Open Source Week – Day 2

IBM Code Tech Talks

02/27/2018

https://developer.ibm.com/code/techtalks/open-source-week/

>> TODD MOORE: I'm Todd Moore, your host for today. Today's topic centers around Cloud data applications and containers. We have a great speaker for you. Doug Davis. He has been part of our team here in IBM working in Cloud native technologies for quite a bit of time and in this case Doug is very active in the Cloud Native Computing Foundation and an active member of many popular technologies including Docker, Kubernetes and the other technologies found in the CNCF. He has most recently been driving our activities around new eventing specifications being developed at the Cloud Data Computing Foundation called Cloud events. This topic that he will be discussing with us now is on Kubernetes, where Doug co-leads the service catalog SIG and is part of the conformance workgroup. So Doug's talk today, this afternoon, this morning, good evening, is on Kubernetes. It is eating the container management space. That's a great topic, Doug. So I'll turn it over to you and let's get going here.

>> DOUG DAVIS: All right. Great, thank you, Todd. Let's jump right into it. On this talk we are going to basically give a brief overview about what Kubernetes is, what the community has been up to, the status of it. A little bit about what IBM is doing with Kubernetes and pointers on where to go to get more information if you want to get involved in the community. We have a lot of information here. Unfortunately I won't be able to cover everything in the charts but there is a lot of information in there you can follow and read later on on your own later. So let's get started.

Let's talk about what is Kubernetes. Kubernetes is an Enterprise level container organization administration engine. Basically there to manage and provision your applications employed as containers. So basically it will manage all the aspects of the container infrastructure itself. Not just the container itself but the volumes, networks, secrets. All the resources that go into your application in order to get it up and running. And that's what Kubernetes is basically there to manage for you. It is what they call -- leverage is what they call a declarative model. You tell Kubernetes what you want your infrastructure or application to look like relative to how it's scaled. How many instances you want and stuff like that and

Kubernetes will do whatever it takes under the covers to make it happen. To make reality match your desired state.

Now, just as a little side note here, for those of you interested the word Kubernetes means helmsman in ancient Greece. It is an orchestration engine for your containerized applications. Kubernetes was originally created by Google. What they did is took their expertise for Cloud scale management from an in house project called Borg to the open source world. A difference between Kubernetes and other open source projects in this space they wanted Kubernetes not just to be open source but open governance. Open source is source code is out there available for anybody to use, leverage, make changes to or submit changes to. That's not necessarily the same as open governance. In an open governance model yes, it is open source. But not everything is controlled by one particular company, right? People have a path forward in which they can get into leadership roles to help shape the direction of the project. And that was very important to Google. They didn't want it to be a Google-led and Google-only project. They wanted to open it up to the world and talk about how they managed to do that beyond open sourcing it. They put it out there, got lots of attention. Quickly started getting a whole bunch of people coming on board. Especially people coming from the Docker world looking for a slight alternative to Docker itself. People wanted an alternative in the space to Docker. So Kubernetes seemed to fit that bill. People were enamored with the slightly different programming it offered up. As a result, it became a very popular project. One of the most popular open source projects out there. The challenges that it brought up we'll talk about. But understand it originated from Google and has been growing very, very rapidly.

So literally about the community itself before we get into some of the technical details of the project. If you want to start interacting with the community itself, look at the communication, MB filing under second under communications. That's where you want to go. You will find information about the mailing list, Google groups, weekly phone calls and in particular the other thing I want to point out is all the various SIGs that they have. SIG stands for special interest group. The Kubernetes is such a large project they couldn't manage it just as a single large project. They had to break it up into individual little groups. Different people are interested in different aspects of it and didn't want to have everybody being subscribing to the same mailing list. What they did is set up the special interest groups or SIGs. If you're focused on one particular area of Kubernetes, say storage. If you go to the storage SIG you can talk to other people interested in that aspect of Kubernetes and focus on just that particular part of it and not get flooded with all the other communications around the other parts of Kubernetes. It is not that as big as the project is you can feel like you can participate and hopefully even take ownership of a particular piece of it because it's broken up relatively nicely. That allows each project to have its own -- each part of the project to have its own life cycle and own governance model to how they work. A little freedom in that that's kind of neat. As I mentioned, since the project has been out there, it has gotten a lot of interest. You'll find a lot of key players over there these days. Google, IBM and Microsoft which shows how popular and successful it has been to date. With that introductory information let's talk about how it works under the covers.

Forgive me for this but this is a gross over simplification but fairly accurate. At its core Kubernetes is just a database. If you look at this picture you will see database in the middle of the screen. And really all it is, is a database with an API server in front of it so you can talk to the database. That's all Kubernetes is at its core. You need more than that. So what they have are these notions of controllers or watchers that basically watch what happens inside the database and as things go on in there, as resources are created or modified, these controllers will detect that and then take action based upon it. So for example, if you create what they call, say, a network resource inside the database it will be a network controller that will detect it and say okay, new network resource, I need to go off and create that network resource in the appropriate fashion based upon the configuration and it will do that job for you, right? But at its core it is a simple database. Go ahead.

>> Is there a specific database or is it one of the regular databases that they are using?

>> DOUG DAVIS: Right. Right now they use SCED. I know it isn't a traditional database, more of a main value store kind of thing. In all honesty they claim you can plug in other databases as long as it adheres to the APIs and has the right functionality. I've seen people do experiments with other databases but right now it's the only one you can use with it. You have the database at its core, watchers watching for changes in the database and then reacting to it. That's the entire model at its core. What's interesting is while Kubernetes comes with sort of a canned set of controllers that watch the database, because the APIs with these controllers use to talk to and monitor the database are basically the standard Kubernetes APIs, you can write your own controllers to do what you want. So if you don't like the scheduler controller that's in there to -- that decides where to put your application on which worker node

and stuff. That's fine. Write your own. It is the same API that the controllers use that you would use when you talk to Kubernetes. So it's all the standard set of API and this availability for you to write your own customized controllers or other components are really one of the big selling points for Kubernetes. If you don't like what it does, write your own. It's not to say in a negative way meaning it's a cop-out and they won't try to help you. It gives you the flexibility and freedom to customize Kubernetes in the way you need to get your job done without necessarily having to start from scratch by redesigning the entire system from the very beginning. It is kind of a neat and flexible model that some people have really grown to love. For those of you familiar with Docker, obviously that flexibility comes a little more complex than you might be used to. Docker has done a wonderful job of simplifying the user interaction model. When you look at Kubernetes verses Docker people may have more of a learning curve with Kubernetes. They give you more freedom. Once you get over the learning curve you will find a lot of similarity between the two. The user interaction model is a little different.

The Kubernetes resource model, the types of things that the database can store inside there is quite a long list. This list I show on the left-hand side doesn't do it justice. I won't go into all these but I want to at least call out one or two key ones. In particular pods. Those of you familiar with Docker may know that when you run your application you will run one or more containers. And each container is basically an entity unto itself. The thing that you will be deploying into the environment. In Kubernetes what they did is said our lowest common deployment artifact isn't a container. It is a pod which is a group of containers. Because they realized that a lot of times people don't want just a single container running. They will have what they call these ancillary containers that sit alongside of the main container. Maybe to monitor the system or manage the log-in for you, right? They found a lot of time these containers are grouped together and when you deploy your main application you find you have two or three other containers that go alongside. They decided to make it a first-class entity. You will ask for pods not containers. And yes inside you can have your containers. I want to point that out from a differentiating perspective from Kubernetes and Dockers because people need to understand that. The other interesting part about Kubernetes from a flexibility perspective is it actually doesn't define a notion of application. That may sound a little odd. What they do is allow you to put labels onto your particular resources, in particular pods. So for example let's say I want to define what they call a service or an endpoint. Those are the resources that allow you to define how you are going to expose your application to the rest of the world, right? So rather than saying this service, which is sort of the definition of my application from an external point of view. Rather than saying this service talks to that pod over there or that -- or those clusters of pods and identify them by name or something else, what it allows you to do is say I want this service to route our requests to all pods that have a particular label. And that way you can actually have a mixture of types of pods, right? You can have some pods with that label that are version 1 of your application and other pods that have version 2 of your application. As long as they share the same common label the service the resource is looking for, requests will get routed to it. It may sound a little bit complicated but it gives you incredible amount of flexibility if you want to do things like rolling updates to your application. It is very powerful. And so you'll find that's the way Kubernetes actually tends to sort of reference objects within the data store or within the database is by these labels mechanism. It is a flexible and cool model to have.

So as I said basically a database. So the main interaction with Kubernetes from an end user perspective is through the controller called cube CTO. It's the crud operation, create, delete and update. That's the crud operations and you pass them in Jason file with the definition of your resource. Okay? So simplistic level -- basic level it's simple. Crud-type operations on resources talking to an API server. Very simple. Now, they have tried to make the UI a little bit softer or nicer in some places so you will see some customized commands. Cube control scale to say you want five instances instead of one. Other the covers understand all the scale is doing is a Kub control apply or update kind of operation to modify the resource model. It just looks more user friendly if you can use the scale there. It's a facade for the real operations. Don't be surprised when you use it that you find you are living pretty much on the crud operations themselves and not using something a little more user friendly. That's perfectly natural.

All right. So let's quickly walk through a real world scenario here. Let's say following the numbers on the picture using Kub control I'll take the employ employment and send it to the API server. It's nothing more than a server in front of the database. And so the API server gets the request, sticks it inside of SED and its job is done. At that point what will happen is three, the controllers or watchers will be monitoring the database and they'll recognize oh, a new pod has been created. Another application is now in the system. So the controllers or the set of controllers are now kicking in and do their job. For example, maybe what you've deployed isn't just a single pod. Maybe it's what they call a replication set. A pod you can scale. What happened is the controller will detect that new resource in the model and take some action. Let's say we deploy a pod and there is a scheduler or a replication controller says I'll need to scale this thing up but I don't have any instances of this pod in my system yet. The replica controller will say I'll take the definition to the pod and create five instances of it inside the database. They want it to be scaled up to five instances. At this point the replica controller is done with its job. It is completely finished and created five new pods inside of the database and it is done. That's great. The database knows about it but aren't actually running it. That's when the next controller will kick in, right? For example, the scheduler controller will say I have pods in the system not actually scheduled to run on any particular worker node. They're nodes in the system that will run your containers. I have these pods. Not scheduled anywhere yet. The schedule looks at the pods. Figures out which worker knows it should be assigned to. Updates the resource with that information and then it's done. Again, still not deployed but we're getting closer.

Next step. We have a Kublet. The brains of a worker node. It is managing all the containers on the node and it's responsible that they're up running when they're appropriate. It's another form of a controller. It's watching the database and detects there is a new pod that has been defined in the system and it's been scheduled to itself or to that particular worker node. It will then take the definition of that pod and using the Docker engine or whatever container run time on the system will go ahead and deploy the pod and all the containers in the pod to get its job done. And that's it. If you notice the pattern here is pretty much you modify a database and things react to changes in the database. And that's Kubernetes at its core and that's the whole model itself. With that basic model you get this incredibly flexible system where you can manage, as I talked about, pods, containers, networks, volumes, everything you need for your application during the simplistic database, watcher, controller model. I want to talk about Kubeproxy. For purposes of understanding it, think of it as just the manager of the networking for that particular worker node. So all inbound and outbound traffic to the containers running on that system basically go through Kubeproxy. Understand that it's really there to manage your networking aspect of your application.

So -- with that, that's basically the heart of Kubernetes right there. There isn't much more to talk about in terms of the core pieces of it. Obviously if you wanted to dive deep into how volumes work or the networking works, there is a long talk that can go on in that. At its core 30,000 foot level that's basically what Kubernetes is all about.

What is the status of Kubernetes itself? They are currently working on version 1.10. 1.9 was last released. It just celebrated its first birthday. I have listed out some of the newer features that might be of interest to you. I won't go through all of them. I don't have time. One or two I want to point out. The first one is a custom resource definition. This is kind of an interesting one. Earlier I talked about how you can customize Kubernetes with your own controllers. It allows you to extend the resource model itself so you don't have to stick with just things like pods and volumes and whatever Kubernetes itself decided it wanted to support. Using CRDs you can actually put other types of resources into the database and use Kubernetes as a data store and write your on controllers to watch for the CRBs and react on them. Allowing you to extend not just the components in the system, but the actual model itself. Which is something you don't see in very many projects these days. Kind of neat.

I want to also point out API aggregation. Take the API server we talked about before, the main interface of Kubernetes. And actually split that out to be across several hosts so that you can have one API server deal with just pods, another API server deal with networking. That way you can better manage your environment and not have one particular node in your environment be a bottleneck for all requests coming in. It's about giving you the flexibility to configure Kubernetes in your environment that best suits your needs. That's again -- I think it's interesting they're thinking about providing as much flexibility as possible to people who want to deploy this stuff in their environment. Not everybody has a one size fits all environment.

What's next for the community? As I mentioned earlier they've been growing by leaps and bounds. That's been wonderful from a project perspective. Everybody is very excited. The down side is they're moving very, very quickly and that's good and bad, right? On the bad side, what they found is that there are too many changes going in too quickly at times. Every now and then they need stabilization releases where they spend one release just focused on solidifying the core, fixing in the bugs that popped up and making sure they aren't going too fast for people. If they move too fast and things break, then they will hurt the user experience and their customers and that's the last thing they want. It is great they're aware of this sort of good problem to have and are taking active steps to try to fix it and that's all good news. Now aside from Kubernetes itself being this platform for orchestrating your containers and thinking that they want to rule the world or eating your container of the world they are working with other communities. For example, Docker, Cloud Foundry because they want to have an interoperability statement. They want people writing networking interfaces or back ends, storage back ends, they want to have those people plug those same systems into Docker, Cloud Foundry, Kubernetes. The Kubernetes community is working with the other platforms to try to develop interoperability statements. New specifications the make life easier for people who want to perhaps use the same storage interface or storage system with their Kubernetes and Cloud Foundry install. Some people like IBM will offer up multiple platforms to their customers, not just Kubernetes and they want to be able to share the back end systems across those platforms. So Kubernetes is working very closely with those other platforms to get that interoperability statement, wife I think is great.

Within IBM as I mentioned IBM offers Kubernetes service in the private and public crowd. Watson does Kubernetes. We are using it ourselves. We're bought into this completely. On this list here you see some of the key activities that IBM is involved in. Show you that IBM is all in on Kubernetes and we're trying our best to not just make sure Kubernetes works just for IBM but the community at large. In fact, a lot of the activity that we're doing in the community is usually driven by what the community needs, not necessarily IBM. Aside from a couple features, Kubernetes works for us fairly well. Our activity in the community is there for the community itself. That's great from the IBM perspective. So if you want to get started on Kubernetes I do have some links here. In particular things like getting started guide produced, a source code and a couple other links here. At the bottom we have some links to some patterns and patterns are interesting concept that I think are relatively new out there. It walks you through a real world example of how to get a particular task done by not just talking about it like a blog might do but actually walks you through the process of looking at the code and actually deploying it for real. They will give you the code for it, the instructions how to deploy it and try to bring you up to speed with the technology itself and not just give you a list of instructions to do and not explain what is going on. They're a great education resource on the technology.

Circling back for a second you want to get involved. One of the best places to start is with the backlog of issues. Most open source communities out there love it when people who want to get involved start by helping out with some of the things that are less exciting. Help them with documentation, trying to get rid of the backlog of issues. If you focus on those things it is wonderful not just from a community but helps you learn things without being thrown into the deep end. You can find low hanging fruit to get involved in and a great learning exercise for you. With that and here with some information and links about how you can get involved not just in the community but in meet-ups and conferences around Kubernetes. Let me pause and see if there are any other questions before we wrap it up.

>> There are no questions in there. I have a quick question for you, Doug. In the architecture document you pointed to a Docker engine. Can you briefly talk about the interaction between the Docker engine and Kubernetes.

>> DOUG DAVIS: Sure. The Docker engine acts as the container orchestrator on a per-node basis. What is going to happen is the kublets, when it detects a new pod. The kublet becomes a Docker client at that point and talk to the Docker engine and say spin up one, two or three containers, however many that are needed within that particular pod, and just tell it to bring it up. So it's really the kublet that will interact with Docker to make sure the right number of containers on this particular node are running and use it when they go down and manage the memory usage. The Docker engine is just one choice. Other engines you can put on there. You can do rocket or you can start and fairly soon start using the subset of Docker meaning container D, which is if you understand Docker is the core container engine within Docker itself. You can put container D in there. The point is you have the flexibility to replace that Docker engine with some other container orchestrator if you want as long as it adheres to the interfaces that Kubernetes has defined, the CRI interface. Another integral part of the system.

>> Okay, thanks.

>> All right. Any other questions? Comments?

>> I don't see any.

>> Okay.

>> Doug, thank you very much. It was an excellent overview of Kubernetes. And as you said, we're all in Kubernetes. Our container service is Kubernetes at its core and runs obviously all the APIs and structure that come with that. So thank you very much, Doug. And we will continue shortly with the live stream. Moving on to our next topic. So just hang in there with us and after about a 20 or 30 second break we'll be continuing.
>> Minor update. You might want to add to the major companies on board with the CNCF. I don't know that you listed Amazon. If you did, I missed it.

>> Did I miss that? Okay. I'll try to remember that. Thank you, Jeff.

>> It used to be anybody but Amazon club, right?

>> The very nice thing about Kubernetes it has united every major player in Cloud across the entire -- regardless of geography.

>> Yep. It's great for our clients and the ecosystem at large.

>> TODD MOORE: Okay. For those of you who are just joining us, it is probably time to get going, Kathy, is that right?

>> Yes, go ahead.

>> TODD MOORE: For those of you just joining us I'll introduce myself again. I'm Todd Moore, IBM's Vice President of open technology here today with Doug Davis. Doug is going to continue down into talk number two now in our streaming session. This time talking about the Cloud Native Computing Foundation. It is an organization that came about really as a starting place for home for those activities in Cloud data computing that were some of the most significant developments that you will find in the industry. And as a result of creating the Cloud Native Computing Foundation, the first member of the family so to speak was Kubernetes. But we then have since added many other projects. And Doug will take you through that. I, too, will help answer questions in this session as I'm chairman of the governing board. So Doug Davis is heading up our Cloud Native activities. He has been an active member in many of them, maintainer in Docker, working in Kubernetes, working in the CNCF helping to support the activities that go on there. He is co-leading the working group within the cloud native computing foundation and maybe could give us inside into that as well as we go through this next topic. Doug, I will turn it over to you and let's have a lively set of questions. There is lots to cover.

>> DOUG DAVIS: Thank you, Todd. A quick note on the title. So there are lots of differing consortiums out there of industry, standard bodies or other types of activities out there. And I think this title is very appropriate in the sense it isn't like many of the ones out there. It is quite a bit different. I think that's a very positive thing and as we will talk about in the upcoming slides. It is slightly different than what you might have seen in the past. Again, we talked about the CNCF or Cloud Native Computing Foundation, it's history, why it was formed, it's status. A little bit of what IBM is doing with it and finish up with information about how you can get involved if you're so inclined. So first let's begin with a little bit of an eye chart. I don't expect anybody to actually be able to see this. If you go to the CNCF website you get a better view of it. The point here is that within the Cloud Native space there is a lot of activities going on here. And it's constantly changing, constantly growing. And one of the things I was quickly realized

is while there are lots of opportunities out there for people to get great things done, there is also a lot of confusion about what people should actually be using, how they should use the various technologies together, which ones are good, which ones are bad. Like I said, a lot of confusion out there and that's one of the things the CNCF is trying to provide to the community is a little bit of guidance and recommendations for people going forward. And we'll talk a little more about what that means. Just understand that really the CNCF at high level is there to help people. It is not there to necessarily dictate things. With that sort of as a level set, let's talk a little bit about what the CNCF actually is. It's there to basically promote Cloud Native technologies and all of the positive or good aspects around Cloud Native technology. That doesn't include just the technology itself meaning Kubernetes and stuff like that but also the best design practices that go around it. Things like computing, using micro services. All the best practices that we've been told for quite some time to leverage. CNCF is helping to make those things a reality. We're past the time we talk about things and pay lip service to it. These things are happening for real this time and it is neat to see it actually take hold and people starting to do all the things we talked about doing in the past but never found the time to do right. The CNCF is helping to make sure it becomes a reality and also there to provide clarity and guidance for the Cloud community itself as they start looking at these technologies. It is daunting to see all the various options out there.

Now having said that, it's very important to understand that the CNCF is not meant to be a king maker, meaning it won't be the one to pick which technologies win over others nor is it there to be a standards body. It is really there just to sort of help provide people some guidance more than anything else, right? Now with that goal in mind, though, they obviously have to have -- I don't want to say necessarily a bar or standards that people have to meet in order to participate but rather they are looking for some sort of -- some best practices that projects adhere to to be sort of brought in under the CNCF umbrella. For example, an open source project that may want to join the CNCF that maybe is only pretty much dominated by one company, right? If that was -- if only one company was participating in there that may not be a good fit for the CNCF. They want to have a set of projects under their umbrella that are really community-based and don't have this feel of maybe being just one company pushing an agenda, right? Things like that are really what they are looking for in new projects that come on board. Obviously they have to be related to Cloud Native. Container-type space and they are definitely not in favor of becoming king makers or picking the winners. So what you will see going forward are competing projects within the CNCF. That's perfectly okay as long as they all adhere to sort of the guiding principles of what it means to be Cloud Native. We welcome the notion of let 1,000 flowers bloom and let the community and industry decide which is the best one for their needs, right?

So with that in mind of, you know, it's an organization for people to come together to discuss best practices around Cloud Native technologies and things like that the CNCF is also a place where people can come together to collaborate on new activities. What I mean by that is, people will recognize there is a spot within a community where some level of interoperability or harmonization might be of benefit to the broader group. For example, defining some set of APIs so people don't have to reinvent the wheel over and over. Maybe they want some commonality in that particular space because the level of -- excuse me -- the level of innovation has sort of died down a little and people are naturally settling towards one particular API set. The CNCF can be the forum where people will come together and say okay, let's see if we can jointly develop a single-A PI definition that everybody can adhere to. They won't mandate it but the community together has decided this is the one we would like to promote at this time. I list a couple of them at the bottom, CNI, CSI. I'll talk more about what they are later. The CNCF is just there to help promote technologies and be a community in which the entire Cloud community itself can come together and collaborate together. They aren't there to dictate anything to anybody. They just want to be a forum where people can work together.

How is this created? CNCF was originally created more as an initial home for Kubernetes and that's why Kubernetes is referred to by its anchor project. As Kubernetes was being developed and as I mentioned in a previous talk they were looking for an open governance location for Kubernetes to continue its growth. They realized that Kubernetes by itself is a good project really will have a bunch of ancillary things around it. All the other Cloud Native technologies like monitoring and stuff like that will be related. So they decided rather than having a single organization just for Kubernetes, what if broaden it and what do we need for the broader Cloud Native space itself? Out of that the CNCF was born with Kubernetes as the first anchor project. It was founded as a new project and IBM was there driving it along the way. We care very much about open source, open governance type stuff. Since then we've had I believe up to 16 different projects now. I list it in the middle of the chart. You can see we've been growing by leaps and bounds and constantly growing all the time. Now, the CNCF isn't just about those particular projects unto themselves. They're taking the initiative to look at the bigger picture for educating people on Cloud Native technologies or best practices. They're looking at setting up certifications programs when people go to use some of the Cloud Native projects that the projects being deployed adhere to a set of well-defined performance tests or certifications and stuff like that. The CNCF is there to help the projects move forward where the projects feel it is needed. And so they are looking for new opportunities to expand enough space. So where is the CNCF? For each individual project, even though they're CNCF projects you have to go to each individual project itself to get information about the project. The CNCF is where people can go for one stop shopping to see what they want to -- what all the various projects are and they can take the next step by following those links. Who is involved? Pretty much all the major players are within CNCF. IBM is there, Todd mentioned, Google, Microsoft Oracle. When it comes time for us to discuss particular projects and solutions to problems that our customers are having, chances are we'll have the right players there in the room and eager to have those discussions. And CNCF isn't just North American or European based. It is really global. In particular we're seeing an uptick within China itself. All right. The current status.

As I said there are 16 projects right now under the CNCF. They have a conferences twice a year. Usually one in the United States and one over in Europe. What's interesting about the CNCF they aren't just interested in their own success, right? They actually are sponsors of other conferences itself as long as they're Cloud technologies. They're about helping Cloud Native technologies. You'll see them do things like be supporters or sponsors of things like Docker-con. That's another unique aspect of the CNCF that you may not see in other consortiums. I mentioned this earlier. They are very much interested in harmonization and interoperability efforts. So some of the projects I mentioned earlier like CSI and CNI. They are taking the lead at helping the community come together to define the interfaces for these back-end systems. That way storage providers, for example, don't have to write a whole new set of -- adhere to a different set of APIs just because they want to plug into Docker versus Kubernetes versus something else. They've done a wonderful job of getting the platforms and communities to come together and say let's stop bickering about different APIs. The functionality matters. We agree on that. Let's go forward from there and people have really jumped on board and come together nicely. The other interesting thing is aside from just defining or looking for opportunities to define harmonization and APIs, they are also taking steps to look for newer spaces where they can help provide clarity. It's not just about clarity just for cloud data at a broad level but a slice of Cloud data. Serverless is a new technology. It's new. Let's figure out what we want to do in that space. They started a new workgroup that IBM is co-chair of. Their mission was to write a white paper going on around serverless. They talk about certain types of architect tours, what kind of use cases, how it compares with other Cloud Native technologies like PaaS for Kubernetes when you want to use one versus the other explaining what is going on without people having to learn it all themselves in a short white paper they can get that information, that's great. Now at the end of that, bill, they came up with a set of recommendations. The technical oversight committee said -- one of them was looking at a new specification called Cloud events. Looking at trying to define what an event structure might look like. They thought that might be an interesting space where we can look for some sort of harmonization. All the different serverless environments deal with events and can't we get an agreement there? They thought it might be a small baby step in looking at a broader harmonization effort. They spun up a new working group to do that. Cloud event specifications being worked on and the point here, though, is it's not just about sort of letting the community by themselves decide or helping the community come together to focus on interoperability around existing interfaces. It is looking for newer things the community might want to look for in the future. They're also involved in things like performance, certification working groups to help ensure people can get a level of guarantee about their platforms when they go to look at resemble using Kubernetes on multiple providers. If the Kubernetes provider claims they're adhering to a certain conformance test suite the application owner can be assured when they run their application on AWS versus IBM Cloud or Google Cloud they'll get a certain level of semantic equivalents because they adhere to the same standard. They also look at continuous integration and cross testing. They have a set of nodes or servers that are available for people to leverage for their own purposes to get their work done but also they are using that as a test bed and setting up a testing environment. They're not trying to compete but offering it as an alternative for people who may want to leverage this. Cross project testing and make sure they integrate together and still work. To help out the projects to make sure issues are found sooner rather than later to help the projects move forward.

All right. What's next for the community? Pretty much it's more of the same. They'll look for additional Cloud Native technologies to bring on board and other areas for interoperability, harmonization and other kinds of stuff. I think this space is right for that work and other places to promote the technology. You'll find them promoting the stuff going forward.

What does IBM do with CNCF? Todd Moore is chairman of the board of the CNCF. IBM is involved in many of the open source projects under the umbrella. We're very active in the performance working group. Defining the performance minimum bar that platforms offering up Kubernetes adhere to to guarantee the level of interoperability and portability people are looking for for their applications. All right. So if you want to get involved in CNCF here are some links for you. The CNCF website is the best place to start. For each individual project within the CNCF you want to participate in follow the link to the project itself directly and just to wrap it up some additional links if you want to get involved in terms of the meet-ups, conferences, join in some of the discussions going on. IBM has a monthly newsletter you might find interesting and we talk about CNCF and other Cloud Native activity and technologies. We have time for questions.

>> There is one here, Doug. It says will -- (inaudible

>> DOUG DAVIS: Great question. As of right now it's not part of the CNCF but that is where it is probably going to go. Everybody involved in ESGO has decided that that's where they want it to go assuming CNCF would take them. It is meeting all the criteria. I think they're just waiting until they reach a certain bar of -- I can't think of a better word other than maturity. They want to make sure they reach a certain feature set. Then they can say okay, now we think we're ready to get that extra promotion that joining the CNCF will give them. Expect that in the not too distant future.

>> Another question that talks about micro profile. There was a pattern with Kubernetes. People are saying is it well used in the industry? Do you know anything about that?

>> I'm not that familiar with micro profile.

>> DOUG DAVIS: That's coming out of our Java team. The micro profile work is being done out at the eclipse foundation and you can join in the project out in eclipse. Many IBMers who are part of that and it is a pattern that we see emerging now. The micro profile work really has only been released a fairly short period of time. But it is gaining folks who are joining it as part of the community and looks like something that will be popular. We expect all of the languages to be well supported under Kubernetes and patterns to arise and take advantage of that. We see node.js popular, Java and others and we believe the micro profile work accompanied by things we've open sourced our

virtual machine implementation, the J9 is out there. Our friends in Oracle have put basically Java into eclipse now. There is a center of gravity that's formed there and it will participate in the new container world in a first class way so join in. It's a good place to get going if you're part of that community.

>> Cool. Can you outline the steps for somebody to participate? I know it's for projects but can individuals participate and can you make a distinction of people who are IBMers and people who aren't IBMers?

>> DOUG DAVIS: On the individual projects themselves and the best way to join those is to just follow the link to the project itself. Look at the backlog of issues, read up on the project and figure out a way to help in small ways. Submit requests to fix those things. That's the best way to get involved in individual projects themselves. At the CNCF level I would go to their website and then in particular I would go to the technical oversight committee part of the CNCF website and look at discussions going on there. Take a look the different working groups that stood up. Things like the CNI and CSI working group or join in on the TOC calls. They're open to the public and have them once a week to hear what they are talking about in terms of newer activity they want to get involved in and start volunteering. For example, a lot of times they look for people to volunteer to do due diligence. It just means look at the project find out information about it. Its background, how many different companies are involved in there. The basic statistics about the project to help the TOC make an informed decision about whether it adheres to the good principles they're looking for in a project. Lots of ways to get involved. I start with the technical oversight committee's part to get an insight what's going on.

>> Going back to what is said about CNCF not being very hard on people to comply. I'm sure that there is a compliance suite that people have to run to ensure that they comply to the standards and how hard does they enforce that? Can you look at somebody and say you are a better citizen than others?

>> DOUG DAVIS: There is always conformance test as well as a mark. So if you want to use the mark and the Cloud Native foundation looks at the mark you have to certify yourself and pass the test cases. And that is an ever growing set of test cases but it is a way that all of us in the community who are using Kubernetes can demonstrate interoperability to our base of users and clients and customers that want to depend on what they have as being something that is interoperable across multiple implementations and that's where the key is. You get choice now. All of the major vendors have implemented it. A great place to be in that regard.

Just to add to that. It's more of a bullying kind of thing. You are either conform ant with the test that Kubernetes has laid out or you're not. You phrased it as one one implementation be better than another? It is simply are you or aren't you compliant with the test suite. This is about interoperability and making things easier on the customers.

>> It seems to me that people can get involved and there is room for people to get involved and jump in and help out, right?

>> DOUG DAVIS: Definitely.

>> Cool. There are no more questions, Todd.

>> TODD MOORE: Okay. Thank you for listening in on our talk on the Cloud Native Computing Foundation. There are links within the presentation for more information. Any of us would be happy to anxious questions about it. We hope you'll join us next in Copenhagen where we have our next large community event. We expect many thousands of people to attend. It is a great place to come together and learn more about what is happening in Kubernetes and related technologies. 12 or 13 talks will be on ISTIO. Join us. We'll talk a pause here and we'll resume the stream in about 30 seconds or so and with that we'll move on to session 3 of the day.

>> TODD MOORE: Okay, so those of you who are joining us here for session 3 of day 2 of Open Source Week I'm Todd Moore, your host for today. I'm IBM's Vice President of open technology. I help to establish open communities and representing IBM. I support the Cloud Native Computing Foundation as the chairman of the governing board and I chair the board of the no JS foundation. Today we'll talk in this session with Doug Davis. Doug has been deeply involved in Cloud Native computing and the technology surrounding that. And is a member of our team who is working out in the CNCF but also worked in Docker as a maintainer and Kubernetes and other organizations. In terms of Docker Doug was IBM's first maintainer to be added to the project and port Docker into other platforms and deeply engaged and quite knowledgeable. Doug, I know this one will be jam-packed. I'll turn it over to you and let's get going. We can't hear you, Doug.

>> Doug, are you there?

>> Seems like Doug has lost his audio.

>> I will ping him. Doug is not on mute. We are going to take a quick pause here while Doug comes back into the platform.

>> DOUG DAVIS: Can you hear me now? That's weird. When I used the headset it doesn't work. Let me try it again. Can you still hear me? I have no idea what happened there. Okay. Do you want me to start over and just continue?

>> TODD MOORE: Why don't you start over.

>> DOUG DAVIS: All right. So thank you, Todd. In this talk we're basically going to give an overview of Docker itself. The community, how Docker got started. Provide a little insight into where they're headed and talk a little bit about what IBM has been doing with Docker and give you information how you can get started with Docker itself. Moving forward.

Before we talk about Docker, I think it is very useful to talk about what are containers in general? As most basic level you can think of a container as nothing but a set of processes run in isolation. By isolation I mean each container gets its own networking stack. Its own process I.D. space and mall points. All the things that you might typically expect with a virtual machine, that's what you get within a container. The big difference here, though, is unlike virtual machines, which are more of an operating system-based isolation, meaning each virtual machine has its own full copy of the operating system. Containers are process-based. What goes into the container in terms of the processes are just what you need to get your job done. Meaning just your application. So within a container, you typically will not see, for example, simple processes running. Not that you couldn't run them but you typically don't. Most applications really don't need those system processes running to get their job done. If you're running any server all you need is the server, run time it uses and that's it. You don't necessarily need SSHD. Don't include them in your container. So when people when they first start learning about containers they try to compare them to virtual machines and from an isolation perspective I think it's fair to think of them in a similar way to get your head around what kind of isolation we're talking about or what you can do with them. Understand that they are radically different in the sense that operating system -- virtual machines are more operating system based where containers are just individual process based. And that's very important distinction there. That distinction is critical when you start thinking about why people like containers so much, right? If you think about what's going on within one particular host. Let's look at the picture on the right-hand side. You have a host with a base operating system and you have your container engine, Docker, which is what we'll get to in a second. That will be the manager of all the containers that are on the right-hand side of the picture and you see a similar model inside a virtual. Replace container engine with VM ware and on the right-hand side virtual machines. Similar model. If you think about what is going on under the covers. With machines you will have a lot more resources being used. Every virtual machine will have its own copy of the entire operating system and have its own instances of all the system processes that are running for every single virtual machine. That's a lot of resources. Imagine a world where you can take those resources, get rid of the processes and operating system itself for you to get your job done and you're just left with your application. That's what a container is. Now imagine how many more containers you could fit onto that host in terms of resource usage than you can with virtual machines. We are talking magnitude more containers than virtual machines. That the one of the big benefits of containers. They have smaller footprints than just virtual machines. All you are putting in there is the bare minimum to get the job done. You don't have to worry about all the extra stuff that's going on.

Now, because they're smaller and you don't have to start up the system processes just to start up the application you get much faster start times. With containers, you could typically get the container itself up and run than within milliseconds. Very quick. The rest of the time might be spent in your application itself starting. That's under your control. But in terms of the infrastructure itself with the container, milliseconds. It's very fast. Compare that with virtual machines. Typically seconds, minutes, maybe hours in some cases. There is really no comparison. Now, having said that in all fairness they're doing a lot of great work to speed up the startup times of virtual machines. Those two worlds from that perspective will get close over time and really interesting to see what the industry does with that flexibility. But for the right now containers typically do start up faster. Another selling points for containers over virtual machines. When you put all those things together, what we're really talking about is containers give you the availability to have better resource utilization for your environment going forward and that means cost savings, which is reduction of money and everybody likes that. That's at its course what containers are about and why people like them so much.

Having said that let's switch over to Docker. It's important to understand that Docker does not equal containers. By that I mean Docker is a tool to manage containers. Docker did not invent the container technology. What they did is they brought it to the masses. They did a spectacular job of making the management of containers really, really easy to use. And that's been their biggest selling point. They didn't actually create brand-new technologies. It has been around for a while. In a form that wasn't as consumable as it should have been. Docker brought it to the masses. What does that actually mean? So there are many different components to Docker itself. At its core what you have with Docker is they manage the life cycle of individual containers. What that means is things like they will do the basic crud, create, read, update, delete operations for containers or their processes. They'll manage the networks that are associated with your container and manage thinking about the container. What they'll also do is set up a file system on a per container basis if that's what you want. By that I mean every single container, while they're running on the same host, will have its own view of the files that it can see. Similar to a virtual machine in that space. I call it the file system aspect. We'll talk more about that later. Understand that Docker manages pretty much all the similar resources associated with containers that you will see with virtual machines. Just without the operating system being involved.

So if Docker is pretty much just this thing that just starts and stops containers and does your mounts and networking stuff that's all great stuff. But is that really enough to get people that excited about it? Well, no. That's where Docker is what I call secret sauce kicks in. I'll talk a little bit about the first big one. Their wonderful user experience. When you first sit down to play with Docker and once you get it installed and realize that with literally four small words on the command line you can start up a new container to run an application and you realize that you did that like I said with four simple words in the command line. You didn't have to read a bunch of documentation or run through a bunch of scripts or go through fancy -- four little words and bam you have an application up and running that someone else developed on Docker and you downloaded it and got it up and running. It is that ease of use that Docker as I said did a spectacular job on that made it so popular. It is incredibly easy to use. Not just from a simple command line interface. Even the rest API for those who want to talk through a program interface did a good job of keeping things simple. They wanted to focus on make -- enable people to get the job done faster and not worry about the infrastructure of the containers that are running. They did a great job doing that. That's the first big secret sauce.

The second thing, developing what they call Docker images. So you can think of Docker images as the container equivalent of virtual machine images. A snapshot of the file system that is inside of a container. That's all it is. A bunch of files in a tar ball. What Docker did is said okay, how can we make it easy for people to create these things? They developed something called Docker files. For the purposes of this discussion think of Docker files the equivalent of a make file. A set of steps that Docker is going to execute in order to populate a container, right? So whether you're just copying files into a container that's going to get a snapshot or run an install script that will download stuff and put things in the right spot and snapshot it, doesn't matter. Docker allows you to execute those various commands in a simple, straight forward way. They offer up -- it is quite limited. Speaks to their simplicity and once you execute the commands you'll snapshot the container. Take a tar ball of the file system and bang you have a Docker image that you can share with other people and they can run your application just by downloading your Docker image. What they've done is they've made the distribution of software really, really simple in terms of creation as well as sharing. We'll talk more about how they actually share that later. Keep that in mind they simplify the process. The third process is layers, it goes to Dockers' desire to optimize how it's being used. This represents the file system view of what's going on under the covers. So take a look at the first two columns there. Container one and two running app one and app two. One is using Tom cat and the second is liberty as its run time to host the application. Under the covers they were both built into Docker images and both built using a fedora as the base file system inside the container. As I said earlier, containers don't have to have file system files in there unless the application needs them. So for whatever reason this particular applications decided they needed fedora in order to have Tomcat and Liberty running with these applications. What Docker did, as the images were downloaded into the system Docker detected that those two Docker images use the exact same version of fedora, rather than downloading it twice have two copies on the system twice, it actually has just one copy of it there and it's being used by both containers. How can it do that? What it does is, it uses what they call a union file system. So looking at the chart horizontally what they have are layers, fedora at the bottom. Tom cat on top of that and app one on top of that. Each layer represents a direct tree in the file system. Docker will merge the three directory together its view of the world relative to the file system is a merging of those three layers. Container one and two thing they have their own file system but it is shared under the covers. Everything from the Tom cat and liberty layer on down to fedora is read only. If container one or container two need to make changes to any files in the file system regardless what layer those files are at. Those updates will be made in the top-most layer and the application will only see the newer versions of those files. So for example let's say app two decides to modify the ETC password file on the fedora layer it can do that. The modification is made up at the app 2 box. The file system will merge them together so app 2 only sees the modified version. App 1 never sees the updated password file. The original version back in the fedora layer is not touched. I went through that quickly and talked about it very fast but understand that this layering and shared -- shared layering approach allows Docker to squeeze more containers into a host to get better resource utilization than you might if you had a full copy of all those layers and all those files for every single container, right? So if you look at the example on the right-hand side, as -- if you look at the example on the right-hand side, as new versions of applications are spun up because they need to be scaled, if all those layers of the application are already there, Docker doesn't need to download it or make a copy of it. It can just reference the existing layers in place and spin the containers faster than it could from the first one. So it is a wonderful little optimization there and resource utilization. Finding the special sauce that goes into people sharing the Docker images. Docker has a Docker registry called Docker hub available on the web for people to use and it is free for them to use. And pretty much all it is is a data store. Upload the Docker image and people can download it. It's simple idea. Because it's free and the Docker engine by default will look at the Docker registry when it's trying to find an image. It means that people can seamlessly share their images with other people without people having to explicitly take actions to do things like download their image. For example, if I wanted to leverage or use say an image for something I can do something along the lines of Docker run UBUNTU. If I don't have that on my host already, the Docker engine will go to the Docker registry, Docker hub, download it automatically, put it onto my system and spin up a container. I didn't have to manually download it, find it or anything else. This goes to that simplicity of the user experience that Docker is shooting for. Again the Docker registry is there for ease of sharing and distribution of these images to make people's lives as easy as possible. Now, it is free for public images the minute you start talking about having private images that only certain people can download they give you one for free. After that they'll charge you. For public free images you can have as many as you want.

All right. So going forward, how is Docker created? It was started by Docker Inc. the company developed a tool to help manage the containers. They are interested in enabling people to get their job done in terms of the software distribution and development process. Docker has grown to be one of the most popular projects out there. I'm sure you guys have heard of it. It is important to understand that as popular as it is, it is an open source project. Not an open governance project. We'll talk more about that in a second. Even some, it has become the defacto standard for managing containers on a single host. I don't think anybody is competing with them in that space at this point in time. Because they became so popular so quickly they drove the Cloud Native frenzy that's going on. You can look at other projects that are really popular like Kubernetes and they owe their success pretty much to Docker for getting us kicked off down the path. Much to their credit they did a wonderful job. As I said, Docker is an open source project but not open governance. By that I meanwhile you can contribute patches and features and stuff to Docker, the Docker Inc. itself at left until recently for the most part had a fair amount of control over the project. If they didn't want something to go in it wouldn't go in. That's the way it is. A truly open governance model there has to be a path for anybody in the ecosystem to get into the leadership role to set the direction of the project. Docker wasn't there for the longest time. Between IBM and other companies helping to put pressure on Docker they've slowly started to change and become more of an open governance model. We'll talk more about that in a second. Understand that's why other projects like open container initiative and CNCF was actually started to help promote this idea of making sure that all of these open source projects in the Cloud Native space were truly open governance as well so one company didn't dominate that space.

So where is the Docker community itself? This gets into a fuzzy space. They are going through a bit of a transition. If you look at the main link at the top, that's probably the best place to start out your exploration in terms of where Docker is and to learn more about it. If you're looking at the source code itself there are two different places you want to go. The core engine itself and that's something that they now call MOBY. I have a link there. If you're interested in the broader picture, Docker community edition, engine and command line tooling and stuff like that you want to go to get hub.com Docker. Who is involved? A lot of the major players are involved or were involved at one point in time. Docker Inc., IBM, Microsoft, there are some players that were very active in the past like Google who have shifted to other projects like Kubernetes but still very much of a good working relationship between those communities or those companies even though they may not necessarily work on the same code base on a daily basis. There is a lot of harmonization and interoperability work going on.

There has been a shift in strategy within Docker itself. Aside from Docker trying to become more open governance what they're looking at doing is sort of breaking up -- actually let me back up a little. Let's talk about why they're going through a transition. A lot of pressure for them to have an open governance model. People want a path forward to influence the project itself. The Docker engine itself was one model excusable for the longest time. They decided in the era of micro services why would the container engine itself be this model? They wanted to break it up into smaller components for a variety of reasons. In particular so that each component could have its own life cycle. Reused for other purposes, people could pick and choose which components they want to install or package together. It made sense for them to break up this model.

So let's quickly first talk about what the Docker environment looked like and we jump forward because we're running low on time. Look at the picture on the right-hand side. Similar to the very first picture I showed. We had on a particular Docker host you have the kernel, Docker engine manages the containers. A cache of local images and Docker engine manages the ports and networking for the various containers. Then you have this Docker client that talks to the engines to get your job done. That's the original picture. At a high-level architecture that hasn't changed. What has changed is what's going on inside the Docker engine box itself. And as I said what they're looking at doing is break apart the Docker engine into individual little components. In particular what they're looking at doing is making those components reusable and more composable but at the same time shifting them over out from under the Docker Inc. umbrella of project to other places. We'll talk more about where they're going with that in a second. So let's talk about how they are breaking it up. A lot of this is -- let's talk about how they're breaking it up the bottom layer. You have the container itself in this picture. On top of that you have a project called run C. It lives in the open container initiative and sole purpose in life is to manage an individual container. That was the first step Docker took at breaking up its model. That was the initiative or the impetus to create OCI itself. After you have run C for managing a single container you have a project called container D which manages a group of containers. It manages a group of run Cs and that's the next layer of components they're looking to do. Container D was put in December of 2016 or close to that time frame. That's the next step in terms of the evolution of them breaking apart the Docker engine itself. On top of that you have other projects that make up the Docker ecosystem. Swarm kit for managing clusters of containers across nodes. Those are all being split on individual projects themselves and some of them will end up over in the CNCF and some someplace else and some my stick with the current umbrella under Docker Inc.'s code. it is a mismatch of things over there. Docker itself will decide what it wants to do with these things. The important thing to note a new project called MOBY. You can look at that as the home for all these individual projects together and MOBY will be a tool used to sort of allow you to bring these various components together into any kind of assembly or output product that you want. So on this particular picture look at the left-hand side. You can see all the various projects that can go into a MOBY distribution. All the various kits Docker is producing. Then through the MOBY tool you decide which of the projects you want to pull together and it will output some distribution package. That is exactly what Docker CE is. Now, if you don't want all the various components. Let's say you for whatever reason don't want the secret management stuff or something like that. You may be able to exclude that library from your distribution because you don't need it. You can do that. Docker has given you the flexibility to customize your container engine to do whatever you need and not have more than it needs to get its job done.

So the transition is well underway, as I said. Some of the things will end up in an open source/open governance home probably CNCF. Other things will remain under Dockers' control. Like the ACP server. They decided they will retain control of them. It doesn't mean you can't contribute to it. They encourage you to and should if you want to which is to understand it is not an open governance model. Docker Inc. will maintain rights over the Docker trademark and came up with a new name MOBY for the opening governance projects. It is not under their control anymore. IBM is using Docker in several different places. We have it in the Cloud that gives you a cluster of nodes that you can run Docker swarm customers on there as you see fit. That's one way you can use Docker within IBM and the IBM container service. A managed Kubernetes install that uses Docker under the covers as a container engine on a per host basis. It is deployed in a managed environment and the IBM Cloud private. Going forward what we will see is probably Docker get swapped out in favor of container D. In those environments you don't need the rest of the Docker engine. All you need is the orchestrator for the clusters of containers. You will probably see container D stop up there. IBM activities going forward or in general we've been involved in Docker for several years now so we have our hands or fingers -- fingerprint over a bunch of different parts of Docker itself. Going forward what we really are going to see through and to finish up are things like porting Docker to power and Z systems. Maintaining the core support is there to support other containers and operating systems. That's where you'll see our work going forward. Three maintainers, two captains and knowledge and skills to help you with anything that's Docker related. In terms of getting started here are some links for some information on how you can contribute to the process involved. They have a backlog of issues that you are more than welcome to take a look at to try to attack. People on the open source projects love it when people attack their backlog of issues. It's one of the best ways to learn about a project. A forum to ask questions and within IBM we have t earns. Education to help you learn about the technology walking through using the technology itself. We don't just give you a list of instructions or a blog. We walk you through it and explaining what happens every step of the way so you don't just feel a sense of accomplishment but understand why things are being done and how those steps were meaningful to achieve a particular bigger picture goal. It is not just about learning about using technology. It's about understanding how to use it in a particular use case for your job.

And then just to wrap it up final links how to get involved not just in Docker itself but in terms of conferences, discussions, IBM monthly newsletters and stuff like that for more information about Cloud Native. Lots of Lynx for you to get involved with. And think I just made it in time. It was a lot of information. Apologize for going so fast.

>> TODD MOORE: Very good, Doug. Very good. Any questions?

>> There was one, it was already answered, the difference between Docker CE and Docker EE.

>> DOUG DAVIS: Gentleman, so Docker CE is the community edition, as I mentioned. That's pretty much all open source code. Docker EE is Docker taking Docker CE and adding some proprietary bits to it, so, for example, Docker data center or Docker registry, that's what will make up Docker EE, enterprise edition.

>> All right. Sounds good.

>> JEFF: And lastly, Doug, I think it's also valuable to use the metaphor you touched on MOBI briefly for our audience, think it's fair they can think of, the way MOBI relates to -- is not unlike the way Fedora relates to the RedHat community and the REL, enterprise edition.

>> DOUG DAVIS: Yes, I think that's a great comparison. Thank you, Jeff.

>> JEFF: There's one more here maybe you can take it while you're getting ready for the next session. It says, Dockers mostly register Docker -- how well does it --

>> DOUG DAVIS: You cut out a little bit. Can you repeat the question?

>> JEFF: Does Docker mostly leverage Docker API? How does it handle compatibility issues with containers?

>> DOUG DAVIS: I'm not sure I understand the question. I mean, obviously, the Docker CLI would use a Docker API to talk to the Docker engine but I'm not quite sure what you mean by compatibility containers.

>> JEFF: I don't know what he means also. That's what he wrote there.

>> DOUG DAVIS: I take that up almost I think on the next talk which is on the open container initiative.

>> JEFF: There you go.

>> DOUG DAVIS: If by containers they mean runc and the lower level container itself, obviously, Docker talks to the lower level CIs or run CIs to get the job done so that's where the interoperability statement comes into play. Hopefully that will answer some of the question.

>> TODD MOORE: All right, so thank you. I'm Todd Moore if you're just joining us. I'm the host and we've been talking through a number of very good talks on containers and Cloud native computing. In this next session, Doug Davis, who is with us, and I'll introduce myself and Doug in a minute, will go on to talk about the Open Container Initiative. So, we're going to take a short pause here to give our folks who will do editing on video later a chance and we'll join up here in just about 20 seconds or so.

>> DOUG DAVIS: Can you guys see my screen?

>> We can.

>> DOUG DAVIS: Excellent. That means you can actually hear me, which is good. It's weird. I don't know what happened last time because I definitely did not unplug my head sets or anything like that. I don't know why you guys couldn't hear me, and Zoom did say I was unmuted so I apologize. I don't know what happened.

>> It's cool.

>> TODD MOORE: Okay. Why don't we get going here? I'm Todd Moore, your host. IBM's Vice President for open technology. I work out in a member of the initiatives surrounding Cloud native computing and with me today is Doug Davis. Doug is also one of the folks in IBM who spends a considerable amount of time out in the open source world working on Cloud native technologies. He's been part of the Cloud native computing foundation, Docker, Kubernetes, and a host of other projects that are Cloud native related.

So, in OCI, Doug is going to take us through how both as a companion project to run through runc and Docker and establish a base of containers, OCI has really come up and become the project that you probably haven't heard about but one that you may want to because here's how we're bringing together standardization of the container interfaces. So, Doug, why don't you go and -- for us.

>> DOUG DAVIS: All right, thank you, Todd. So, as Todd alluded to, the OCI or Open Container Initiative group is probably one of the more important projects out there even

though probably most people don't know that much about it. And it's important because it basically is the foundation for a lot of the Cloud native work going on there, not just in Docker but also Kubernetes, Cloud Foundry, it really is the lynch pin going forward and the fact that you probably don't hear much about it is the fact that they're doing really, really well because there are no issues so let's jump into it. There we go.

So, today we're going to talk about, what is the OCI, why it was created, give an overview of the community, where it's headed, what IBM has been doing with the OCI and then finally wrap up with some pointers on how you can get involved if you want to.

So OCI. Open Container Initiative. So, a long time ago, Docker realized that the management of a single container was not something that should really be under one company's control, meaning Docker Inc's control. So, they decided along with a whole bunch of other companies, including IBM and others, to extract this single container out from under Docker itself into a new project, into a new community called OCI so that it can really be developed by the community itself. And in particular, be reused by other projects aside from just Docker. Because really the idea about significant container is becoming sort of a commodity or sort of a boring infrastructure as people like to say.

It's not something you're going to get a whole lot of innovation around because it's just part of the native technology that's there. What they did is start OCI to do a couple things. First, to move that core technology, called runc, which is about how to run a single container out from under Docker so it can be managed and owned by the community at large. Take or to find the interfaces for that runtime, meaning the runtime specs, so how do you interact with the runc in a standardized fashion. What do the formats of the bits and side, even agreeing that it is a tar ball to begin with, all that definition had to be written down and agreed to. That's what the OCI's goal were.

As I said, this is a goal of making this infrastructure available to the broader community and not under one company's control, and what really does is then allow memorial to innovate on the layers above this, meaning container orchestration, which is what you see going on today between Docker, Kubernetes, Cloud Foundry, different ways of interacting with the container infrastructure and covers based upon your particular need and that's where people want to do innovation and try to differentiate themselves.

Okay. So as I said, OCI was created by extracting the core engine, the core container management system from the Docker engine itself called runc and then they took their specification for what a Docker image looks like and basically turned that into this generic container thing, not just from Docker's thing that this should be a commodity but they were also getting pressure from companies like IBM to convert nor into an open model because while they've been open sourced, they weren't open governance. That pressure encouraged them to make this happen. So the OCI is a Linux foundation project and does continue to be used by Docker itself and that's actually a really, really good thing not just for the obvious reasons but a lot of times when projects get split off from other ones, sometimes they end up becoming forks and then you end up having two versions of these things, the real one that's used in the original project and then this other one that sort of everybody else tries to use.

Docker didn't want that to happen so they made sure that from day one, runc continued to be used in the Docker itself. Use cases, make sure they didn't create this fork in the industry. Now, who is involved in OCI? A lot of the key players that you might expect. CoreOS, owned by RedHat, Microsoft is there. Pretty much most of the key players who want to host infrastructure or container infrastructure are taking part of it. So, where is OCI? Opencontainers.org is the URL. I list off the three projects. Starting from the bottom up, you've got runc, which is the actual implementation of that container run itself. The specifications for how to interact with it and then the image spec itself with the container image, so those are the three that are there. Now, in each of they had individual repos is actually like a project unto itself.

Now, overseeing all three projects, though, they have a technical oversight board and their main purpose in life was to get the entire OCI going. Once they did that, and three projects were underway, the TOB pretty much went dormant because they didn't have anything to do. There really wasn't much to discuss, they've gone quiet.

Now, since the OCI is kind of wrapping up its current version 1 of these three work items, they are starting to look at what to do next. And we'll talk about that more in a second, but, so they are going to come back to life. Now, however, there is one other aspect to OCI which is a certification working group, which IBM is co-chair of. They're looking at defining what it means to do certified -- how do you do implementations or distributions get in essence the OCI seal that says, yes, we are OCI compliant and compatible and you can be guaranteed a certain level of interoperability and harmonization across all the various that people claim to be OCI compliant so they're working on that.

Before we go much further, let me make sure I explain what

I meant by OCI and runc being sort of the lynch pin for the three main, most of the various Cloud data projects out there, right.

So, if you look at the various picture here, what you see at the base level of the picture, you have your container, which is pretty much a file system that was developed or put in place by an OCI image. Based upon your preferred platform, whether it's Docker, Kubernetes, all things will have the equivalent of a container infrastructure whose sole purpose in life is to really spin up new instances in runc which in turn spins up container. I didn't want to have either Docker or container D.

Obviously, container D, it's project here. But the point here is that all the major projects are actually using the same runtime under the covers and what that does is not just allows them to share code and development resources, but what it means is, you're going to get interoperability around the deployment of your application, meaning you can be assured that if you build an application from a Docker environment. If you take that Docker image, as long as it's OCI compliant, Cloud Foundry or Kubernetes, it should still work because it's the exact same runtime and that gives you or our customers the level of optimization that they're looking for. They're not locked into the platform, they can switch back and forth if they want to get the job done. It's about giving our customers the freedom of choice.

Okay? So current status of OCI. Version 1.0 of the runtime and image specifications were announced back in July of last year. We can expect version 1.0 of the implementation of the runtime very, very soon. In fact, I think as of right now, they're going through a vote to try approve it so I expect that hopefully in the not too distant future. IBM is a -- project and obviously contributed to the community. There are some sort of ancillary workings or projects going on around the OCI, for example, OCI tooling which is going to be used by a testing or performance group going forward, and that's still kind of under development.

Now, I already mentioned the certification working group. Related to that is sort of the trademark board and they're working to make sure that the trademarks and names are all used appropriately and people don't misbehave in terms of using those. You don't want people to be claiming to be OCI compliant when they're not really OCI compliant and the OCI technical oversight board, as I've mentioned, they've been pretty dormant because those two projects have been going along very, very smoothly, but now they're starting to look at what it do next. And let's talk about that for a second.

So they have this runtime, definition of an image, and specification of runtime, so that's all well and good but one project people are starting to look at would be really good for standardization next, and that's distribution. Meaning, we have this image but how do I go about finding it someplace else or downloading it from some registry because obviously there's the Docker hub itself, which most everybody uses, but there are other registries out there and some of them may have their own set of APIs and that's going to be problematic.

So, they're starting now the discussion of, can we look at coming to some sort of interoperability statement around how they're going to share these images and how we discover them. It's still very much a work in progress. There's no guarantee they're going to do it but there's a lot of desire to do so. In fact, all the major players who are involved in OCI have pretty much stated that they want this to happen. They just need to figure out the final little details around scope and things like that to make sure it's done appropriately. But, I do expect that to happen in the not too distant future.

Okay. So OCI with IBM. Obviously, because OCI is part of the core component of Kubernetes, Docker, Cloud Foundry, all of which can you get on IBM Cloud, obviously, it's a core component for our infrastructure. We in terms of development activities, though, over there, we're not doing a whole lot, mainly because they're kind of wrapping up their activities. There really isn't new intervention going on. Obviously, when they start getting into the distribution space I expect us to be very involved in that but in terms of current activities because they're sort of wrapping things up, there really isn't much going on there in terms of IBM being involved because there just isn't much to be done. Having said that, IBM is a maintainer on the project and contributing where necessary, and of course, as the certification, trademark working groups finish up their work, we will continue to be involved in that.

But from a technical side, things are pretty much wrapping up for the existing process. In terms of getting involved, the best place to start is with the OCI website itself. You can take a look at the GitHub projects under GitHub.com/opencontainers. We do have OCI channels that people hang out and chat about things, and from IBM's perspective, we've been pushing patterns which are educational workshop or self-driven workshop things. Because OCI is this sort of component under the covers that you don't really interact with directly for the most part, they don't have any patterns that directly talk about OCI components, like runc. However, we do have other journeys related to containers which will obviously use runc, and I pasted a link here to the container set of patterns that you can look at going forward if you want to start playing with those and just understand that runc and OCI is obviously a key component of what you're playing with going forward. We do have a newsletter, monthly newsletter. To keep up with that. With that, I'm pretty much done. This is one of the shorter topics because it's so well tightly scoped. So, are there any questions?

>> There are no questions, Doug, but I wanted to ask one myself, on the last session. In terms of time and space when we're reading up a container, if you are getting a new OS, does that get in close to the time of the DM?

>> DOUG DAVIS: Excellent question. So, it's important to understand that a container image is just a snapshot of the file system. It's a tar ball with a bunch of files inside, right? So when you start your container, normally you'll be starting off the processes for your education. Let's say concrete example. Let's say you have a Docker I mean only or container image and inside of there, it has Tomcat and inside of there, you have your war file for your application. Right?

Now, when that container starts up the container image will start up Tomcat, run the process to get that up and going, and that's it. Even though Ubuntu may actually be on the file system itself, the container runtime will not start any of the Ubuntu system processes unless you explicitly go out of your way to tell it to, which you normally don't need to do. So the start-up time is no different whether Ubuntu is in the Docker image or not because you're not actually starting any processes. The only impact you may have is download time. Obviously, if Ubuntu is part of a Docker image. Has contact with java runtime, it may take a while to download the first one because Ubuntu is a little bigger, but in terms of runtime, it's not going to be a difference because you're starting off the exact same processes and in both cases, you will be skipping the starting up of system processes.

So, operating system will take a little bit longer because it will have the system processes that it needs to start. Is does that maybe sense?

>> Yeah, it does. Got it. And the second question is in the OCI realm, what is VMWare doing there? Are they out of the game completely?

>> DOUG DAVIS: So, I think my answer should probably be, we have to -- VMWare, we can find out publicly. VMWare has not been a very active participant in OCI, as you might imagine. They're very focused on VMs. I think it's fair to say they're spending most of their time trying to speed up the start-up time of virtual machines because I think might have mentioned it on this talk r a previous one, typically VMs take longer to start up than containers but they are doing a lot of work to get the

start-up time of machines to be as close as possible to containers and I think that's probably where they're spending their time because they probably see some benefits of virtual machines over containers for whatever reason. So, I suspect their goal is to try to make things on par from a performance perspective and I think that's where they're spending their time, but this is complete speculation on my part and I'm probably speaking out of turn so let me stop there.

>> TODD MOORE: So, thank you very much, Doug. Another very informative session here as we close out day 2 of Open Source Week. Thank you all that have come and participated and thank you for the questions. We'll be, again, bringing up great topics to discuss tomorrow so please tune into the livestream and want to thank all of you, again, for coming and joining us. So, thank you, Doug, and this will end Session 4 for Day 2.

(Session was concluded at 2:51 PM CT)