Microservices, Kubernetes and Istio – a great fit!

IBM Code Tech Talk
August 23, 2017

https://developer.ibm.com/code/videos/tech-talk-replay-microservices-kubernetes-istio-great-fit/

>> MARC-ARTHUR PIERRE LOUIS: Good morning or good afternoon, wherever you are. We are really happy to host another Tech Talk in our series. My name is Marc-Arthur Pierre Louis, your host and moderator of this series, and this morning we have a trilogy of technology to talk about. We are going to talk about Kubernetes, which is container orchestrator. We are going to talk about microservices. And we are going to talk about Istio, which is a program that manages microservices in a secured way. And to wrap it up, we are going to talk about journeys that use these technologies. You are in for a treat this morning. And we have also a trilogy of technologies to talk about these technologies we have Animesh Singh, we have Tommy Li, and we have Anthony Amanse. These three guys are going to take us through these wonderful technologies. Without any further ado, I want to turn it over to Animesh for the continuation of this great talk.en Animesh, you have it.

>> ANIMESH SINGH: Thanks, and thank you for joining. My name is Animesh Singh. Along with me, I have my colleagues Tommy and Anthony, who actually work with me in terms of creating these exciting developer technologies around cloud and container technology areas. And we also have with us Chris Rosen, who is the offering manager for Kubernetes offering from Bluemix, and Nilesh Patel, who is the offering manager on the Istio side.

With that, let's dive deeper into the talk. The talk will essentially start, as Marc pointed out, going into the microservices evolution from there what's the relationship with microservices, the orchestrator, Kubernetes, and where does Istio fit into this picture.

Right now slides are available at the following link, which I mentioned, the Developer Journeys, which we will be using here, you can actually bookmark that URL and reach out there in terms of looking at the Developer Journeys which will be part of the talk. And if you have any questions post the talk, you can reach out to us on our Twitter URLs, which are listed here for all five of us.

So with that, let's go into a bit of the microservices. Essentially, as we are all aware of what a monolithic application is, and typically the modularity within a monolithic application is bound within the features of the programming language, for example, modules, packages, et cetera. When you need to scale monolithic

application, you essentially are scaling the whole instance of the application, which is not conducive. Also, if one of the processes, there's a small bug in your application, the whole application comes down. If the chance of one instance of the application comes down, the other instances will have the same problem as well.

More than that, when you are trying to update, even if there is a small update, you need to update the whole application, redeploy it, which is, again, something that is a huge cost.

Now, when you talk about an upgrade, for example, if you want to change programming language for a particular feature which is suited to a particular programming language stack or a cloud stack, in that case, you are going to upgrade the whole application just because a small change was needed.

So there's a strong cost of these monolithic applications to developers who are bound by your technology stack, bound by your programming language, and it's very hard to change from that perspective. So then came the microservices, and as the slide mentions, it's an engineering approach which focuses on decomposing these applications into microservices which are typically centered around a business capability. And you have strong and very well-defined interfaces for these microservices to interact. Essentially, the pay load on these microservices is JSON based. If efficiency is important, you can use options such as Google protocol buffers, et cetera.

Now, when you want to scale microservices-based application, you can only scale the part which is resource intensive. For example, if your customer data access is resource intensive, you just need to ensure that particular part is scaled. The same for updating. If you are updating a small microservice, you don't need to update the whole application, redeploy the whole application, and then, you know, the down time along with it.

So with that, let's talk about where container orchestraters fit into the microservices paradigm. Typically, Microservices are encapsulated inside containers. There is this one-to-one relationship between a microservice and a container. And most of the people who have started on their container journey, they start with one container, but at first your growth is easy to handle, but as the proliferation of these containers and microservices happen, it becomes very overwhelming. Right? We definitely need something which can manage these containers and microservices.

So there comes your container orchestrator. Now, what essentially is part of a container orchestrator paradigm? Some of the functionalities which you get as part of the container orchestrator are like scheduling, cluster management, service discovery, which a lot of the people think are the top three functionalities which should come as part of the container orchestrator. If you look at the container stack list,

starting all the way from physical infrastructure going to virtual infrastructure, like your hypervisors, then you have operating system, the container engines then go on top. Finally, you have the container orchestrators, and some of them which are pretty popular right now are Kubernetes, Docker, et cetera. And based on the survey done by devops.com -- and there are many surveys out there -- Kubernetes has definitely taken the lead in the container orchestrator space, and most have rallied around and are using Kubernetes for their workloads, which are based on containers.

So with that, let's dive into Kubernetes, into some of the details, and most of you will be familiar with what Kubernetes is. Essentially, it's a container orchestrator to manage your containers. It was inspired by Google's experiments internally. 100% open source. Some of the functionalities which it provides are scheduling, healing, horizontal scaling, et cetera. Something like self-healing, for example, if one of your containers dies, Kubernetes will automatically reprovision it. If the node on which your container is provisioned, that dies, Kubernetes takes care of it. In terms of what is autoscaling, you can do and set policies to scale based on real-time usage or resource usage. There is load balancing which is provided as part of the ingress, et cetera. And Kubernetes will also roll out new versions of your applications, and if something goes wrong in an automated fashion roll them back in. Last but not least, it also provides secret and configuration management so you can keep your configuration and secrets outside your application in a secure manner within Kubernetes.

This is a high-level logical architecture of what Kubernetes looks like. Typically all the interactions which you have from your CLIE, UIE, API, they essentially go to the Kubernetes master. Kubernetes master is the one which is responsible for controlling and managing the cluster, and then there are worker nodes, and the worker nodes are essentially the ones where your workload runs. On the worker nodes, you will find Kubelet, which is a Kubernetes agent which accepts all the commands from the masters. Then they have a network proxy which is called Kube-proxy, which is responsible for all the routing activities from inbound and egress traffic. Definitely the Docker, which is responsible for provisioning all the Docker containers which comes as commands from the Kubernetes master.

And some of the concepts within the Kubernetes architecture, we talk about pods. Pods are essentially the smallest units of deployment within a Kubernetes. One of the things which essentially pods do is it's a collection of containers that can -- that run on a worker node, and they share the namespace, network, host name, et cetera. And they are typically ephemeral in nature. Pods can come and go and die. If you want to expose your pods as a static entity or if your back workload is on Kubernetes, you can create a frontend service which can give you a static way to actually target the

backend, and behind the scenes, behind the service endpoint, your pods can come and go, but it's intracting the Kubernetes service.

Then there are application controllers which are responsible for maintaining the end state of your Kubernetes deployment. You can essentially create a template which tells what is going to be there and state that I might need these many pods, these many services, et cetera. And the replicate controller is responsible for ensuring that the end number of pods, the end number of instances, the end number of replicas you have specified, they are available, and then it's also responsible for scaling that.

But in Bluemix, we have IBM Bluemix Container Service, which gives you a managed Kubernetes service offering. Essentially you get a single tenant Kubernetes master, which is behind the scenes once go to Bluemix and provision a Kubernetes cluster. What you are getting is a Kubernetes master which is managed by Bluemix, and then you get work are nodes, and these worker nodes are all single tenant. They are -- in case of dedicated, they are single tenant top to bottom. That means the nodes, hypervisor hardware is all dedicated for you. What this does is you don't get any subscription, so there are none of the noisy neighbor problems which you typically encounter in a cloud environment.

As I mentioned, the master is totally managed for IBM for high availability, scalability, and upgrades. And the worker nodes are actually managed by the customer.you can scale them up, down, do capacity management, deload, et cetera. IBM also provides you checkers for an update. For example, if there are OS batches, docker in general, Kubernetes need to be updated, it's up to you when you determine to do the work or not. Et cetera. IBM doesn't have any data access to the data which is residing in persistent volumes or data which you are storing on worker nodes, so all your data is secure within this kind of node.

With that, let me go into some of the Kubernetes Developer Journeys which we have created. These are essentially on-ramps Developer Journeys within IBM help you be productive with our cloud, data, and AI technologies. You can reach out to them at developer.ibm.com/code/journey, and everything you need to solve problems are part of those Developer Journeys, and as we go further with the talk, you will see some demos in action for the Developer Journeys.

As I mentioned, some of the on-ramps we created around Kubernetes and how do you use our Bluemix Container Service, how do you integrate it with the pipeline are integrated in this journey, so I would strongly recommend for you starting or going to implement to go in and take a look at it in terms of how to get your workloads running on top of Kubernetes. So we have everybody's favorite starting workload is a WordPress, so we have a scalable WordPress deployment on top of Kubernetes. Where essentially we use MySQL as

a backend within the container, but if you want to get the production-level capabilities like replication, data backup, et cetera, you can use MySQL service from Bluemix as well.

Same with Git Lab. This has become very popular hosted repository on more than a hundred organizations worldwide are now certified Git Lab hosting providers. So from that perspective, you can use Git Lab to get started on Kubernetes, and again, with host SQL, which is a backend database for something like this, you can either run it within containers, consistent volumes, or better yet use a SQL service for Bluemix, which gives you all the replication backup what you will expect from a production-level database.

Cassandra, essentially NoSQL databases, as we are all aware, they have taken great traction, so this journey dives into how to get your Cassandra clusters running on top of Kubernetes. It goes into some of the core Kubernetes concepts like Kubernetes for services, so a lot to learn just from Kubernetes concepts perspective in this journey. Then we have some journeys around Kubernetes and microservices, which is the topic here we are addressing. So essentially, for example, a lot of Java developers, Spring Boot is very, very popular in terms of creating Java-based microservices application, so we have a journey in terms of how to get started with respect to Spring Boot applications on Kubernetes, and we also show within this application that if you want to call out some serviceless technologies from within a Spring Boot application, how do you actually do that?

This particular journey actually starts going into some of the concepts of Istio, which we will bring up later after the first demo, and why Istio does become important. But the core idea here is that you do have microservices-based applications which are centered in a particular language framework, so you probably can rely on a lot of language framework-specific capabilities to do some of the platform-specific things like service routing, resiliency, et cetera, but what you actually start getting into the polyglot world, then you do need to make sure that you are not rewriting that code in every particular language. It's often you are not packaging that with every microservice. You actually delegate that to a platform-level framework which can handle all those common functionalities which are consistent across all these different polyglot microservices.

Then we will go to the first demo which we have today, essentially, which is going to talk about MicroProfile-based microservices application. I was talking about Spring Boot and the evolution of Java EE, a lot of developers have actually started creating Java-based microservices applications, and a lot of other like IBM, Red Hat, et cetera. They were working in the background in terms of creating and coalescing around a technology called MicroProfile, which has just come out, and we'll see more of it during

our October Java 1 launch. The whole idea there is you can use MicroProfile essentially, and it gives some of the core capabilities which come as part of the Java EE framework.

With that, let me pass on to Tommy, who can actually demo this particular technology and how to run MicroProfile-based microservices on top of Kubernetes. Tommy?

>> TOMMY LI: Hi. Hi. My name is Tommy. So we go to our journey page, able to see the architectures of our journeys and all the technology we are using. From here, you could get the code and it will redirect you to the GitHub page.

So now let's go over to this journey. Our sample application is called Microservice Conference. It's build with a frontend and four backend microservices and stores our data to the database. Our frontend is (Inaudible) and our four backend microservices are within Java. They are all running on WebSphere Liberty managed by Kubernetes. We also use NGINX to do our load balancing.

Let's go over some of the files we use for this journey. So the first one is deploy for speaker. So here you can see the journal. You can see some, and you can change what image you are using like here. And we also have this for Microservice Builder, which it provides secure connection between the microservices and matrix.

Let's go over the NGINX. With NGINX, we will provide what microservice. (Inaudible). Therefore, when we deploy the NGINX demo files, we need to specify what source IP we should use. And our NGINX will route all our traffic to the node port 30856.

Now let's go over the deployment. So for the prerequisite of this journey, we want to have our Kubernetes cluster ready. So for our first steps, we want to clone our repositories. So since I have already done that, let's go ahead to the second step. Then we need to install the two add-ons, microservice fabric, which essentially helps a secure connection between the microservices; the ELK sample, which helps us collect metrics from the MicroProfile conference.

So next thing we need to do is install Helm. This is a Kubernetes package manager, which you could use it to easily install various add-ons from the Helm libraries. So once you install Helm, you want to run Helm init on your Kubernetes, and it will create a Helm service in your Kubernetes cluster which allows you to install various add-ons on Kubernetes.

So next you want to add the repository for the two add-ons and do a Helm install for both of the add-ons. Once you have done that, you should be able to see in this terminal you have fabric, kibana and sample elk installed on your Kubernetes cluster.

So now let's move on to the second steps. In these steps, you want to install Maven and Java 8 JDK, which you need it to build applications, and you need to run this script to get all the source code for our microservice applications. And we would also do Maven clean package for you.

Next we will move to the third steps. In these steps, you need to install a Docker CLI and Docker engine to build your Docker image and push it to your docker hub. Here you need to run these scripts with docker namespace, and it will build a docker image for you and push to your docker hub.

Now let's move to step 4. This step want to make sure Kubernetes is still running. So Kubernetes is still running. In these steps, we want to run this script to change our source IP and docker image name. So let's do that. And once you run this script, you want to make sure the add-on is running on your Kubernetes and the necessary jobs are finished. From here you could run kubectl create manifests to install all the necessary files for these journeys. And once you have done that, you could ruin kubectl proxy to go to your Kubernetes UIs and check your status of your applications.

So once you do kubectl processes, you would do localhost Port 8001 UI. You check and see if all the applications are running.

So now we want to the see what public IP we have. So for Bluemix cluster, you want to run the (Inaudible) with your cluster name. And this should be able to show you the public IP and private IPs. But since I think the Bluemix command is not working, I will have the IPs, so I just use this from here. So it goes to the appropriate IPs to port 30056. You are able to see your application is running. So it takes a while for all the microservices to port all the information. Then you should be able to see like the speaker just finished running and the session is also finished running. Now you can do votes. You should be able to go to the vote page and see what you voted from the session page.

Then from here you can also go to port 30500, go to Kibana and see what are your metrics from these applications. You go to Kibana, you see this application just started running, and you can see the details from here.

As you can see, Kubernetes is very easy. Now we will hand it back to Animesh to talk more.

>> MARC-ARTHUR PIERRE LOUIS: Before you go, Tommy, I have a quick question for you. So say for instance I wanted to do this myself, okay, and I go to your repo and look at the instructions. How long it takes me to get the whole thing installed and I be able to see the applications?

>> TOMMY LI: In general it will take 30 minutes because in the beginning, when you install the microservice add-on, it will take up to 20 minutes to install an add-on on Kubernetes, and you can take a break for that. Once you have done that, the rest of the steps should be able to be done in 10 to 20 minutes.

>> MARC-ARTHUR PIERRE LOUIS: So it's not a big investment of time, so I just wanted to tell the attendees that it's something that you can try, you know, try that and see the technology run. Thanks.

>> ANIMESH SINGH: Thanks, Tommy, thanks, Marc. If you want to

just get a feel, there is a deploy to Bluemix at the top as well which can be used, which essentially the whole journeys are integrated with IBM pipeline, which allows you to be able to then push these deployment on Bluemix by using the DevOps pipeline. If you don't want to run through all the manual steps, you can use that to tie in all the things.

Thanks, Tommy. I will share my screen now. So with that, we just talked about MicroProfile, and Tommy also brought up one of the things, Microservice Builder, that's another offering from IBM which essentially allows you to tackle the whole lifecycle of the microservices. So essentially, in terms of creating your microservices in terms of after you have created you should be able to deploy it from using the Microservice Builder, your microservices, you can add add-ons like ELK, essentially the Kibana UI which saw came because of that particular add-on, so you should definitely try it. And MicroProfile, as I mentioned, it's getting a lot of traction. IBM Red Hat and a lot of other vendors are involved. You have Tommy Li (Inaudible) which are a part of this particular effort. The initial baseline was (Inaudible) but it is evolving, and you will see a much more robust launch during the Java 1 framework. So for all you Java developers, definitely start looking at MicroProfile for creating your Java microservices.

So with that, let's go into some of the other areas. So we just saw -- we talked about microservices. We talked about the role container orchestraters and Kubernetes play in the microservices world. So it seems Kubernetes is great for microservices. But then why do we need a service mesh, and what is it? Right? So the whole idea with respect to I talked about some of the key functionalities which came as part of the container orchestrator landscape; right -- scheduling, cluster management, service discovery. But when it comes to more sophisticated features like traffic management, failure handling, and discovery, Kubernetes leaves a few things to be desired, especially when your application deployment are pretty big. Now imagine an application broken down into multiple microservices. Each microservice has multiple instances, and each deployed instance has multiple versions. Typically even a simple application with this kind of deployment model you are looking at hundreds of microservices.

So what else do you need? You essentially need a very strong visibility in terms of how the traffic is flowing between microservices, how you are routing traffic for microservices based on request, origination point. How are you handling failures? With this model when you have a thousand instances for an application deployed as part of microservices, you embrace yourself for failure. Failure will happen. Microservices will die. Is there enough strong capability either within your application or within the framework to actually handle those failures and in a graceful manner;

right? We also need a strong identity insertion between these. So service architecture was created to handle these. Think of service mesh with a network of interconnected devices with routers and switchers, except in this case, the network is at the application level. Other services and routing delivery and all other tasks are then offloaded to the service mesh. The main goal is to get from paint A to point B in a reliable, secure, and timely manner, and service mesh actually allows us to do that.

And how does service mesh do that? Typically, this is achieved by using proxies or to intercept all incoming and outgoing network traffic. Proxies in a service mesh architecture are implemented using the sidecar. The sidecar is essentially attached to the main application or main microservice and complements it by providing platform features.

With this kind of model, your microservices can use the sidecar either as a set of processes inside the same microservice container, but what we are doing within this service mesh architecture is these are all residing in a separate container alongside each of your microservices.

With that, let's go into Istio and what is it. So Istio essentially is an implementation of the service mesh architecture. It is created as a collaboration between IBM, Google, and Lyft. It uses a sidecar pattern where sidecars are actually enabled by the proxy and are based on containers. By injecting envoy's proxy servers into these sidecars, you are actually able to then intercept all the incoming and outgoing traffic. You are able to perform fine-grain routing. You are able to get more visibility. Some of the architecture components from the Istio control plane are pilot, mixer, proxy, et cetera.

Now, if you look at what Istio pilot provides, it is essentially responsible for propagating all your configuration to other components of the system, but more importantly, all your routing and resiliency rules actually go into Istio pilot. Then you have Istio mixer, which is essentially responsible for all the policy decisions and showing you all the telemetry data using Grafana, Zipkin, et cetera, and it uses a plugin-based model so you can actually provide your own custom backend to show you more visible data around how the traffic around your microservices is flowing. Definitely the proxy which we talked about which came as a -- which is called envoy in this case, it came from Lyft. They actually -- it's tested, used in a lot of the production deployment, and that's essentially provides all the sidecar features which come as part of this architecture, like load balancing, protocol conversion, et cetera.

So once your application is deployed, so you have the Istio control plane where all the control components are running. This typically goes on top of your Kubernetes cluster, and once your application is deployed, microservice-based application, that's in

the Istio data plane.

One key thing which I do want to highlight is essentially you don't need to make any change to your application to get all the goodies which a service mesh or Istio brings. Assuming that your microservice-based application is deployed, you can then slide in Istio as an add-on on top of Kubernetes, and you will start getting all the capabilities which provides around resiliency, traffic control, visibility, security, et cetera. I talk about how this is done. Essentially all the traffic leaving the Kubernetes ports, all microservices are transferred, rerouted by these proxies, which are envoy in this particular case. And these proxies are the ones which are responsible for implementing all the layer7 routing, enforcing policy decisions, et cetera.

So with that, let's go into some of the Developer Journeys which we have created around this combination, which is Kubernetes, microservices, and Istio. So one of the key features which I talked about what a service mesh or Istio brings is resiliency and efficiency. So this is a work we did very recently where we launched a journey around it, but when we talk about resiliency, Istio brings it to your application without any changes to your code. You can actually create circuit breakers which are responsible for ensuring if something goes wrong along a particular microservice path when you are trying to reach a point B, then there is a fallback plan which can be implemented. In this case, as you can see, for example, the circuit breaker is saying if there are a certain number of maximum reached, we should be able to put things in the waiting stage.

Some of the resiliency features that come with Istio are timeout, retries, health checks. And you can actually use systematic fault injection to actually test all these things. Istio comes and provides that.

So this is a Developer Journey which we just recently launched last week, which actually takes the same MicroProfile sample application which Tommy showed you, and then what we do is we bring the Istio add-on as a side add-on on top of that particular deployment, and then because of that, you start getting all these capabilities around resiliency with respect to this. For example, if cloud and database which is being used as part of this application is getting a lot of connections, you can create a circuit breaker which can say, okay, if there are more than X plus Y connections, then let's do a circuit break.

So essentially the idea is you can define how many maximum connections you want to have to a particular microservice, how many you want to put in pending, and if your load increases more than your max connection plus the pending connections, then all the incoming requests will be rejected. Right? So this actually gets you out of the box some of the resiliency features.

Similarly, for example, if you have a pod where you have two

instances of a microservice which are deployed and one of the instances is not responding, then Istio gives you capability to actually detect that in an automated fashion, create a circuit breaker, and eject that pod which is not working after X minutes of time so that your traffic is always reaching to the microservice instance within that pod or the set of instances which are working. This is, again, part of the journey, so I would strongly suggest you go and try that out. And definitely, if sometimes there are delays; right? If you are trying to reach either a microservice within that framework or if you are trying to reach a service which is outside the Istio control plane, and there are certain delays which are happening, you can configure Istio to tell that, okay, if a delay exceeds a certain threshold, then let's do a timeout, and for the rest of the course, we wouldn't be trying to load that back in microservice, which is slow to respond, again and again. We will have a phone-back which will respond and say the service is not responding.

Imagine now all this code, all this logic being in the microservices. You can imagine the bloat. And in different languages, you need to rewrite all this logic; right? So all this comes as an add-on with Istio so you don't need, as an application developer, to inject all this code within your microservice and you are just worrying about the functionality of which microservice is supposed to provide.

Then there is another journey we launched which goes into the other aspects of Istio, for example, traffic control and visibility, which are the key features. So as I was talking about traffic control, things like when you want to do traffic splitting, for example, if you are doing a deployment or rolling out a new version of a particular microservice or set of microservices, then you can selectively use how much traffic you want to read out to the new version. In this example, for example, as you see, if new version of service B is being rolled out, I just want to redirect only one person to the traffic to see whether it's robust enough it can handle the load, things are working before you actually start routing all the traffic. And not only you can do weight-based or percentage-based routing, you can actually also do content-based routing or essentially, for example, if the traffic is coming from an Android operating system or if the traffic is coming from an iPhone, if the traffic is coming from a certain geographic location, there is -- there are configurations you can do to provide logic to handle these requests differently, where to route them based on the requested content, whether it's a particular user, particular device, particular geographic location. All those policies you can define within Istio.

And the visibility definitely, as an operator, you would want, as a deployment person who has deployed the microservices-based

application, you want strong visibility into these traffic flow, what's happening with these microservices, how they are performing, how the requests are going through this fleet of microservices which you have deployed, so Istio provides you Grafana and Zipkin as add-ons out of the box. These are not the only ones. You can actually use Istio mixer to create your own custom backends. So essentially the whole idea is that all the traffic spans and all the data is collected by the envoys, and then they are sent to mixer, and mixer is the one which then has adapters for Prometheus, et cetera. And then from there onwards, you can see the data in the Grafana dashboard or the Zipkin dashboard when you are collecting traffic spans. But you are not limited to them. You can actually go and create your own custom adapters if you want to use some other backend to see the monitoring data.

With that, I will go into this particular journey and have Anthony from our team, who is going to talk how do you actually manage microservices traffic using Istio on Kubernetes. So this is journey which is published at this particular URL, and what we did was we took the Istio sample application, which came as part of the Istio release, and part A of the journey is essentially taking that sample application, deploying it on top of Kubernetes, and in part B we actually extend that application change, quite a few of the microservices, to connect to a database instead of storing data in the container, and then show because Istio brings you all these great features, there are some drawbacks or some extra work which you might need to do in terms of configuring, for example, based on the protocol, you are using to connect to an external service, whether it's JDBC, you will need to configure Istio a bit differently so as to enable outbound traffic. If you want to collect request traces and all the monitoring data around it. With that, let me pass on to Anthony, who will actually go through this step.

>> ANTHONY AMANSE: Hi. My name is Anthony, and I will show you a journey on microservices using Istio as the traffic manager.

So what are we expecting on this journey? For part A, we are using the simple BookInfo application, and we have four microservices, and the Reviews microservice is a Java application. It has three versions. And our product page is a Python app. The details is Ruby, and the ratings is a node JS app. So it is a polyglot microservice.

So for Istio, we can route 50% of the traffic to version 2 and version 3 or even by the user base, like for a user name Jason.

And for our rollout release, we can do 100% on our latest version of the Reviews service. You can also do access policy where we don't want the traffic from the version 3 of the Reviews to pass through the ratings. And for part B, we will need to do some extra work on the MySQL side.

And so now I'll show you where to get the code, and this is our

journey page from the IBM code site by clicking get code, you will be routed to the Git page. # let's start by the simple application without Istio. I already have Istio and Kubernetes prepared. And so let's deploy the BookInfo application. We are going to deploy it using Kubectl using the provided files.

And we should get the public IP of our cluster using the workers. It depends which Kubernetes app you are using from which service. You can use Minikube.

From there, we can access the application take note that we should always wait for the pause to be running.

So you will see that the stars changed because we have three versions of the review service. It's either black stars, red stars, or no stars at all, where you don't access the ratings service.

So let's proceed to deleting this application and applying Istio. So for Istio, we need to do Istio CTL Kube-inject. Take note that for this alpha release, .1.1, we are manually using Kube inject, and for a future release, we should -- there's a plan not using this at all, and it will automatically inject.

So now let's access the application using the public IP and the node port. You will see that it is the same application without any code changes. We are now using Istio, so what does Istio bring us?

Now let's proceed to traffic management. So right now we are using Istio to manage the traffic, and let's say we want to route all microservices to their version 1. Let's take a look at the yaml file. You will see that there is a tag called version and V1. It is dependent on the labels on our microservices when we deploy the BookInfo app. It has the labels version.

And let's proceed to routing a user named Jason. You want that user to see the version 2 of our application. Let's sign in as Jason, and you will see that he will only see the review service version 2, which is the black stars for the ratings. And that's it. Then let's go to a 50% to version 2 and version 3. Let's take a look at the yaml file for the version 3. So as you can see, we have the tag called weight. This means that 50% of the traffic will go to version 1, and 50% will go to version 3 so let's apply it using Istio. You will see version 3 of the microservices approximately 50% of the time and also the version 150% of the time.

So now let's apply the version 3 of the review service. Let's take a look at the yaml file. Now, for this yaml file, we are using version 3, and the weight is 100%, so we will only see -- we are only supposed to see the red stars. And that's it. This is really useful for our rollout release. It's either you are releasing to a subset of users or either a percentage of users. It also matches user agents such as to know which browser you are using. You can do that for a rollout release 2.

And now let's do -- access polls enforcement using Istio mixer. For this one we are using a denial, and let's take a look at the yaml

file.

So for this yaml file, we want to deny the traffic from the review service that has a tag called version 3, and let's take a look at the application.

Now we are not seeing the rating service because the Istio rejects the traffic from the version 3, but if we log in as Jason which uses the version 2, you'll see the black stars. Because the traffic is coming from the version 2.

Now, another benefit of Istio is it still can use the Prometheus and Grafana add-ons. You can collect metrics and logs. Let's deploy it using the provided yaml file. If you have a load balancer, you don't need node ports. It will be provided a public IP.

Now let's go to the Istio dashboard. You will see in the Grafana dashboard is the traffic requests. So now you will see the traffic request, and if you refresh the page multiple times, you will see which traffic is going to the version 3, version 2, or version 1. It depends on how you route your application.

So what else can we do with Grafana dashboard? Well, for Prometheus service, we can do response size, add more tracking services, track more data, and let's proceed to response size. We are adding a new metrics called the response size which collects how much bytes it receives from which service. Let's try to give our application some traffic. And now you'll see that it collects data of the response size which approximately shows one kilobyte per request.

And now there's another add-on called the Zipkin. The Zipkin collects traces from our microservices. This is our Zipkin dashboard, and once we get and request traffic from our application, you will now see that the Zipkin now has collected this, and if we click one of those, we will see how much time it took for the request to complete and how much time it went through the microservices. The details, the reviews, and ratings.

The Zipkin traces is really useful when you are debugging stuff and you don't know which problem and which traffic isn't going through the microservices. So for part B, we modify the simple BookInfo application. It uses external data source rather than static data. And Istio provides -- by default, Istio does not allow external traffic, so we have to modify some configuration. But first, let's edit the secrets yaml file. You have to encode it in basics, and I have already modified it. And let's just apply the yaml file. From MySQL, you can either use on-prem or external service composed for MySQL. And take note that right now we are using IP ranges 172.30. This is specifically for Bluemix clusters. We are telling you that Istio -- the Istio platform will ignore other IPs, and these are the only IPs that it will intercept.

We need this one so that our BookInfo application, our details, reviews, and ratings, can access the external data. So now we'll

first deploy the product page, and for the details, reviews, and ratings, you need to have the option for include IP ranges. This means that the Istio will only intercept this range of IPs.

So now let's deploy the details, then the reviews. Then the ratings. Wait for them to be completed, then access our application using the public IP. And now you will see our BookInfo application. You can post reviews. It saves data to our external service, external MySQL. And that's it.

I will pass this on to Animesh.

>> ANIMESH SINGH: Thanks, Anthony. That was a great demo. Hopefully you all got a lot in terms of staining how Istio works and the kind of functionality it brings to the table. So definitely I also see some of the questions which are coming. I mean, a couple of questions I answered by text. Some of the things are around support for position volumes using Istio. So right now Istio deployment are not using position volumes. That's in the works. What I would definitely suggest is if you go to Istio.io, there is a Google groups link, and you can join that, and they have a weekly call, and we can actually send the information which happens with all the community members where some of these questions in terms of, you know, when is a specific volume support coming, can be answered.

There was another question regarding does Istio replace Kube DNS? It's essentially using Kube DNS as part of the service discover mechanism. Goal of the Istio is -- definitely Kubernetes is a starting point. Kubernetes is a primary platform on which right now it's being provided as an add-on. But to be able to move it to other container orchestraters, like, for example, you know, if you have microservices running on cloud, et cetera, from that perspective, definitely all these adapters will tend to utilize the core capabilities of the platform which exist and then build on top of it. So yeah, as I said, Kubernetes is being used by Istio for some service administration and discovery.

Now, some other questions were regarding Node-RED and whether --

>> MARC-ARTHUR PIERRE LOUIS: Yeah, Animesh, wrap up that question because we need to add one more announcement for our next Tech Talk, and then your guys can remain on the Chat room and finish the questions. So you can finish that question.

>> ANIMESH SINGH: Yeah, so I think with respect to Node-RED, essentially Node-RED has adapters to push serverless open functions on top of Bluemix. As far as I remember, there is no direct integration where you can actually push microservices on top of Kubernetes. And the rest of the questions, definitely, we can take either later, we also have Chris and Nilesh who can take some of these questions.

If you want to reach out to us later, all our Twitter URLs are here, so definitely feel free to hit us with your questions. The

slides, as I mentioned, and the Developer Journeys are listed here to take a screenshot and hopefully, Marc, you are going to point to the next talk which is going to go into more details of Istio, so over to you, Marc.

>> MARC-ARTHUR PIERRE LOUIS: Thank you, Animesh, thank you guys, thank you Tommy, thank you Anthony for this great presentation. The video is going to be available so people can come back to it.

We are going to go on a weekly Tech Talk now, and that starts next week. On August 30 we are going to have another Tech Talk about the cognitive retail chatbot. You can register here at this link I am pointing at, and that will put the chat bot technology at your fingertips. And you can use that in the retail environment. So you can do customer product search, add/remove item in a shopping cart, all this in a chat pot, and you can do that and you can come next week on this Tech Talk to find out about this great technology. And presenters are going to be Scott DeAngelo and Rich Hagerty. I promise you that's going to be a great Tech Talk, so please make plans to be here next week on the same time, same channel to talk about this great technology.

Thank you for your time, and also we are working with Nilesh Patel, who works with Istio, to have a tech talk devoted specifically to Istio, so that way if you want more information, you can come to that Tech Talk, and that's going to be announced soon. Thank you for your time, and looking forward to talking to you next week for our cognitive retail chat bot Tech Talk. Thank you very much. Have a good day.

>> KATHY GHANEEI: Thanks, everyone. Good-bye.

(End of session, 11:01 a.m. CT.)