they wanted to do was use – In this case, they happen to be Azure customers, but you can imagine any telemetry ingestion mechanism that a cloud provider absorbs, and they were using Azure event hubs to ingest telemetry data. But they weren’t using Brigade on the backend to listen for telemetry ingestion, and each type of telemetry ingestion would kickoff a series of brigade scripts on the backend. So things like ETL work turned out to be extremely easy to set up and maintain, because they’re just Kubernetes applications. Every brigade pipeline – Brigade itself is just the Helm install. There’s a chart for it, right? Then every Brigade project is just a Helm install. There's a chart for that.

The setting up of whatever your structured pipeline is, is just the JavaScript file. JavaScript is easy because pretty much anybody can pick it up relatively rapidly and it’s widely known already. So it just turns out to be easy. People ask us why we chose it. It's not any more complex than that. But basically you can do anything; ChatOps, ETL, long-running processes, just all kinds of things with Brigade. Personally, I think it's the geekiest, coolest, most useful tool we have, even though I think Draft is right in the wheelhouse of a service-oriented development.

[0:38:31.9] JM: Okay. So I think I mischaracterized it, and that's' because Brigade is a tool for running Kubernetes workflows, and a when I read that definition, I think I associated with it something like, "Oh, you want to do a map reduce job, and then based on the result of the map reduce job, maybe you want to run another map reduce job," but ultimately these are going to have some finite lifecycle. When, in fact, you're thinking of it in much broader terms where you have potentially a collection of short-term tasks and long-running tasks that together compose a workflow. For example, ChatOps. If you wanted to do ChatOps, you have to have these long-running event listeners for people saying, "Hey, Slack bot, kickoff a CICD pipeline," and then in order to spin up that CICD pipeline, maybe your Brigade workflow needs to spin up some shorter term resources, but you still are going to be needing to run those event listeners that the ChatOps event listeners as long-running resources. So encompassing both long-running and short-term tasks.

[0:39:49.8] RS: It really is. You actually described it pretty well when you re-summarized it. The fascinating thing about it is that it can encompass those things. Another thing that people ask me about, because it's JavaScript, you basically say – In JavaScript you say, "I'm listening for this event with this container that knows how to listen, and when I get that event, I want to do this thing with the payload and I want to pass the data to this container and I want that container to run, and then after that –" So you just use JavaScript to structure your flow.

But people think of it also as, "Well, isn't this sort of like serverless?" like OpenFaa, or Kubeless, or Fission or something, a functions as a service kind of thing. The answer is yes and no. You could certainly use it to run one function. There's no reason you couldn't. But it's really structured container workflow as a service rather than functions as a service. So yes, it is that flexible. You can kickoff entire long-running activity and you could just fire something and have it go away once it's done, either one.

[0:40:55.0] JM: It seems like you could compose it with those serverless systems, becauseo those serverless systems are really about being able to schedule resources in an opaque way. You just want to be able to run functions without addressing the resource allocation, and Brigade is slightly higher level than that. With Brigade, it seems like you want to be composing workflows among applications, among long-running applications, and among short-term applications, and among any other thing that can respond to an event handler, and you're not even thinking about the level of functions. So it seems like the serverless people are actually operating it a little bit lower level of abstraction.

[0:41:39.9] RS: Or I would actually say it at a higher level, but it really depends on your perspective, right? So let's take a sort of an application example and then you'd tell me which ones higher and which one's lower level. Brigade not only manages, creates a container for and runs the script and then returns the data and so forth, but it manages all the underlying resources in Kubernetes that you need. So if you need shared storage, it creates the PVC or it creates the other storage mechanisms, say, if you were going to use a disc that was going to persist or some other database access. It will manage that in Kubernetes for you, whereas you don't have to do that no matter where in the chain you are.

So if we take an example, it would make complete sense use functions as a service to kickoff based on some event and then turn around and invoke a Brigade pipeline from the functions as a service. In that Brigade pipeline, it would make complete sense to also turn around and kickoff individual relatively atomic functions as a service in the brigade pipeline as well.

So if you had functions as a service calling Brigade and brigade calling functions as service, the only one that sort of manages the internal pipeline in a long-lived Kubernetes way is the middle layer, the Brigade layer. So my question would be; which one is more abstract?

[0:43:01.0] JM: I don't know. I'm not sure.

[0:43:02.4] RS: Obviously, it's rhetorical and you can tell that I'm smiling. It really doesn't matter actually. The question is; which one is the tool you want to use, and they're both completely composable. You could trigger functions from brigade and you could trigger Brigade from functions. The question is; what works for you and what makes you agile and efficient?

[0:43:24.1] JM: Do you think there will be some centralization around the serverless platforms? Actually, I think a better question is; are people actually using these things outside of the companies that are building serverless platforms, or like Brigade – For example, Brigades is very exciting to me as a kind of software journalist, but are people using these within their companies or is it still like a little bit too early. Are people still just like getting their feet wet with Kubernetes as supposed to figuring out workflows?

[0:43:58.2] RS: Yeah. So the easy answer is yes and no. First of all, people are using serverless all over. It's super easy to do, and there are tons of short-term event-driven, super easy jobs and people want just something to hook up and get going, and that's absolutely happening, right? So I feel good about that. I certainly do. It's s just one more tool in the box to make the world easy.

The problem with serverless is that it's important to realize that once you have something that's easy and small, you're going to use 5 million of them because they're easy and small, and then all of a sudden you're back to composition, which becomes a problem. So serverless doesn't have a composition approach, and that doesn't matter at some scale, and then at some scale it will start to. Brigade has, as part of – Naturally, it's as a workflow engine. So composition is part of its model. So in that sense, it sort of makes sense.

What I would say also about Brigade specifically, as supposed to serverless, which definitely is being used on mass or certainly is in my experience, is that brigade, because it's so powerful and yet not serverless in a kind of an obvious way, we are seeing a lot of use in people who appreciate the functionality it brings, but I wouldn't say it's splashing the tech world on fire yet. I actually think it'll have longer legs in many cases than a lot of other things simply because of the power it brings.

[0:45:20.5] JM: Indeed. So that notion of the function as a service sprawl where you start to use lots and lots of them and maybe you don't really have a big picture view of where all the different places in your application or you're using different functions as a service, that would not occur as easily in Brigade because you have your atomic unit or at least one of the atomic units is the workflow, which is a chunkier piece of functionality than a function.

[0:45:52.0] RS: Yeah, there is a fundamental problem. I'll sort of restate it, because I love serverless, because it's another tool and I get to use it for things that it helps me with, right? But composition is going to be a problem with serverless, and in addition to which, once you start running serverless, it bills by the second or microsecond or nanosecond or petasecond or some second. At some point, you're running a VM and running a whole VM becomes less expensive than being billed by the second and paying for all those standby VM's that actually are underlying the service.

So there is sort of a trade-off both kind of compositionally and also financially at some point, depending on what you're doing. Brigade just turns out to do the composition by default. I remember, Brigade app is literally a configuration file and a JavaScript file, which means it's an application. You can debug it. You can walk through it. It's a structured program. So right in that space where composition start to become complex with serverless, Brigade starts to be darned interesting, darned interesting. Both of them have tremendous usage, but in different ways.

[0:47:03.7] JM: What are some of the other tools in the container ecosystem that are exciting to you?

[0:47:07.9] RS: I am a big fan of their concept – Okay. There's a couple of things off the top my head. I'm a big fan of the logical viewers that are out there. I use, for example, Weave Scope, to see my logical applications. There are several others that do a great job. There was a company that got bought called Netil that did a great layer 3 sort of examination of your cluster and they got bought by somebody or amalgamated, somebody whose name I forget, which I feel bad about, but people can look it up. I like all of the developer tools that are coming out. Draft tries to tackle very specific niche. I like it. I like where we're going with it.

The other thing that we do, and then I'll get to other companies as well – That we do is the Kubernetes extension from VS code, which allows you to get out of the process of kube control for a bunch of basic tasks like [inaudible 0:48:02.9] your pods or opening a terminal and dorking around inside a pod when you're developing and so forth. Something that is very, very common but you can't stand cutting and pasting, among other things. So that's extremely useful. The YAML extension, which the Kubernetes extension that Red Hat did in VS code is really, really valuable. There's a bunch of others that are either useful there, and that's for VS code.

Other tools I really, really like other than the scope and the Netsil one was, I think, Scaffold from Google is a great way of doing things that is similar, but not the same as Draft. I think the Jenkinx-X dev ops thing is super, super interesting and useful.

Oh! Another thing that if – So Draft believes firmly in artifact as truth, and that's why we don't create a pod and sync our data into it. What we do is we actually create the container image from your code and you work with that actual artifact, right? As I said before, artifact is truth from the point of view Draft. But there are other tools that give you slightly different experiences that are fantastic as well. I think vapor-ware has this thing called ksync, if you don't know about it, which among other things, basically, you can set up a container in a pod in Kubernetes and ksync will full-duplex synchronize their files from local into the pod, which is a completely different way of going at it, but the experiences fantastic.

There is a possibility that your artifact is not exactly what you think it is when you go into your dev ops work, but I think it'll be a wonderful experience for a lot of people. We have a bunch of other tools coming up that I know about. Oh! There's a great debugger called Squash by Solo, I think, which involves basically a bunch of sidecar work, and so forth. But it's a brilliant experience as well. It’s fantastic, and they’re doing fantastic work. I think there's four or five others that are coming just from the KubeCon in Copenhagen that where at that I suspect are going to add to that ecosystem. I think that, right now, that makes things a little complex for people, like which thing do I choose and so forth. I get that, but it's early days, and later on all those tools are going to help us find the right way to do it for everybody.

[0:50:18.2] JM: All right. I know we’re up against time. So I think of serverless platforms as one of these things that are higher-level applications that are being built on Kubernetes, and what I wonder is what are the other distributed applications that you expect people to build on top of Kubernetes in the near future, like databases or –

[0:50:41.6] RS: I see a lot of stuff being built on Kubernetes because it's so flexible. So I'm very bullish on Kubernetes generally because of its flexibility, and it was built that way and, I think, that part of it is going to really be the thing that is fantastic.

However, it turns out that there are some applications that don't really lend itself to that. Kelsey Hightower has been sort of tweeting recently about the fact that he doesn't think you should do persistent data on Kubernetes, and I actually agree not because you shouldn't or can't, because if you know you should and can, obviously you can go ahead and do it, and I think that speaks to Kubernetes flexibility.

However, highly available partitioning of data was always a hard problem and we just figured out after 20 years of VM experience how to do it in a kind of coherent and reasonably reliable way. Turning around and just making it containers because you can is just not really an intelligent use of your time from my point. So I'll put that out as a sort of a personal caveat, but I pretty much think that Kubernetes can run almost anything outside of that kind of thing, or maybe even should. There are always going to be environments where it doesn't make a lot of sense, but with other extensibility mechanisms, we haven’t really talked about things like Federation, cross cloud or hybrid across private and public cloud, like in your own data center. Those are interesting scenarios that are going to come.

Then there's things like disconnected scenarios where Kubernetes sort of assume the pod is either there or it’s dead, because I can't find it, or the node, for example, but you have things like Virtual Kubelet, which allows Kubernetes masters to think there's a node there and there may or may not be a node. It could lazily created when Kubernetes wants it, for example. You can imagine that being used in IoT situations where a node, of course, is either not there or there depending on what the device at the edges is doing, and Kubernetes just gets the information it needs to. So there's lots of interesting scenarios quite above and beyond the sort of scale out end-user app like Twitter or Uber, if that's your model, and I certainly think a lot of compute intensive stuff is going fit and already has with things like TensorFlow, ML workloads and various other things, Spark, and Kafka and all these kinds of things, which by the way are Helm Chart. So if you want to install them with the one command line, you can just do Helm install Kafka.

[0:53:10.0] JM: Indeed. Well, I think that's a good place to close. Ralph, I want to thank you for coming on Software Engineering Daily. It's been great to have you talk about such a variety of projects.

[END OF INTERVIEW]

[0:53:21.2] JM: If you are building a product for software engineers or you are hiring software engineers, Software Engineering Daily is accepting sponsorships for 2018. Send me an email, jeff@softwareengineeringdaily.com if you're interested. With 23,000 people listening Monday through Friday and the content being fairly selective for a technical listener, Software Engineering Daily is a great way to reach top engineers.

I know that the listeners of Software Engineering Daily are great engineers because I talk to them all the time. I hear from CTOs, CEOs, directors of engineering who listen to the show regularly. I also hear about many, newer, hungry software engineers who are looking to level up quickly and prove themselves. To find out more about sponsoring the show, you can send me an email or tell your marketing director to send me an email, jeff@softwareengineeringdaily.com.

If you're a listener to the show, thank you so much for supporting it through your audienceship. That is quite enough, but if you're interested in taking your support of the show to the next level, then look at sponsoring the show through your company. Send me an email at jeff@softwareengineeringdaily.com. Thank you.

[END]