Implement computer vision for PlayerUnknown’s Battlegrounds live streams

IBM Code Tech Talk

November 15, 2017

https://developer.ibm.com/code/videos/tech-talk-replay-implement-computer-vision-playerunknowns-battlegrounds-live-streams/

>> This morning we are going to talk about PlayerUnknown's game, this is a interesting topic this morning. And to take you through that, we have two technologists who are familiar with this, who have worked with this, we have Cullen Taylor and Spencer Krum who are going to take you through that. Without any further ado, we are going to turn it over to them, sorry for the little late in starting, but I think they are going to make up for that. Cullen, you have it.

>> CULLEN TAYLOR: Cool. Everybody see my screen?

(cavernous audio).

Welcome, everybody. Thank you for joining, Cullen Taylor to talk about Rotisserie. We are developer advocates in the digital business group at IBM. We are focused on developing

open source software around gaming and sports. We speak at open source conferences and attend conferences and talk about our efforts both in booths and stand alone talks. We have a background of DevOps work.

>> I think you have to switch inputs, man.

>> Hmm?

>> Are you trying to show slides right now?

>> CULLEN TAYLOR: Yeah.

>> You are not showing the slides.

>> Share screen.

>> There we go.

>> Awesome.

>> CULLEN TAYLOR: We have a background in DevOps work in the community and at IBM, and we specialize in container technologies, like Docker and Kubernetes, most of our work is focused in building things around those two things.

(very cavernous audio).

(.

That is a Rotisserie. But also Rotisserie is a Web app for passively viewing active streams on twitch TV, we will get more into the red zone concept. It is written almost entirely in Node.js, runs in Kubernetes [inaudible] we will talk more about that as well. It is still undergoing early active development. We are reaching 1.0 about now. It focuses on a single game but

we are thinking about porting it to more games as time goes on.

What's with the chicken? You might be asking yourself that and that is a good question. But what the heck is PlayerUnknown's Battlegrounds? Or public G, I'm not going to say PlayerUnknown's Battlegrounds because that is a mouthful but it's a hundred [inaudible] it was developed by a Korean game developer in conjunction with a [inaudible] or player unknown. It shares DNA with arma 3 [inaudible] it refines concepts and entered early access this year on March 23. It gained traction quickly, sold 25 million copies to date and it's [inaudible]

The basic mechanics are that you fly in a plane over this big huge 8-kilometer by 8-kilometer island. Players jump out of the plane and parachute down to the ground anywhere they choose.

What happens is you have nothing on you, so you need to constantly scavenge for weapons, and armor and backpacks and whatever else you can find, med kits, anything, to help yourself survive. You can queue solo by yourself which means you go into a game of 100 people and it's you on [inaudible] or three or four person squads of 33 or 25 each.

The last person standing wins and it literally flashes on the screen, winner [inaudible] where the Rotisserie name came from. When you die, you are dead. You requeue and go to the

game. You don't get any responds. Once you are dead, you are dead. That is it. Players are forced to cluster together by circle on the map, draws over the map that you see here. That constantly shrinks so you have to keep running towards the circle, in order to survive.

There is the winning screen.

Here is an example of those shrinking circles. You can be here, and you want to be here to be safe. This blue circle will close in on you, and say if I was over here, then I would constantly take damage until I'm dead. That is a background on the game. That explains the game. I want to make sure that everybody was up to speed. I know not everybody might know exactly what the game is but that is the 10,000-foot view.

The idea of this software project itself stems from me and my colleague Spencer Krum who will talk to you in a little bit, wherein [inaudible] a tiny town in Sweden [inaudible] basically, we are there for this thing called dream hack. Dream hack is this thing that you see at the bottom. It's this huge party, thousands of people go, they all bring their computers, they all play for days on end. They host big tournaments for all the big video games there. We are there, enjoying it, doing research on [inaudible] we are sitting in a sports bar type place [inaudible]

(very cavernous audio, I can't understand).

We see on the TV that instead of showing traditional sports or soccer or something like that, they are showing [inaudible] another video game out there. The way that they are doing this is that they have the bartender's laptop hooked up to all the TVs in the sports bar. Every five minutes, he has to constantly find a new stream, he is looking on streams on twitch and he has to manually go through and see what might be a interesting stream. It's tedious. He has to do it every five or ten minutes.

We started thinking about, how could we automate this. We were drawing on napkins, like you would have to look at any one of these things outlined in red to find an interesting stream of counter strike. It could be this team name, it could be amount of money, it could be how many rounds they have won, it could be what round it currently is in the middle, it could be the time left. There is even more than that. But we started thinking, this is going to be a large engineering effort, and we are going to have to look at all these different things and process this data. It's ridiculous.

What if you had to look at one number. That's that number right there. This is the only thing that matters in pub G in Battlegrounds, the only thing that you care about if you are looking for an interesting stream is how many people are alive in the game, because the video game when you have 99 people and

they have all just spawned into island and dropped dead and they are winning it's a little boring, it's dry. You see them picking up weapons. They might run into one or two people. Usually not. You see them looting or driving in a vehicle across the map, and that is not fun to watch.

We thought about how we can make it more interesting to watch. You flip through streams that are always at a low number of players alive.

So I'm going to see if there are any questions in the chat, just real quick. I think we are okay. I'm going to continue on.

The high level overview of the project is the user hits the app UI and the app UI is showing you the result of all this stuff going on in the background. In the background, all this is deployed on Kubernetes. We will talk about that in detail later.

But all we do is make a twitch API call to get a list of streams. We record some footage from each of the streams. From there, from getting some footage of each of the streams we will take a screen shot, so we have one frame of that video, we will crop it down to just that number of players alive, and we will do object character recognition on just that cropped screen shot of just the number.

What happens is it will return that value of that number

back to us in plain text, and we will associate that plain text with the name of the stream, put them all into array and sort of by the numbers, and the one that comes up after sorting, that comes up at the top of the array is considered the best stream available to watch.

This is some of the code, we are going to go through the app. We start by grabbing a list of streamers. This code has changed because twitch released a new version of their API and sunset some features in their old API, spuriously. But anyways, we get back JSON payload of streams. We parse through that in Node.js and we filter by English language streams because that box over here, if the game is not being broadcast in English language, this will change based on the language. Based on that, it can change the position of where this number is because it changes the size of the box, it's really annoying.

So the idea is add multi language support at some point. But right now, we have just filtered by English language streams which most people [inaudible] English anyway so that is not too bad.

We filter out streams that are flagged as for mature audiences because we want this to be a truly passive experience. We want people to be able to throw it on their second monitor or whatever, and deal with let it sit there and

constantly flip through streams. The problems with streams flagged for mature audiences you have to manually click a button that says yes, I understand, let me watch this stream. We don't want people to have to constantly look at the second monitor to do it. It's there to be there in the background.

We parse through the JSON payload. We limit it by [inaudible] 20 or so streams and describe the name of the stream out, the streams out from the list.

From there, we go to record stream. There is no real good solution in Node.js for recording streams that we know of right now. That might be another good point of development for us to do and tackle. But there is a Python package called live streamer which makes this easy. For each stream we currently spawn a live streamer process to record a second or two of footage and then kill it.

This piece of code here shows it spawning a live streamer process giving it our twitch token which is stored as a environment variably they are in the containers or you can run this on a bare metal or VM, however you want to do it. We personally run it in containers.

After four seconds, it will go ahead and kill that process. It will spawn out this without a message to the console. With reason we keep it alive for four seconds is so that it has enough time to handshake with twitch and start writing to the

file, because if you don't give each process four seconds, sometimes it cuts off too early and you have a corrupted video file. So you won't get footage out of it.

From there we take a screen shot. We have one raw frame of data that we can use. From there we use a open source software called F FM [inaudible] makes it super easy, take the frame at the beginning of the video. Now we end up with this, like I showed earlier. Now that we have done this busy work, we have this list of streams and we have got a single frame of video from each of those streams. We have to crop the streams down. We use image magic which is another open source piece of software that we utilized. There is also a Node.js wrapper around image magic, and so that is just included as a dependency. We crop it down to just the number of players alive. That is always there no matter what, even if your map is up, even if you are aiming down your sights, no matter what you are doing, as long as you are in game, that number of people alive will always be there. So we can rely on that.

Now we have a list of streams, it's first come first serve. It's completely unordered. We have got a cropped screen shot with just the number of players alive. Now we need to get the value of that number from that cropped screen shot.

We, Spencer, you want to take over from here?

>> SPENCER KRUM: Sure. I can talk about it from there.

>> CULLEN TAYLOR: Go ahead.

(loud typing).

>> SPENCER KRUM: Can I share? Yeah. All right.

Hi, everybody, my name is Spencer Krum, I work with Cullen on the projects. This is where the computer vision kicks off in the flow of the program. We created a, there is a request.post there. What we have done is made a microservice out of object character recognition. Tester act is open source and there is, I don't think we have a code example of it again but on our GitHub repo there is a application, small file and a Docker container that goes with that. There is a microservice just for optical character recognition. You submit PNG files like that and it returns the results of what it thinks the text is in there.

Inside of that, we created a process pub G. If you look in the middle here, right there you can see that there is a process pub G sub, sub URL. We actually wrote some of the logic that we need specifically for this task into that route on the microservice.

The number has a natural bounding of what it can and should be. It should always be above 1 and less than 100. If we get any value that isn't in that space, we get F or 300 or something like that, we throw it away and write it in as a hundred.

That basically compartmentalizes our logic. We record 50 streams, we get 50 screen shots. We take 50 screen shots, we get 50 cropped images. We get 50 integers. We put those into an array. We sort the array. We have this value. That value is this sorted array of tuples of stream name, and it's number of people alive. The rest of it at that point then is all on the front end. The way that the front end works is it will, has a piece of jQuery and every 15 seconds it will fire up, and go back and hit that all end point and look at which streams are available and which aren't.

Let's look at what it looks like. We are going to talk about the Kubernetes part a little later. But for right now let's look at what it, the app API end point looks like. What that looks like is, and if you have got questions, we are happy to answer those questions. The first question that is coming out is, what happens if the number changes in the recorded four seconds? Well, yeah. There is a delay between when the app finally knows what the number is and what we got. When you actually watch this application like if you go to this URL, what you will get is you will see that the number displayed by the app and the number alive is often a little different.

But they are close enough that it works. This end point we are going to use dash S to make [inaudible]

(loud typing).

You pipe that through JQ. You get this nice streamer, 47 alive, that is the URL for the player. The way the UI works is it takes that UI for the player and puts that in an iFrame which is how twitch wants you to embed streams. There is a JavaScript time out, that is 15 seconds passed, time to hit the all end point, find the first one and switch to that.

If you look at the head here, head is the UNIX command that takes the top ten lines of the file. Right now it's impact QT and they are at 14. It's going to switch to impact or it's going to stay on impact. We wrote logic in the website there for that.

Another question, if you have multiple similar alive counts do you have secondary logic to filter [inaudible] that is a super good great we. You get two people who are in the same map so they always have the exact same number, or they are playing together or both streaming or playing doubles or something.

The way we are doing that now is sorting alphabetically so you get a consistent user experience. We have talked about sorting on player health, sorting on gear. Once we have an application set up that allows us to crop specific parses, places in the UI, and then go back, so for anybody that doesn't -- oops -- oh, no. There we go. Let's go back to this. This is what the UI looks like. We can grab the player

health which is the bar down here. We can grab something from the mini map. We can grab some of this information.

There is lots of opportunity to get more context out of the game. As Cullen was saying in the beginning, one of the reasons we gravitated towards PlayerUnknown's Battlegrounds is because so much of the most important information is summarized out of this one, 15 by 15 pixel box and that is a high contrast. That is a good feedback. Higher viewer count is nice because so the twitch API does expose how many viewers such and such has. You can swap based on number of viewers.

But the only thing I'll put on there is that one of the desires we had for this, if this became popular, I don't know that this is going to become crazy popular, you can imagine a hundred people are watching at a given time. The thing literally looks at every twitch stream or at least I think it's going to. Right now it looks at the top 50. But you can imagine a 17-year-old or 10-year-old streaming pub G for three friends and he is in the top eight people on his game and he looks at his twitch feed and he's got 150 people watching him because he is on Rotisserie. He is like ah! That could be a exciting moment to get streamers that don't get a lot of attention to get them more viewers and make them excited to do it. That could be really fun.

What if a streamer has a overlay on top of the remaining

player count, even though that is unlikely because [inaudible] it totally happens. There are a couple streamers that have, I don't think we have anybody that literally has a block of overlay on top of it to hide it. But we have people that have a rotating thing or sponsor logo back there and screws it up.

Unfortunately, we are happy to work with any of these people, we have a black list on the app that says we will never show that person, because the count for that person is never accurate. Unfortunately that is what we had to do. If the Rotisserie had a little speed, everyone would naturally stop using it, because they are throwing away twitch.

It stopped obfuscating that number because they are throwing away twitch viewers. That is a good question. Solid user name, ten out of ten user names right there.

Let's go back to, I guess we should show the app. It will take me a hot second to show it. Let's go ahead and do that real fast. Just a second while I figure out how to do this.

This is what the app looks like. It's pretty solid. Spencer is not allowed to write JavaScript, that is a important part of what is going on. We have the next closest down here and the top up here.

I don't know why this person isn't actively playing. They just died. So we are going to expect it now that they have 91 people, we are going to expect that to swap over pretty much in

the next ten seconds or so. Then because we are showing ads we are going to have to watch an ad. Paul asks, any thoughts on how you catch someone who is overlaying a false number to confuse the game viewers. We will deal with that problem when we get there, but yeah, the [inaudible] between people who are abusing automated systems and people using automated systems is a fun one, it's a important part.

Two things to get out of this, are the number at the top, this gets back to the question, net smile, about 36 and 31, so it's been a few seconds. So we are off. You can see it switched up to this guy at 22. One of the reasons you see it oscillate between a lower and higher number is that the optical character recognition is not perfect.

Sometimes that 22 gets misread as an F or something, because OCR is from the '90s and not good. If it's misread, it gets automatically written in as a hundred. Sometimes you will see oscillation. That is one of the places we want to improve. There is a couple things here, pin stream, view the code, fork on GitHub, pin stream because of the logic for switching it's implemented on the front end in JavaScript. You have ability to do things like pin, I want to stay with this guy forever whether he dies or not. View the code will take you to the GitHub repository. Fork will let you [inaudible] technical deep dives coming out of our technical staff in the next couple

days. This is what the app looks like. It's not complicated. Because gamers like to be in dark rooms sometimes we implemented dim the lights so your eyes aren't burned by the white light.

That in place, what I want to talk a little bit more about is the code. Are there any questions before we move on to the code and the Kubernetes components of it? I don't think I can see the -- oh, no. Okay. I can see the chat. Right. Alex KV asks did you train OCR model for the game, the OC model we are using is Tesser act, it's a off the shelf thing you can get from open source. One of the places we want to go for this, especially considering that we have, we work at IBM, we have this baseline of where it is without any kind of like deep learning or machine learning, right.

What we can do is train our own model to do the OCR. We can look at apples to apples, are we doing better. That would be a fun place to take this I think next.

I suppose there is no API in pub G service where you can get that information directly? As far as we know now, there isn't. We talked about E-mailing blue hole and can we do this directly with a API because that would make this app easy to write. But in a way, part of the fun of this is doing the OCR and building a application to do that stuff.

Certainly, the infrastructure we have built now where we

have a automated pipeline where you can get a screen shot out of a twitch stream and slice it and do something with it, that can be used for anything, any time, there is numbers all over these video games. Tracking that kind of information over time would be an interesting thing to do regardless.

Let's go to the terminal, this is the, we talked about what this API end point looks like. There is also a current end point which we will show you the top one. That is how we started implementing the thing. The all give you as much information as you can is good. One of the things to do is implement pagination, so that you don't get the whole thing if it's really long.

There is a couple caveats to this application right now. We are scratching a lot of files to disk. We are not making it as microservice-y as we could. We could make it a lot better.

Robert asks is there any number in the IBM, which I don't understand. Maybe someone else can follow up with him. Yes, net smile says, twitch has a option for people to host other user streams, if you can filter such duplicates.

So yeah, I don't know. If someone is hosting, they won't show up on the API basically. The program flow starts with us getting, hitting the twitch API and getting a list of twitch streamers. If someone is hosting someone else, they won't show up in that list. So that will never happen. It's [inaudible]

iFrame jam, we can use the hosting functionality but they rate limit it.

Unless we want to, we can rewrite the app to be a hosting thing but we host every 15 seconds which is way less than the rate limit for hosting. You are allowed to host twice every hour or something like that. Good questions. How many viewers are we getting on the app itself? We haven't done work to make it public yet. [inaudible] we have basically zero viewers on the app. What we did do and this is fun, if you are familiar with the Apache benchmark tool, it's a thing.

(loud typing).

It will let you evaluate, is your website performant so we can do Apache benchmark and, 1,000, 100 so that will basically, at a hundred requests, concurrent requests, it will do 1,000 requests and try to see how the app performs. You are able to tell that immediately that the long, we had 0 failed requests, and 1,000 complete requests, and the longest request took 645 milliseconds. Even though we have zero users now we are confident that this could scale up to more users than we are going to get. The hardest work is twitch. Twitch we embed a iFrame and somebody has to do video, stream literal video from twitch servers. There is a element of twitch, need to have the servers for that, but that is not our problem.

Good question.

There is no, Brian asked if there is any nepotism available for IBM streamers streaming pub G on twitch. The answer is no. We are not going to add special rules, so streamers that we like more end up higher in the rankings. We are a impartial and very responsible application developer.

Like I said we deployed the thing in Kubernetes. We used the IBM Kubernetes offering. It's available in IBM Cloud. Cubectle get nodes, we have a handful of nodes. We have two nodes. This is running version 7.7.4 that is like the API server I guess or possibly the version of the cublet and there is version. I have a older Kubernetes client but we are at 1.7.4 plus whatever hacks IBM has done to get it working. 1.8 I'm sure is on the pipeline. Kubernetes 1.9 is going to land relatively soon. I just learned last night that one of the cool features that Kubernetes 1.9 has is Kubernetes secrets will be encrypted at rest which is pretty exciting.

It took us a while to get the application up in Kubernetes. The team, there is three or four of us sometimes, we had to learn a lot about Kubernetes in order to get started. But that is kind of fun because it gives you the opportunity to learn.

What I want to talk about is the first minimum viable Kubernetes deployment that we had. Then which we don't have now, and then once that is well-established, I'll talk about the, where we are at now because the complexity jumped in order

to get to where we are now. We will talk about why that is.

The first thing to talk about is what is your deployment pipeline from a DevOps perspective, like what actually causes the code to go to production. The answer is we have this make file. The make file, we route it once and kept iterating on it. The first thing we do is scratch the actual short version of the current shot so git rev parse gives you this B 2A, it's like the last six -- oh, it's the last six characters of the [inaudible] the point is every time we do deploy or generate Docker containers or whatever it is, we want to make sure they are tagged with a specific number that we can use to identify them uniquely versus everything else in the environment.

What we will do here is, make file, and then we got a couple things. We create the rev. We have a script called make image.SH that builds our 3 Docker images. We have a Docker image for three different things. We have one for the OCR microservice. We have one for the app itself and one that is just a engine X in a static container because we want to serve things like HTML JavaScript that doesn't change, jQuery, that stuff. We want to shift those through. That one is cool.

You can look at images static server Docker file and you can see we are doing this nice copy, and that copy is basically taking everything that is in the public directory of the git repository, shoving that into a specific place in the engine X

image and engine X job is to serve those files, this is better than having Node.js and [inaudible] files over and over again. Use the thing for the right job.

Gary Horn asks how flexible/easy would it be to use this for another game, in the future could the user choose a game and select [inaudible] right, this thing is kinda cool. It immediately opens up your eyes, all this cool stuff. The framework of watch streams, pull a screen shot and something, something crop and go, that is super flexible for everything else. You would have to discover where on the screen the pixels, the number, whatever you care about is, deciding whether or not OCR is good enough to do it and persisting that data somewhere and ranking.

The next couple stages we want to go through with this application, one of them is figuring out how much we want to [inaudible] instrument it with a time [inaudible] so we have all this stuff moving around and we can watch these numbers change, stuff like that. We can bring in more metrics and build a synthesized number of what is the most exciting stream. It is not just number of people alive, it's how many stream viewers you have, how much health you have, information about the map. Once you build that framework, that translates directly to counter strike or whatever you are trying to do. That is a good question. That is where we want to take it.

The number one thing we want to do is evaluate whether or not our own model for doing the optical character recognition increases accuracy enough to make it worth it. That would be cool stuff to do as well.

Omar asks, is implementing on all [inaudible] we are using OCR which I think was MLed or something at some point. But [inaudible] OCR but in like the very near future there will be new Watson natural scene text that is coming out eventually or beta like quarter 4 this year, and then training our own model with something like Tensorflow to just do a specific task we want to do is definitely on the roadmap. Now that we have a baseline of what it does with just OCR, we can write on a graph somewhere like this is the jump in accuracy we saw, as soon as we trained our own model. This allows us to evaluate models. When we trained it this way, we got this accuracy bump. Trained it this way, we got this accuracy bump. We can do apples to apples comparisons.

Niles asks, sorry I tuned in late, did you talk about the hardware infrastructure where this is running on? That is what we are getting at right now. Most, the part that switches, the part that plays the video is all in your browser but this is hitting a API end point which you can see there. That API end point is served by an express, Node.js express app that is running in IBM's container service in Kubernetes.

(indecipherable).

You can see we are running some pods. Brian Gleason and somebody, oh, yeah, found a bug, if streamer is expecting another player, they come to the top, yes, so you can queue in PlayerUnknown's Battlegrounds with two or three buddies. When you die, you can watch their screen. What we found is most of the time, a squad will have one streamer, not two streamers.

Yeah, it's technically not the person playing who is streaming but you are still watching a game that's got six people left, and you are watching the player view. When you are spec take iting another player you are not given more information than the player has. You are locked to where their camera is pointing. It's essentially the same view. It has been been brought up we should try to filter it out but it's still exciting so I think it makes sense.

Yeah [inaudible] open source, if you want to take a stab at writing something to detect when the spec Tating box is up there we are happy to look at the requests and get the functionality working. At the least, we can annotate the API, because if you look at this, this API, we have this object, which is stream name, updated, we can add spec Tating equals true. Then the UI could decide what to do. People could have a check box, like don't show me spec Tating people, only show me live people. You can make it better without being

prescriptive about what the user needs to do with it.

This make file, we make the images, which is nice. Then we tag the images, if you are familiar with Docker you tag images. We push them into a registry. We have [inaudible] where our Docker registry is because we switched registries a couple times. We use the IBM container registry. We use Docker hub. We go all over the place. The fun part is the deployment.

Down here there is make role which is like do everything. Typically, me, I'll merge somebody's request and type make role. That will do all the image building, all the image publishing and this Kubernetes deploy which ends up being this line right here, on the other side, kube.

(indecipherable) standard Kubernetes, the way you [inaudible] it turns it into a object and submits that to the Kubernetes API.

Specifically what we are doing here is taking dash F which is the file name and the dash which is the UNIXism for standard in which is why there is a pipe right here. What we are doing is we are taking this, I don't know if this W is supposed to be there or I added that, we are all using vim. Anyways, I don't think that is supposed to be there.

What we are doing is we are taking this file which is where we have our Kubernetes YAML and processing it with M sub string. M sub string is templating for bash., it takes a

couple environment variables and finds those strings in the file it's processing and expands those out. After we started using Kubernetes we needed higher level tools than just a bunch YAML will provide. We needed to have the same deployment. We need it to be checked into version control but needed it to be slightly abstracted so I can deploy to my cluster, so we can swap out Docker registries. The most important thing that happened first was we wanted to deploy always with the image that we tagged.

We didn't want to screw with latest pointers or screw with any of that. We said we will always create a image with the same 6 letters in it which is the short, it's a tag, right, and then we will go through our Kubernetes file and swap out -- this is a Kubernetes pod. This line from 1 to 21 is the minimum you need in order to start something in Kubernetes. This is called deployment. It will create a pod. You see here we are using image tag. We are not submitting something to Kubernetes with this dollar sign in there. We are using M sub string to process this file, replace that image tag with something like some short image tag like AF, I guess it will be -- well, it's going to be like AF. It's going to be like something dash AF. Or maybe it will be just AF, so it's always unique. We can always get back to what it's doing.

But then that leads us nicely into deployment. We started

out with a simple deployment with three pods, three cluster IPs services and one ingress controller.

I'm going to take you through exactly what that looked like. All of this is checked, YAML is checked into git but the make command is run on somebody's laptop. We don't, we still don't have some kind of DevOps pipeline where as soon as the code merges, it's immediately picked up by a robot and deployed to Kubernetes. I lied a little, we have two secrets, three pods, three cluster tags.

Let's look at the secrets really fast. What you do is write out, scratch out a file like this. This is not checked into version control. This is something you need to know to do but it's documented.

Kubernetes has two different secrets, it has these types which are opaque which are passwords and key phrases and tokens and stuff, and what we need to do is put two different tokens in for accessing twitch because that is the way the application begins. It begins with a authenticated kernel request to the twitch API. That returns the list of streamers who are streaming.

There is a second token you need which is used for actually pulling that video off of the twitch, like if you want to pull a specific stream that needs to be authenticated. You have a different kind of token.

What you fill in on these all caps is base 64, echo, my password, and I almost shouldn't show you this. So that is wrong. And this is right. The difference there is, one, has a new line at the end of it. Echo by default will put a new line at the end of it. It is easy. We did it all the time to take this thing and put it there and then be confused why your app doesn't work because there is new lines and passwords and things are not authenticating properly. I'll show you one of the ways I debug that stuff.

But it's worth noting this is a encryption. This is just encoding it such that it can be put in the YAML file safely and submitted to the URL end point, to the Kubernetes end point safely as well. You got secrets in place. You need three pods which we saw. This comes down to we have one engine X server basically, does a little bit of telling it where to [inaudible] this is how you inject, these lines right here, how you inject a little variable. We have the app itself which is that main workhorse application. You can see here we are injecting two different secret values as environment variables. The app can pick up environment variables and start doing its authenticating tasks.

Down here we are telling it the location of the OCR application, which is something we don't need to do anymore. We are using Kubernetes auto discovery in order to find the

location of the OCR microservice. Down here we have the microservice itself which has no fancy configuration at all.

Do you use [inaudible] no, we are using this make file. I want to get it to the point where it's robots because I'm a big CI guy. But we are not quite there yet. Any time you put, you encrypt your production credentials Travis to run them [inaudible] I don't want to do them right now. I haven't gotten the time together to build Kubernetes auto dep loir or evolve wait someone else's.

(indecipherable).

There is a lot of stuff on get pods but these three pods, think of as Docker containers, exist down here because of the three deployment artifacts we just looked at.

When you run kubectl get services, you can see there is services attached -- oh. Hmm. Maybe get away with making this wider. Boom. Got 'em. There is services attached to these things. There is cluster IP, and cluster IPs which addresses virtual addresses you can reach out over Kubernetes to reach to, I'm going the rate this talk, okay. I don't need to rate this talk right now.

Those are created by these services. These are simple. You do kind service and then you don't do anything. You use the selector. This is creating a virtual IP attendant address, or something that is connected to the Rotisserie static

application, Rotisserie OCR app application. Once you created six things, now you have this cluster inside Kubernetes where applications can reach out to each other. That is where you get the point.

Then you need a front door application, that is where the early ingress comes in to play. The early ingress if you are familiar with routing and stuff or engine X configuration, this is what that has turned into.

You will see that we are taking the static and the all, there should be like a slash in here too, but we are routing some of the traffic to the engine X server to serve static files, and routing some of the traffic to the all end point to the app itself, so you can hit those API endpoints and make it work. Once you have floated all those in IBM Cloud container service your application starts to work. You can then come here and do something like get ingress and kub describe, we will use this because it's easier to -- oh, dang. Describe-ing Rotisserie ingress. You will see it has, well, because we have done fanciness and it no longer works, if you had done all the steps I said there, you would have a IP address, a public IP address you can end point DNS at. What we have and we had to get more serious about it and I am sensitive to everyone's time, in order to add let's encrypt and TLS certificates to the application, we had to increase the complexity of the app,

Kubernetes deployment. You can build things on IBM BlueMix and IBM container service with just a couple pods, couple services and ingress controller. But we wanted automatic certificate renewal through let's encrypt.

In order to do that, complexity went up and to the right which I guess might be to the left, up and to the right. We had to get serious about the actual network flow from the outside world through the ingress controller, through the load balancer into the services into the applications itself.

I'm going to show that, in 7 minutes you are not going to get it but I'm going to show it a little bit. The key thing to understand is that we have this thing called kube Lego is a project that let's you, project on GitHub, you put a small amount of YAML, this is all you have to submit, and it creates a deployment in your Kubernetes cluster that automatically sets up certificates for your ingress controllers, which is great. It's really great.

There is some complexity to it though. If you are familiar with let's encrypt it's a automatically renewing certificate authority. You have to get the same stuff all the time, the URL, the E-mail, that kind of stuff.

But the way it functions actually is by creating its own ingress controller, its own cluster IP and several other resources. There is a fair amount of look, leap of faith

actions it's doing. In order to do it, we had to, there is a element of when you create that ingress controller that there is magic on the back end that is wiring up your front end to work. In order to normalize our deployment to look more like what kube Lego expects we had to get more serious about it.

We created all this stuff in get serious.YAML. We created another ingress controller but explicitly labeled it with the engine X class. This is more realistic in terms of what routes look like.

We had to build stuff where we had to physically, we had to explicitly deploy an engine X container as a deployment, and then use the very, the accurate, authentic upstream inject engine X controller. Once we did that and wired it up with something called a load balancer type service, then we had a front door that came through that processed data out to our static service and app service, but also was detectable like kube Lego. Kube Lego could automatically modify. It's beyond the scope of what we have time for.

I want to leave the last couple minutes for questions. But Kubernetes is super dope. I've been using it for a couple months. Let me show you stuff we can do real fast. You can do.

(indecipherable) exec, this random pod LS, if you go to the pod and run commands in it, you can cap files in it, you

can do any debugging stuff you want to do inside the container.

The other thing you can do and, you can run this env command which will dump the environment variables of the application. That is a way to detect if secrets have been screwed up. There is extra white space blank finds between secrets in the output of this command that means you added a extra new line.

So yeah, two more minutes for questions, three minutes for questions. Does anybody have any? This has been really fun. Marc or Cullen, do you want to go off mute so you can answer questions too or anything you need to add?

>> MARC-ARTHUR PIERRE LOUIS: I will take one minute at the end to talk about the next couple tech talks we are going to have. If there is no questions, I'll do that.

>> SPENCER KRUM: There is a couple questions. Brian Gleason, the chat is [inaudible] which I appreciate. (overlapping speakers).

>> CULLEN TAYLOR: There is a question in the zoom chat about would you have any resources to get more info on developer advocates from James.

>> SPENCER KRUM: Go to developer.IBM.com/code and there is find developer advocates, you can hit me on Twitter if nothing else. We are happy to reach out with other people. You guys have been very good attendees. We appreciate that kind of feedback.

The CI pipeline to BlueMix continues delivery [inaudible] I want to say thank you very much. This has been a ton of fun.

>> MARC-ARTHUR PIERRE LOUIS: Thank you very much. We appreciate your time for this interesting presentation. I'm sure the guys are going to download the presentation and start playing with engine X and CT L and all that great stuff. Thank you for your time, Spencer and Cullen. Next week we are not going to have any tech talks because it's the holidays. We are going to pick up on November 28, and we are going to talk about BAE or bot asset exchange, which is a enterprise, exchange that we have. We are going to demonstrate that to you. We are living in the time of bots, good bots and bad bots but this one is going to be a good bot.

So November 28 and the speaker is going to be Anna mitta guha and on November 29 which is the next day we are going to talk about the IBM Cloud private and how that can help you to be efficient with your datacenter, and the speakers on that one are going to be John Zak owny and Tim Robinson. We are going to give you a break next week so that you can enjoy the holidays but we will pick up on November 28 and on November 29 for the BAE bots as exchange and November 29 the IBM Cloud private. Thank you for your time and looking forward to seeing you in two weeks. Bye-bye.

(end of session at 11:00 a.m. CST)

Services Provided By:

Caption First, Inc.
P.O. Box 3066
Monument, CO 80132
800-825-5234
www.captionfirst.com

***

***