EclairJS Transcript

IBM developerWorks Open Tech Talk

Nov 9th 2016

https://developer.ibm.com/open/videos/eclairjs-tech-talk/

>> IBM's upcoming open source projects, which really, open source consumers and contributors, we feature these projects and make them available to you, and we highlight those projects that are really, show IBM's commitment to open source, and the smarts of our technologies who are really putting out phenomenal projects. We are recording this series and they will be available to you as soon as this call is over. At the end of the session, we will ask you to fill out a survey to allow us to better those presentations as we go along. Today we have an exciting project to show to you, a upcoming project. And it's EclairJS. I was at a open conference two weeks ago and the sessions about Node.js were standing room only. Node.js has caught the imagination around developers.

EclairJS has taken the benefits of that. It is providing JavaScript environment for developers with APIs to access Apache Spark, so that they can [inaudible] SQM machine learning and [inaudible] it is a great project.

(muffled audio).

That we want to discuss with you this morning. From the EclairJS team we have Brian Burns who is going to take you through this exciting project on the call, on the chat room is [inaudible] will answer technical questions. Without any further ado, I'd like to turn it over to Brian Burns for the continuation of this exciting talk. Brian, you have it.

>> Thank you. Thank you everyone for joining. (faint audio) I'll introduce EclairJS to you. I'm Brian Burns, developer lead on this project. I work in IBM's emerging technologies group, we are a forward thinking group. We are looking at technologies that are emerging on the horizon. We prototype ideas and make sure they are in line with where our customers are going. Here we have EclairJS that is one of those projects. It is a Node.js front end to Apache Spark. You can talk from Node.js applications to Apache Spark. We will go through the Web applications and data analytics as the world is right now, a brief overview of Node.js and Apache Spark

overview. We want to get to the heart of the problem we are trying to solve and how you talk to spark applications that are running, show a overview of how the life cycle of the spark application, then we will get into EclairJS, a intro, do a quick series of demos, then we will go over the EclairJS architecture and server requirements. We will show you how to deploy some of the pieces out on our BlueMix services, and how it can be used in notebooks.

Then we will wrap it up and talk about our direction moving forward.

Data analytics and Web applications, we have a increased need for Web applications to be driven by data analytics. Companies are generating lots and lots of data, whether it's from users in their transactions, sales transactions and resale, cell phone data, there is a lot of data out there.

Node.js is a highly popular scalable Web application platform. It can do, handle many connections at once. It is very good at asynchronous I/O, it is not good at doing computations because it's [inaudible] driven. Large computations are handed off to back end systems.

Spark, Apache Spark is a popular data analytics platform. It is growing rapidly out there now. However, it's very difficult to communicate with from a Web application standpoint. It is still kind of in the deal to do batch mode. EclairJS connects, takes Apache Spark, Node.js, puts them together so you can drive Apache Spark applications from Node.js and display the results.

Node.js is a popular service by JavaScript runtime which seems like companies of all sizes across industries is rapidly growing. We find a large number of JEE shops out there, and even .net shops are coming over to Node.js, and trying this out. Is it an event loop nonblocking asynchronous I/O. It is meant to handle many, many connections at once. You see this so it's becoming very popular at the front end, the Web application, ecosystem, because it is able to handle those simultaneous connections at once. Given that, it scales well horizontally out in the Cloud.

It's very favorable for the front end.

Spark, it's a distributed compute platform with an API centered around the data set abstraction. I'm saying data set because they are slowly phasing out, if you are familiar with Spark, you hear about RDDs they are replacing RDDs with data sets. That is why I'm saying data sets here. That will be the abstraction of the future.

The thing with spark is it has immutable memory data processing where work is distributed from a cluster of worker nodes, so computation is done quickly, because it is in memory.

It runs on the JVM. There are multiple language that are supported, scala and Java run natively, they are bind directly into spark, it runs because those are both JVM languages it runs seamlessly. Python and R are supported as well. They do proxy into the JVM, so your Python ends up running on the JVM eventually.

Why is it hard for Web apps to talk to Spark? One of the things we have to look at when you are looking at a Spark application, is the way it's set up and deployed. Spark provides shell script called spark-submit. It launches an application called the driver program, you can run it, you can tell it to run that program locally on your machine, which is called client mode, or if you have a, something like a yarn cluster in the background you can actually have yarn execute the driver app out on its cluster, so that would be called cluster mode.

If you can look at the diagram there on the right-hand side, you see the driver program makes a lot of connections. You have the cluster manager which would be equivalent of Spark master, this could be a grand master, a evo master if you are on BlueMix. It has connections open up to that, has connected to every single worker node on the cluster.

There are a lot of Web socket connections going [inaudible] in Spark program which means that it has to be inside the same local area network, as those worker nodes are on.

Typically the driver program, you might not want to put out there right on the front end of your Web application ecosystem as these things tend to be your clusters like that tend to be behind the probably a couple layers of firewall.

That is one thing to keep in mind. You have to have all the ports open for driver program to communicate with. That is one of the things that makes it difficult. The other thing is there are no, when you launch a program in spark like that there is no REST API to go and get results. You have standard in and standard out. But that is about it. What typically happens is, it's up to the actual Spark application itself to do something with computation. It can either put those results into a database, or K file on HTFS or object store or write to a message queue such as Capla or some other data store, you name it. That is typically what happens. You would have another application then retrieve the data from the external store.

The bottom line is you don't typically talk directly to your Spark application. You communicate with the things that it produces, like I said whether in databases or in message queues or other things.

That is part of the reason why it's difficult. This is in general Spark in general, this is kind of a known thing. We are

going to try to address that. EclairJS, what we are doing is we are trying to combine nodes asynchronous I/O based Web front end with sparks machine learning and distributed capability. (background noise) this is a powerful combination. You are able to utilize nodes' ability for thousands of current connections at once and be able to do things with Apache Spark, and do heavy-duty data analytics computation.

What we provide is we provide a SPM module called this the client side, this is a API in JavaScript and it mirrors the Spark API you might see in Java or Scala.

There are certain things that are very JavaScript specific, but that makes sense, is where JavaScript is not a type language. So we provide that API. With that API it is basically across those, it acts almost like a RPC if you will. We send commands to execute to a server environment (background noise) and those are all done over a Web socket. There is a single Web socket from your Node.js application.

(ping).

Back to the EclairJS server. As far as the server is concerned we provide a Docker container for trying stuff out. It is a Docker container with a Spark environment in there, running in local mode. There is nothing robust in there. But it is good to kick the tires and try out EclairJS with.

We are a open source project on GitHub. We have been open source from the start. We are Apache licensed. We have a developer license.

It's easy to get up and running and start running the examples. Basically, we provide some examples right at the top of our repository of how to get things going. We can run in the Docker container as the server at this point, just to try things out.

(coughing).

Excuse me.

Basically, what you do, go into the repository, examples directory, do a MPM install. That will install the module dependency to run the examples. Docker container is going to start up the server. You will run any one of the examples. I'll show you some of the code here of what it looks like.

Is it pretty straightforward.

This is your typical, this would be a typical hello world Spark application, where on basically, all I'm doing here is creating an array of five elements, distributing it across the quote-unquote cluster, although here you are in local mode.

You are calling a mapping across that data set and adding one to each number in the set, and returning the results.

If you see here, this is running, this will be a Node.js application, up here at the top, I don't know if you can see

this at the top line where my cursor is, this is where we require the EclairJS module. The next thing we are going to do is create an instance of that module here. Then now if you are familiar with the Spark API this should look familiar. Now we are creating a Spark session. This is Spark syntax, previously you might create a Spark context directly, but with spark 2.0 we have the session interface now.

This creates a session. Your Spark master is going to be local, because you are not providing one here so it's to local. Once you do this, now you are instantiating a connection or our implementation goes back and starts a connection with the server.

Again we move on and we actually start creating our data set. This is a typical Spark API here. There is nothing different about it. You might see this, you could see the same syntax if you were in Python or if you were in Scala or even Java.

This syntax should look familiar. It is a one is to one mirror pretty much. We can deal with directly with JavaScript arrays, we don't have to create a new list, if you are in Scala or new array list or if you are in Java we use the native languages syntax here. We created this array.

Now, if you notice, we call, we want to map across this set. We want to add, we want to have our lambda function, referring to these as lambda functions, because these execute out on the cluster. This doesn't actually execute in the same scope as the driver, it's out on the cluster where all the work is done.

That performs the results. Spark handles this for us, makes sure the executives get the things to do. After that, once we have the data set we want, we want to do a collect. This is where we start to diverge a little from other languages. What we are doing here is that collects, that is in Spark called a action request. Everything that is done up to here is lazy. Internally what happens is you are building up these data sets or RDDs, whatever you want to call them, what is happening, is it's building a internal, something of a Dag for lack of a better word. It's not until you actually want to get data, that it actually executes that Dag. It is a lazy oriented framework. Here we do a collect. That is when things are going to happen on the server side. Spark is going to execute this plan that it created.

What we do, to be good citizens within Node.js, we return a prompt. A standard [inaudible] prompt what we return. [inaudible] provide when it's ready function, we define it here. We get the results. Here we are logging it out to the console.

As far as the JavaScript, the syntax of the JavaScript

code, we do make use of promises and we make use of (coughing) Excuse me, make use of promises and we are able to pass in the JavaScript lambda functions that we are actually able to execute out on the worker nodes. Later on I'll get into a deeper dive of the architecture and go over how that is done.

Why don't I show a quick demo of this actually running. We have also out there in our, under our EclairJS organization name, under GitHub, we have an EclairJS examples repository, where we provide more concrete end-to-end demos, one of the demos I'll show you here is a sales demo. What this is doing, this sales demo, back here, what the sales demo is doing is actually, it's taking sales data from a fictitious store, a year's worth of data from a particular store and it's building a linear regression model based on the sales and the price and the number of units sold of a particular item.

What the demo is doing is streaming in, it builds this model, now it streams in data for sales that are actually going on for that day. At the end of the day, what we want to do is look at the number of units that were sold, and then we want to test the data for that day against the model and to see how accurate we were, to see if the prediction is, how close it is from the actual what really happens.

It's a kind of complicated demo. There are several pieces that actually have a server running where we are going to pump data into, we are going to pump data into capita, this is going to be our streaming piece. We are going to write some records and we also have a start-up our server environment here. Here what I'm going to do is actually come in and start up the node application that is our app. There is a lot of moving parts right here, and because this is a demo environment, there is a good chance things don't work. But if you are familiar with doing this stuff, it's usually the case.

But I won't be negative. I have some hope. We want to wait a little for this. This log here you can see this is our server side. You can see all the code coming in that we are executing over on the server.

We should get some streaming results coming in fairly soon. Once that starts, I'm going to go and load up here, I'm going to load up the actual application. We have a bunch of stores in the New York City area. I'm going to click on a store, see if we can get data coming in.

What you are looking at here is basically what I described on the right you have a bar chart showing the sales data coming in hour by hour. We are playing this through quickly to illustrate the point. This is fake data. The data is being pumped in. We are trying to illustrate the point of a real Node.js Web application at work here. This is all written in

Nodes. The UI is presented like you would at any other HTML interface in Nodes.

There is a hundred percent Node.js. What we have is we have that Web socket communication coming from the server, we are sending Spark commands to execute. We are getting the sales data in. Once we get a full day here, what this application does is from the Node.js side is sends the data over to the calls predict on our linear regression model. This is using the sparks machine learning API. We call predict on that model to see how far off we were.

The orange line represents the actual sales that are going on. The blue line represents what the model had predicted for that day. On some days it's on, some days it's off. But the idea is what you would typically do in a real environment is run this and you would continually test your model against what is actually happening. Then you would have the opportunity to update that model with new data to make it more accurate.

Again this is just merely a demonstration purposes, but I think it tries to drive the point across, that we are trying to get at, that you can build whole Node.js Web applications using this platform and talk to Apache Spark directly.

I will stop that demo now. We will go back to the slides. I'll actually show you, we will get into a little bit more of the architecture. There we go.

Okay. You can look at the code here, you can follow this URL and go directly to the code and actually see the, what is going on, and you can actually follow the regular Node.js Web application.

The architecture, we make use of, if you are familiar with Jupyter notebooks this should be familiar to you. We are using, we are leveraging Jupyter as a back end server. We make use of something called the Jupyter kernel gateway which is a, we either run on the notebook server, used for serving up notebooks. We can run on the Jupyter kernel gateway which is a slimmed down version of the notebook server without the notebook UI, because you can almost think of the EclairJS as a headless notebook. The Node.js app is acting like a notebook would in some senses.

You can follow, you can read up on that here with the Jupyter kernel gateway and documentation.

We also are using another product that was created by IBM emerging tech and that is Apache Toree kernel. When you are in a notebook environment you have a process of a kernel, kernel is responsible for executing the code that you want to evolve. You might have kernel for different languages, Python kernel, name your language. Somebody has written a kernel for these days. We have already written our own in emerging tech. It's called

Apache Toree. Toree brings in Spark into the mix. Toree has a interface where you can plug in your own language interpreters, and what we have done is applied a JavaScript interpreter.

(coughing).

Excuse me, to Toree.

We make use of that. What we do, that is JavaScript interpreter actually runs on Java 8 Nashorn engine, JavaScript interpreter built into the Java 8 runtime. We make use of that, so those lambda function that is get executed on the executors get evaluated in Nashorn by us. We provide a framework to make that happen.

We also make, in the client, EclairJS client, we use Jupyter provides a Node.js module called JS Jupyter services. What that does is allows, it is a convenient API for talking past the kernel gateway. We sit on top of that. When we want to send code to execute with our API wrappers, we go down into the JS services library and send that code over the wire using that module.

Again that is an open source, that is open source is where you can check it out there, the links on the slide.

A graph is worth 1,000 words. You can see what the server environment looks like. Over on the left, you can see here the Node.js applications. You can actually run this using the BlueMix SDK for Node.js. I can get into that a little later, I have a slide on that.

The green here represents our artifact. The client module which is a MTM module running in the node app, now on the server side, you have the Jupiter kernel gateway with Apache Toree. There is a local Spark distribution on that machine, that Spark distribution actually launches Toree. Toree is acting actually in this case like a Spark driver application.

The code that we send is actually ended up being executed on Toree remotely.

What we basically have done is using Jupyter, we have built an RPC-like environment, where the Node.js value would be the client, Toree side would be the broker if you are talking in traditional RPC sense.

From there, it is pretty much straight up Spark communication, so Toree is the driver application. It communicates with the Spark master. That is the Spark cluster manager on the right. That [inaudible] work out to the cluster.

We also have an EclairJS jar file. That jar file handles execution of the JavaScript lambda functions out on the workers.

I'll get into that in more detail in the next slide.

Spark takes care of a lot of, as far as distributing the work. That is what spark does. Toree is our, the way we actually execute the code that we want to execute from the

Node.js side.

To reiterate too, this server and infrastructure, we provide this in our Docker container. We don't have separate workers out. Having is run in local mode, in that Docker container. However, IBM analytics actually has their Spark service and if you look at your diagram here, this is from a 50,000-foot view, this is what their current architecture is.

We are able to run in the Spark as a service environment, in our development version, and active development right now but that should be supported coming soon, because this is the same environment that we run in.

So, to take a little bit of a deeper dive on there, you can see that on the Node.js side, you can see the EclairJS node API. That is our Spark wrapper API. Once we get that, you can see some of the code we might do. Let's say we have a data set, you want to do a filter on it.

You might do something with the code on the right. What that does is that is the Spark API. Then we take that code and we actually send it into JS Jupyter services to be executed over on to the Apache Toree side. This is really the Python protocol, again going back to the notebooks, this is the protocol. It ends up being a JSON object with a code snippet to be executed on the servers, so we are wrapping it up, JS Jupyter services does this, wraps it up in this code, JSON object here.

This gets sent down into eventually the kernel gateway, the kernel gateway routes the code to be executed into the kernel, and that kernel being Apache Toree. Apache Toree delegates that to our JavaScript interpreter, that is the EclairJS interpreter. We can start to provide a series of, if you look at the Java API they provide a series of function objects. They have function objects for handling flat map test case or regular functions for handling one argument, two arguments, three arguments, etcetera, pair RDD, they provide all these function interfaces. We actually implement those interfaces. Our implementation takes a sterilized version of the lambda function that we sent over from Node.js.

This, once we are a Spark function implementing their interface, they can then, Spark can take that and distribute it out to the worker nodes. Then on the worker nodes, our jar is out there on to the workers, and what the worker nodes does is our implementation of those functions actually gets executed. What happens is that function gets called. We take the stringified serialized version of the JavaScript function and avail it on the Nashorn JavaScript interpreter that comes with the JVM. We got the results that were returned from invoking the Spark function and Spark takes care of the rest. It comes back through, the results go all the way back up and eventually

when you do a collect on it, we get the results back and send it back to Node.js.

We do take care, there is a little more going on here. We do take care of the serialization of objects coming in and out. You have to remember we are crossing the boundary from Java to JavaScript. We are serializing parameters and result types from Java to JavaScript and JavaScript to Java and vice versa. That is part of what our EclairJS Java wrapper does. That way, the JVM, that way spark can serialize those results back and forth because having is done using serialization.

JVM serialization from the worker nodes to the clusters, that is how Spark is implemented. We have to make sure that everything is serializable and can move freely throughout the cluster. We take care of that. We make sure that happens.

I mentioned a little of this already. We are able to deploy on BlueMix. The BlueMix provides right now there is what they call a SDK for Node.js, this is a, basically a node environment, so you can take your typical, you would build your typical Node.js application, from the BlueMix console you could create a instance of this, create a instance of the application. You would build a man test file which is typical of deploying these things in Cloud environments. Would you do this in BlueMix or any other kind of path environment it should be familiar. All you have to do for EclairJS is, as long as our module is in there, it will work fine. The environment variables that you set, tell it where the server is. If you are on the BlueMix environment, this should be done for you. If you find the IBM analytics spark as a service to your node app that will be done the environment variable and we know to look for that environment variable.

Otherwise we would set a environment variable to point to where your server is. That is documented, that is on our wiki.

I mentioned the IBM analytics for Apache Spark. The port for that is coming soon. We have that working in development branches. There are a few things we need to touch up before we release that out into the wild. But we will be, keep a eye on that because that will make deploying our stuff easier. We will be doing some blog posts and putting up demos of that fairly shortly, I would say within the next week or so, we should be ready there.

That is exciting, because it gives us a place to actually launch a server environment where you don't have to actually deploy a server environment yourself. So you are stuck with the docker container.

If you want to deploy your own server environment, we do have instructions for doing that out on the wiki. It is essentially installing the Jupyter notebook server. Spark

cluster, however you want to instantiate one, there is a million different ways to do that. It is basically having a spark cluster, having the Jupyter notebook environment and making sure that story is a kernel that is available. Once that is done, we move on. So there are instructions for that on the wiki as well.

By the time that we actually run on a notebook in server environment, you can actually execute JavaScript code from a notebook if you wish. Remember our Node.js application is basically, think of it like a headless notebook client. It is not that much different. If you are running a notebook server on the background, you can choose to run, you can actually open up a notebook and you would choose the Toree kernel in our config in it, and that will ends up coming up with the JavaScript interpreter. Yourselves you can start writing JavaScript yourselves. Notebooks are popular for data scientists. They are used mainly for cleaning and transforming data, numeric simulation. Data scientist might take a sample of data, play with it a little bit, make some graphs of results and use it as a reporting tool to show the results of their analysis.

This is one benefit that you actually get from the EclairJS server, you do get that notebook environment. It is not our, this isn't our main goal here. Our main goal, our focus is on the Node.js side, not necessarily on the notebook side. But it is a benefit of the environment if you wish to take advantage of it.

In conclusion, this shows that Apache Spark, we can use it to connect to Node.js Web applications development. [inaudible] distributed compute platform. Next I would suggest go to our repository, clone our repository, look at the examples. Run the example applications. Developer contributions are welcome. We have an ICLA contribute to the project. But we are certainly open to developer contributions, even if it's, at first if you don't want to sign the ICLA, try it out, if you find any issues, open up some issues. More than welcome to do so. We can communicate with the issue tracker. We can communicate on our mailing list. We have a Google books mailing list, we have a Slack Channel you can join. There is multiple ways to communicate with us. Where everything is open source, so we do all our development in open source, so, yeah, come on and check it out.

So that will basically conclude the presentation.

>> Thank you, Brian, for that presentation. I think you sparked also some interest as you went through this EclairJS and what it proposes to do. I wanted to ask a couple of questions in the realm of machine learning. What APIs that are available

that EclairJS makes available to people for machine learning?

>> BRIAN: Everything that is in the Spark API, we have wrappered most of those APIs. If you go to the Apache Spark, look at the documentation for machine learning, we do provide an API for all of those. If you look in our examples directory, when you clone the repository, and that examples directory there is a machine learning directory, and all the examples that were implemented, we basically implemented all the Java examples in JavaScript. So we ported them all over to our API.

You can take a look there, for example, of local machine learning things. But it is extensive. We concentrated heavily on that to make sure that we have all those APIs working.

As far as, the only piece of this Spark API where we really haven't gotten into yet would be the graphs, the graphing API. But as far as streaming Spark sequel machine learning, we have most of those APIs implemented.

>> Great. Those examples are available, people can just model after them and create and see the power of the [inaudible] with those examples.

>> That's correct. They will be able to look at the API. Those examples will demonstrate the API that we have implemented of Apache Spark. They should be able to run those examples against the Docker container.

There is a run script in there, in that example directory. Make sure you run that when you are running against Docker container so it knows how to find the data.

>> Great.

>> BRIAN: It's up there.

>> I tried it on ubuntu. It worked. I loaded the Docker container and tried to put something that you provide. What other environment are these available on?

>> BRIAN: You can run, if you want to get the server environment up and running, you can run everything locally yourself. As a matter of fact, this is why, if you look when I ran that sales demo, I actually have the whole environment running locally on my machine here. I can run the Jupyter notebook server and I have a kernel installed on this machine. I'm able to go into the notebook environment and run everything.

There are instructions on our wiki. We have a wiki page. I think it is building your environment is up there, there should be a link there which you can follow of how to do that. On whatever machine you can set that up, you will be able to run that way. Coming soon you will be able to deploy the IBM analytics Spark as a service right from the BlueMix console. We can run directly on that. That should be coming fairly soon. Hopefully, we have time, I can actually try that out. I don't know if that will actually work. But we can give it a shot.

>> You can give it a shot. You can share your screen and give it a try. As you are switching to your screen to show that, and I wanted to also take the time to invite the participants to our next call, which is going to be on November 30, it's going to be about Rosie Pattern language. I was at the ATO two weeks ago and I saw the presentation. I was very impressed by the Rosie pattern language. I didn't know that there are some regular expressions that you can run and those can take quite a bit of time. I tried them on Node.js and some of them have taken 60 seconds. Some of them have taken 130 seconds. It is quite a simple regular expression. So Rosie will give you a better expression especially when you are going through lots of data and you can pass through the data through the Rosie pattern language. That would be something to attend for our next tech talk which is going to be on November 30.

Rosie pattern language, I learned a lot about it, and you can go to our website and see my blog and see my experience with Rosie. Are you ready to switch, Brian?

>> BRIAN: Yeah. I'll grab the screen here. You see my screen?

>> Yes, I can.

>> This, I'm into the BlueMix console here. I've restarted the service. There is a notebook here I keep for looking at the log files. This documentation here in the Spark as a service BlueMix had a look at the gateway logs and look at the kernel logs, so I fired up this notebook basically to wake up the system.

Here, I did switch to our development branch. I did a MPM install on our development branch, where we have the BlueMix stuff happening. I'm going to run this example from locally, typically this node example would be deployed out on BlueMix itself. But I'm going to run against the server. I have this V cap services environment variable that I'm going to actually source. That should enable the server to Node.js client should pick up the variables there, be able to connect to the service. I have no guarantee this is going to work. We are in heavy development right here.

I'm going to run that simple JSON file. Where is that thing?

>> Examples?

>> You know what, I gotta check that, oh, I know why.

You can see here, I'm giving the harmony flag to make sure that we run, that node actually executes in [inaudible] mode. Jupyter JS services module uses a lot of exo 6 syntax like especially the abbreviated function syntax. Make sure you pass that harmony flag in.

You can see this long URL, this is the Spark as a service

URL I'm going to connect to. We get a kernel back. Let's see if this is going to work. Let's give it a couple seconds.

See if this is, it's going to bootstrap, Spark as a service, once it goes to sleep for a while you have to kick it to get it back to back alive again. I haven't used it in a couple days. Let's give this thing a couple seconds to see if it actually comes through. What we can do from the notebook side is we can actually see some of these cells. You get this virtual sandbox, Spark as a service. You can actually get to the log files here. You can take a look at the logs, to see this, actually starting a kernel. See if it came up okay. Okay. We have a kernel.

Started our Toree kernel right here, let's look at the log, make sure that is okay. I'm going low level here. Remember we are in DEV mode. It's starting up. It should be okay. I don't know why it's taking so long.

>> Cutting edge.

>> It's cutting edge. Probably not going to be too happy right now, but again it's under active development, and we will get there.

>> All right.

>> BRIAN: But trying to run with the same thing I was showing you, I showed you earlier in the presentation, that is basically your basic [inaudible] trying to create the numbers data set and doing map over it and print out results, nothing crazy going on here. But in the meantime as we get this developed, we will get this working fairly shortly. The goal is, the demo I ran earlier, the goal is to get that working where everything is deployed on BlueMix.

So the data we want to use all the BlueMix services, so we want to use object store for where Spark is going to load the batch data from to build the model. We are going to pump that data into the other BlueMix service called method chop, basically Kafta what I was running here locally. We want to bind to IBM analytics Spark as a service. We want to deploy that node application up to BlueMix using the Node.js SDK. That is the goal. That is what we are trying to achieve right now. Hopefully, fairly soon we will have that working.

>> Great. All right. Thank you, Brian, for that presentation, and that [inaudible] available after this call and people can download and replay it and see what Brian went over.

I'm going to ask Kathy to put the survey up so that people can give us their feedback so we can better operate our conditions as we go forward. We thank you for your time. We are waiting for you for on November 30 for that great call about Rosie pattern language. Thank you very much. Thank you, Brian. Thank you, team. We will talk later.

>> Thank you.
(end of meeting at 10:50 a.m. CST)