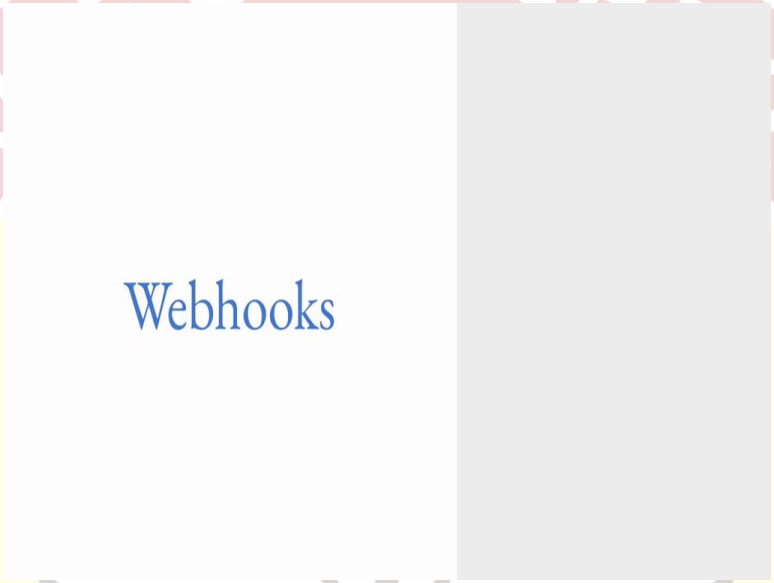
IIT Madras

ONLINE DEGREE

Modern Application Development - II
Professor. Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology, Madras
Webhooks

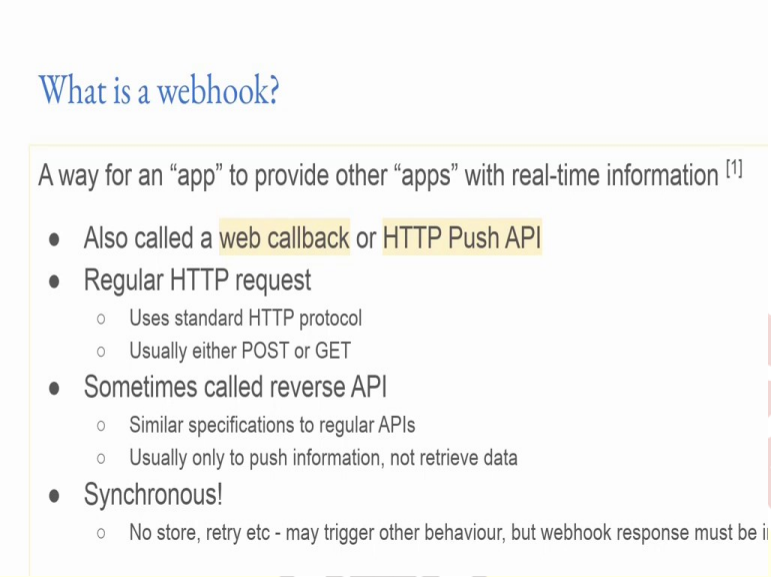
Hello, everyone, welcome to Modern Application Development part 2.

(Refer Slide Time: 00:14)



So, now we are going to look a little bit more in detail at what webhooks are and look at an example of how a webhook might be implemented, and how you can sort of pass information from one server to another as part of the webhook.

(Refer Slide Time: 00:29)



So, what is a webhook? It is a way for an app to provide other applications or apps with real time information. So, this definition is something that I have picked up from an article on the SendGrid. Website. SendGrid is again, a service that allows you to send email messages bulk email messages, or in general, email messages and they also provide their own webhooks.

They have an article about webhooks, which is there are a number of useful articles written by people who usually provide these services, that are worth reading to understand better, what was their motivation for providing webhooks. In this case, it is very clear, one app wants to tell another app, when I say app, I am just talking about a process or an application, that might be running on different servers that are completely unrelated to each other.

And this notion of real time information is something that is useful to keep in mind. Because ultimately, what we are interested in is how do I get real time information, meaning that as soon as an event has happened, I want server A to be able to send a message to server B. Now, the mechanism that is provided is a regular HTTP request.

Because of which this is sometimes also called a web callback or an HTTP Push API. Why API? Because at the end of the day, it is a programming interface. It is an application programming interface, it allows you to communicate between one application and another. HTTP Push because it is using the HTTP protocol and it is pushing messages from one server to another.

Now, you have to be a little careful with this terminology, because there is actually something else called HTTP Push protocol, which of course is related to this. But HTTP Push protocol has a very specific definition. It is an IETF standard, and therefore, this term HTTP Push API that I am using over here is a little bit more loosely defined.

But the reason why it is used is because it helps to convey a clear picture of what is being attempted out here. And similarly, you have the term the web callback that is a bit easier to understand. It is based on web technologies and you can literally think of it as a callback. I register a callback, meaning that app A tells app B, whenever you want to inform me about something, just call me back on this URL.

As I mentioned, it uses the regular HTTP protocol, the standard HTTP protocol, usually, either a POST request, or occasionally a get request POST is more common for the simple reason that within this callback, you might also want to pass a small amount of information, you usually do not want to have huge amounts of information, only a very small amount, because otherwise it becomes a full blown HTTP request.

And sort of defeats the purpose of this lightweight messaging idea that we have been talking about. GET requests are problematic, because everything has to be encoded into the URL, you cannot really get a whole lot of useful information across over there. So, POST is more common, it is structured, it is easier to send across, and it is a simple enough URL to use.

It also means that POST requests are generally not cached and which is a good thing for these web callbacks, you do not really want the results to be cached, because that is what sorts of defeats the purpose of a callback. This web hook is sometimes also called a reverse API, for reasons that I already mentioned.

An API is usually used in order to pull data from a server. For example, the list of courses, you are enrolled in, the list of students enrolled in a given course. All of that information is pulled out as part of APIs, of course, you do also have the rest of the crud, which is to create, update or delete, where you are sort of pushing information onto the server.

But for the most part, rest API's and similar kind of mechanisms are used to pull information off the server. Whereas, a webhook is very clear. It is not even trying to do a crud operation. It is just passing a message from one server to another. It is pushing a message from one server to

another. And hence, the terminology reverse API. Once again, like I said, this is not some kind of a standard terminology. These are just terms that are in common use because they help to understand how these systems are working.

One very important sort of distinguishing factor between a webhook and a messaging queue is that a webhook is actually synchronous. What does that mean? You are making a call, you are making the callback and when you make the callback you have to connect, and you have to be able to send the message.

In general, web hooks do not provide for retry or store and forward or store and retry. Those kinds of mechanisms that message queues provide you with, if the server to which you are trying to make the callback is offline, it just gives up, it does not really try. So, the whole idea is that it is the webhook should also therefore be something that just calls and returns immediately, it should not make the collar sort of wait for a long time.

The expectation is that when I post something to a webhook, whoever is receiving that information, just accepts the data and sort of gives me back a response code immediately. And then does whatever they want behind the scenes. They might get ready to do some new work or something of that sort.

But if you sort of hold it up and say that you are going to keep that connection occupied, that is effectively in some sense, abusing the system, you are doing something which is not expected by the person invoking the webhook. So for example, if you register a webhook with someone like GitHub, and your webhook is such that it just sort of hangs for a long time, GitHub is probably not going to like it and over time, they might find that this is blocking their resources, and they might just stop using the webhook, or they might kill off that functionality for you.

So, the idea is that you make a brief synchronous call. But the important point is webhooks are fundamentally synchronous, they do not sort of allow you to just leave a message and then get back later. That if you need that kind of functionality, what do you do? You use a messaging queue.

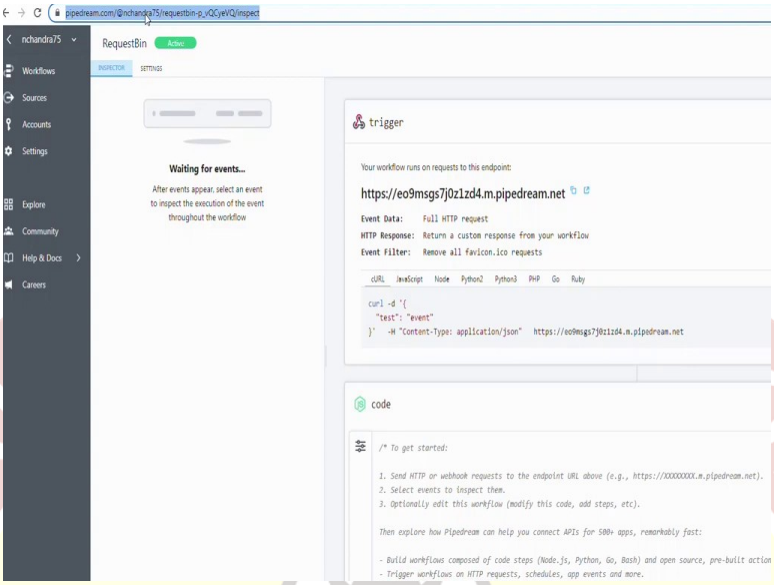
(Refer Slide Time: 07:06)

So, I am going to show you a bit of an example of what you can do with something like GitLab. So, GitHub and GitLab are both very well known and popular. Git hosting systems. So, Git, by itself is a version control system, GitHub and GitLab are front ends, are web based front ends that provide a lot of additional functionality on top of just storing the Git repository.

And this whole idea of a webhook is not part of Git, it is part of GitHub and GitLab, they provide that functionality for you. There is nothing fundamental about a webhook in connection with Git. So, what I am going to show here is how you can set up the receiver first, you need to have something that will receive your webhook. And once you are able to receive that webhook, you can decide what to do with it.

Now, one of the things that we have over here is actually there is this website called request bin and request bin essentially allows you to create temporary receivers. All of those receivers do is they will accept any request that is sent to them and lock them along with all the data that came along with it, which means that we can use it, for example, to debug a webhook.

(Refer Slide Time: 08:34)

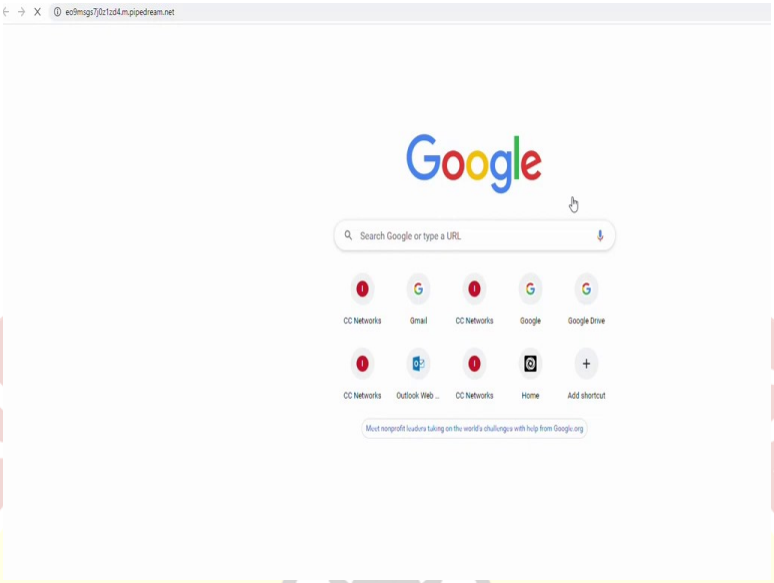


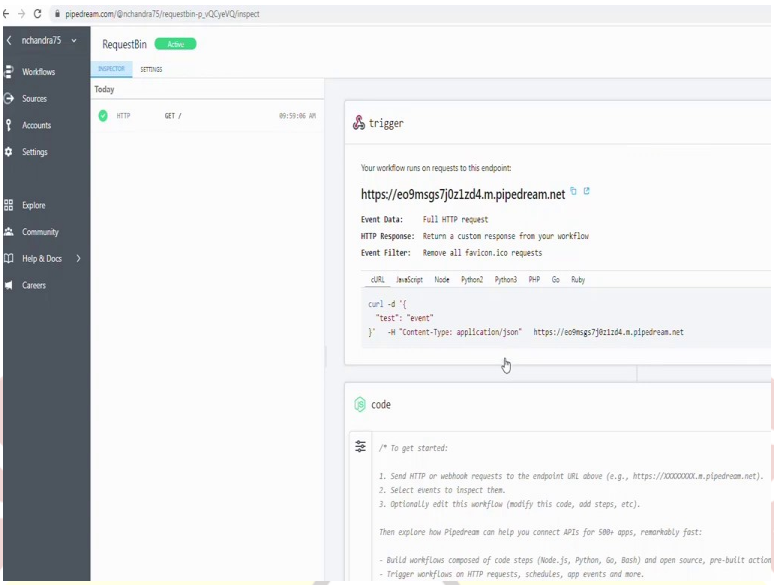
So, let us see how that works. So if you go to request bin.com, in turn takes you to a website called pipedream dot com, which is essentially sort of the parent company of whoever created request bin. So, you could create a public request bin if necessary, or you could just log in with your Google account, for example, and create something that will persist.

So, now what I have done in this case is I have created an account. And after that, I created a new request bin. And when I click on create new request bin, what ends up happening is I see this page out here, which basically gives me some information on the right hand side saying this is the trigger and it gives me more data. It says your workflow runs on requests to this endpoint.

So, look at the endpoint, it is something HTTPS is, some strange string of characters something dot pipe dream dot net. And what we expect is that if I make a request to this particular URL, I should find some information popping up out here on the left hand side. So, let me first just copy this URL.

(Refer Slide Time: 09:46)

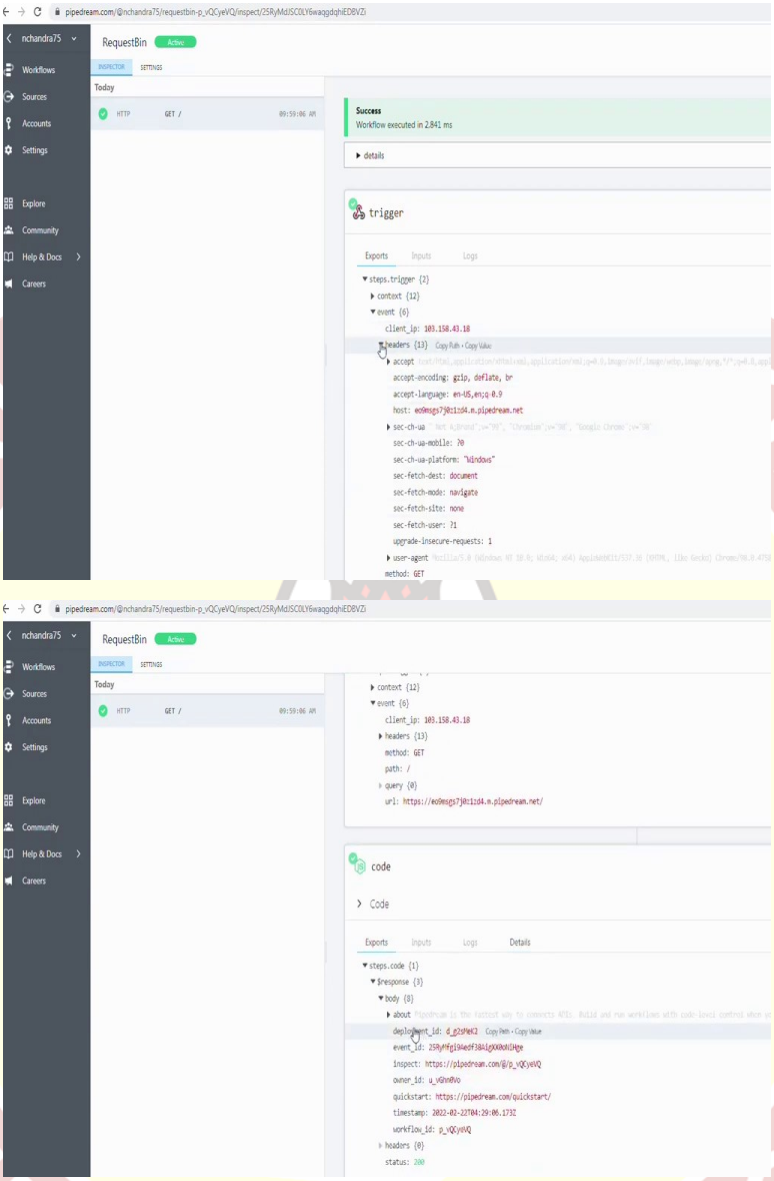




Open a new tab, just paste it in there and let us see whether anything comes up as a result. So, you can see that some kind of JSON data has been displayed out here. So what exactly happened, when I connected to request bin, I can also see now that on the left hand side, it says where it was saying waiting for events, it now says today there was an HTTP GET slash at this particular time.

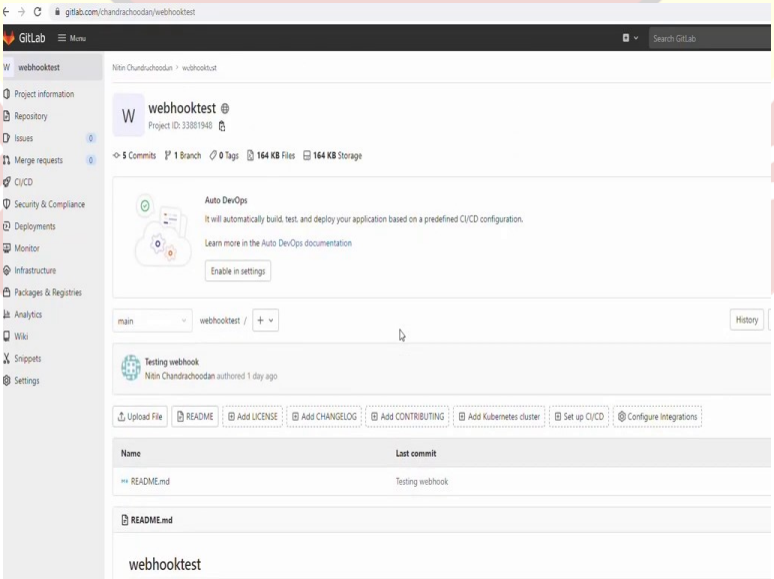
So, in other words, what has happened is this particular URL, this endpoint that has been provided to me by requestbin, when I directly make a request to it, it is a get request. It just returns some JSON data, which is some about information. It basically tells me something about pipedream and it also tells you about how it is the fastest way to connect API is at some their description of their service. How is this useful for us? What it means is that any request that is made to this particular URL.

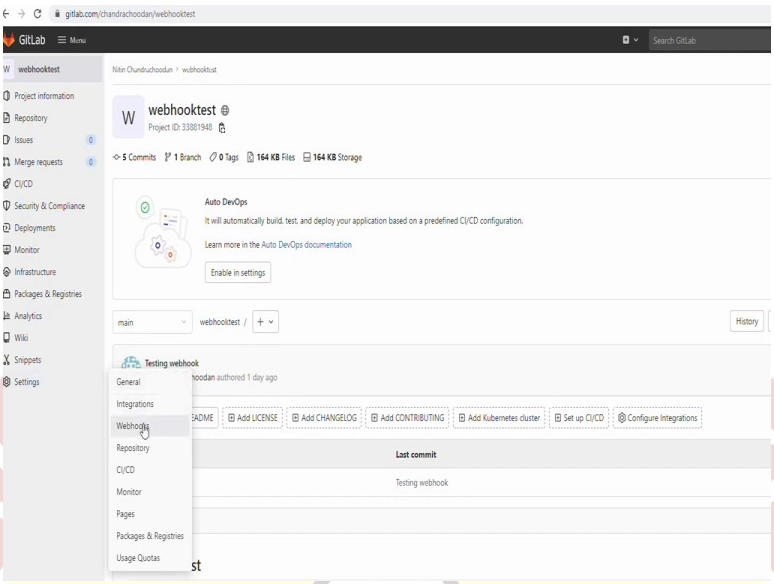
(Refer Slide Time: 10:51)



On the left hand side, when I click on it, it will show me a lot of details about it. It basically tells me the trigger, it says where the client was, what the headers were, that was sent by the client, It also says exactly what the query was. And it also tells me a little bit more detail about the exact code the response body, which was essentially the JSON data that was sent back to me. So, what it has done is it has taken the entire data that was sent as part of the request, sent back over there and it also shows me what information was sent back to the client, in this case was displayed out here. Now, how do I use this?

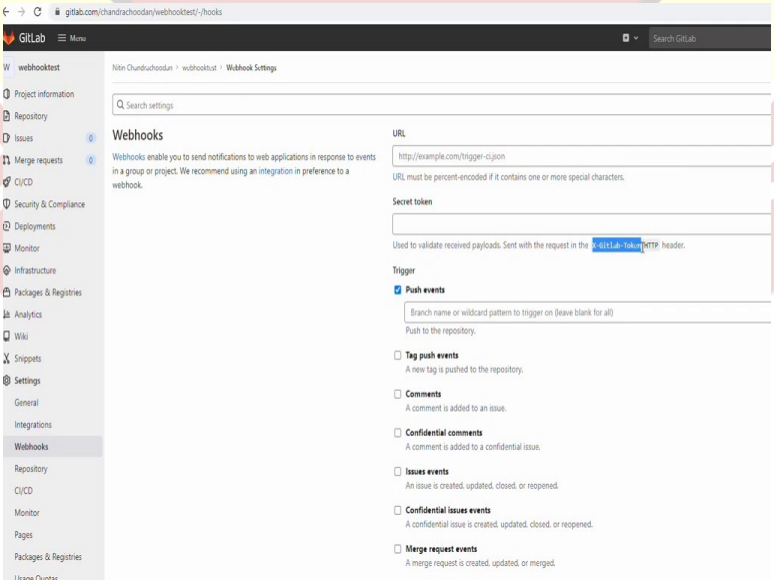
(Refer Slide Time: 11:36)





For that I have created another repository under GitLab. So, you can go ahead and get started with this. You do not need to use the same GitLab repository. In fact, you cannot because it is private to mine. But you do not need anything. In fact, if you look at it, this repository that I have created does not even have any files associated with it. And all that it has is it is just I created a repository initialised it with readme, and I am done. Now, what happens is, in this case, when I go to the repository, I can go down to settings on the left hand side, and it has an option out here to register a webhook. I click on web hooks.

(Refer Slide Time: 12:18)

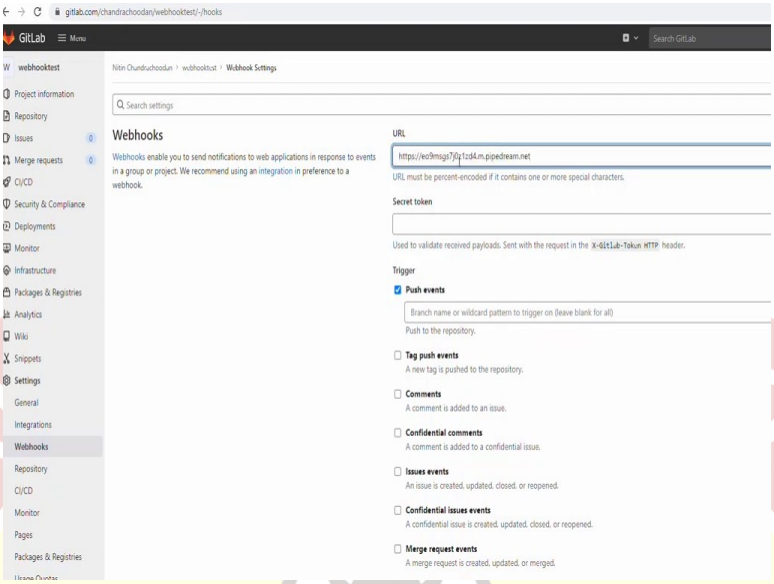


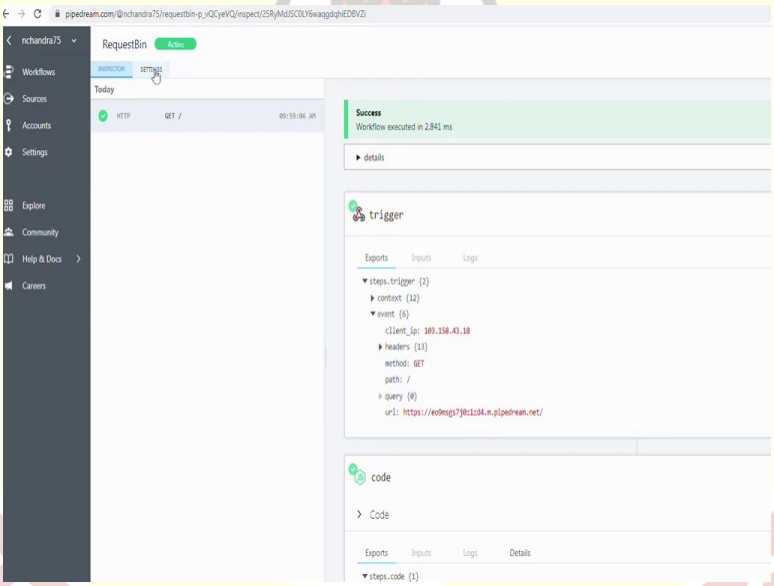
And what it tells me is, these are the webhooks that are currently sort of registered for this particular repository. It asks me for a URL, I can specify something called a secret token over here. Now, this is optional. What happens is, whatever I put into this information out here, will be sent as a header as an X, GitLab token HTTP header, along with every request that is being made to the URL being provided above. Why is that useful? It means that, in my receiver code, I can look for the GitLab token. And if the token is absent, I know that the request did not come from GitLab.

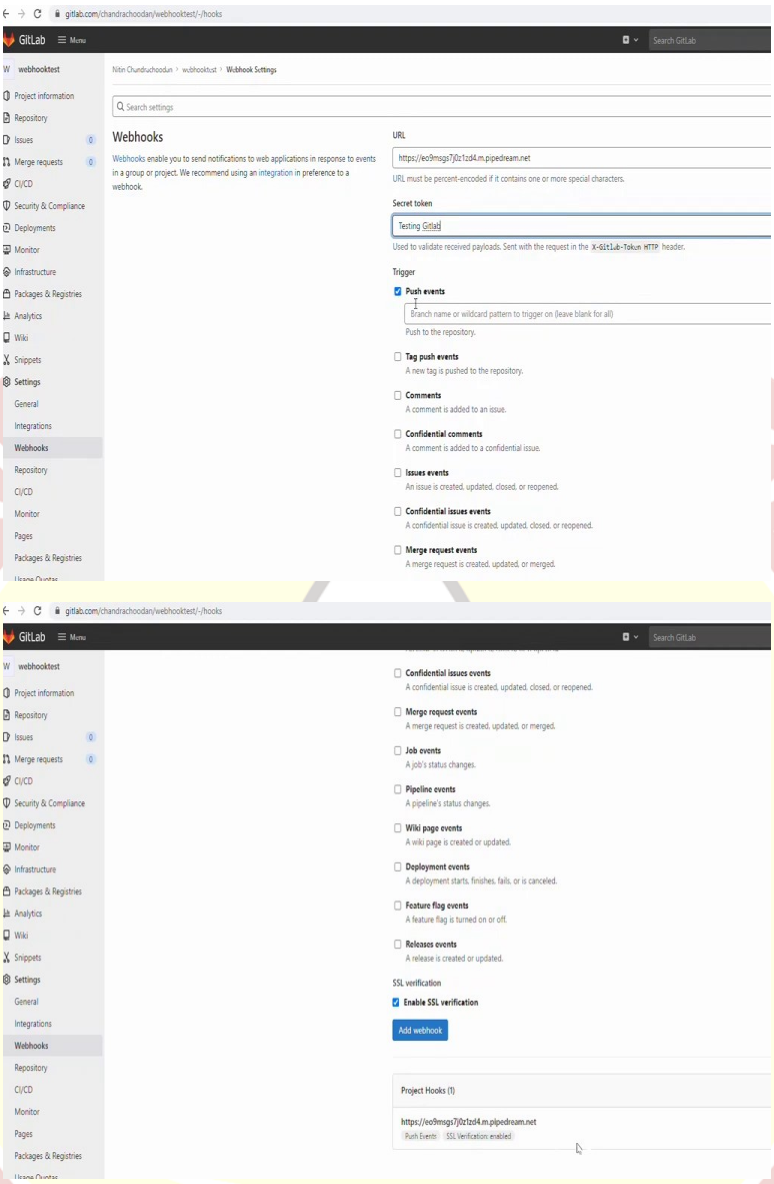
Somebody else got hold of my URL, and is just randomly calling me without really having useful information. Now, this is especially important. For example, let us say that the URL is something that ultimately would post messages to a chatroom. If someone got hold of that URL and they did not have this GitLab token, it did not and if you had not set a get lab token, it would mean that just by posting some random data to that URL, you might start spamming the chatroom, or even worse

You might even potentially be able to get information or even in general, disrupt whatever was the purpose of that URL. So, this is a very simple form of security, which at least allows you to make sure that as long as you have a secret token over here, which is not easily guessable, and is also included with every request that is sent along to the webhook. The first thing you do inside the web hook receiving code is to ensure that the token is present. Now, like I said, this is optional. It is not really that anything has been checked over here. So, how do we do this?

(Refer Slide Time: 14:13)







What I need to do in order to register a webhook is I can paste that URL in there. So, what I did is I just took the same URL that had been given to me by Pipedream and so this URL that was provided to me by Pipedream, I take the same thing and put it in as the URL for GitLab to called. I do not provide a secret token I am not particularly interested over here, or I could, you know, just try it out. I could put in some information out here testing GitLab. And what I say is that I am going to do this for push events. I could in fact, select what kind of events.

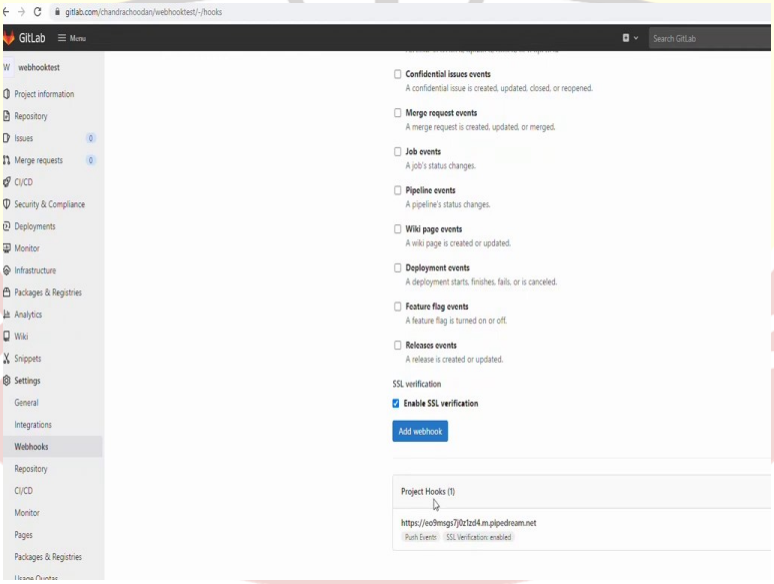
Now this makes it even more interesting. What it says is I could for example, select only tag push events. What is a tag? A tag in Git is something that allows you to take a particular commit

and associate some special functionality with it maybe a new feature release, or a bug fix or something like that.

So, a tag push is usually made whenever you have some improvement on the code or something that you want to flag on the code that says, this is something special. It is not just a temporary commit, I have actually got some functionality that I want to mark, clearly. Which means that you could choose that only for tag push events, this particular URL is called or maybe for any kind of comment added to an issue or confidential comments, events on issue, there are a whole bunch of different options that you have, or here that you could select.

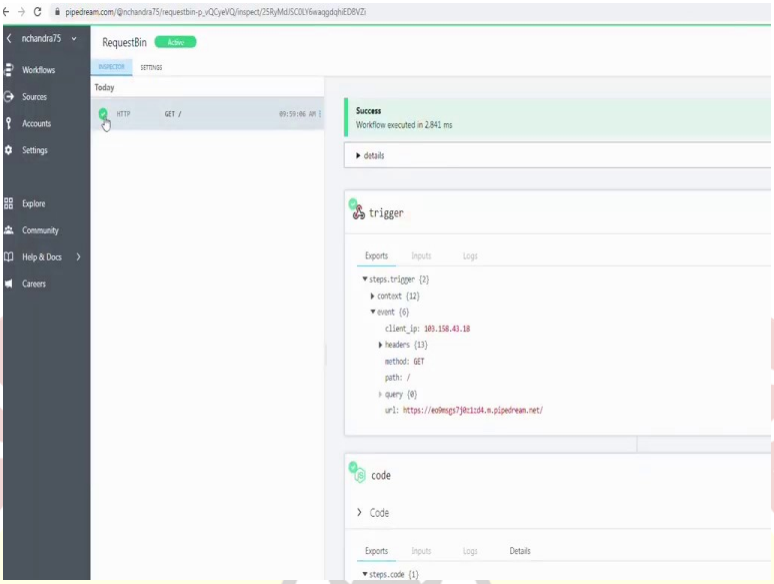
It also allows you to enable SSL verification, this is usually useful where you would want to skip this is let us say that you are running it on either an HTTP endpoint or you are running on HTTPS, but with your own privately signed certificate. Maybe you are just running it on a browser on a server that does not have a proper SSL certificate. Generally not a good idea, because that opens up other possibilities of data leaking out, which is why by default, they enable SSL verification. So, once you put all this in, you can just add webhook.

(Refer Slide Time: 16:20)



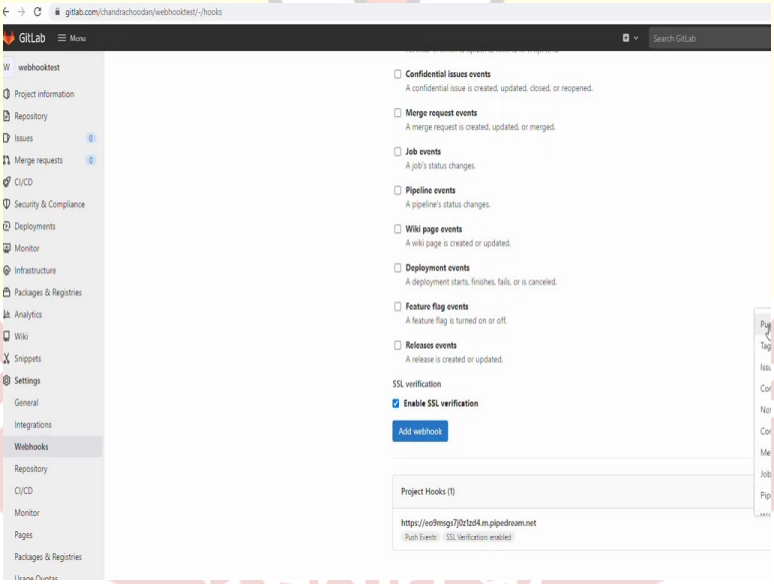
And you can see that it is now registered down here, there is one webhook associated with this.

(Refer Slide Time: 16:29)



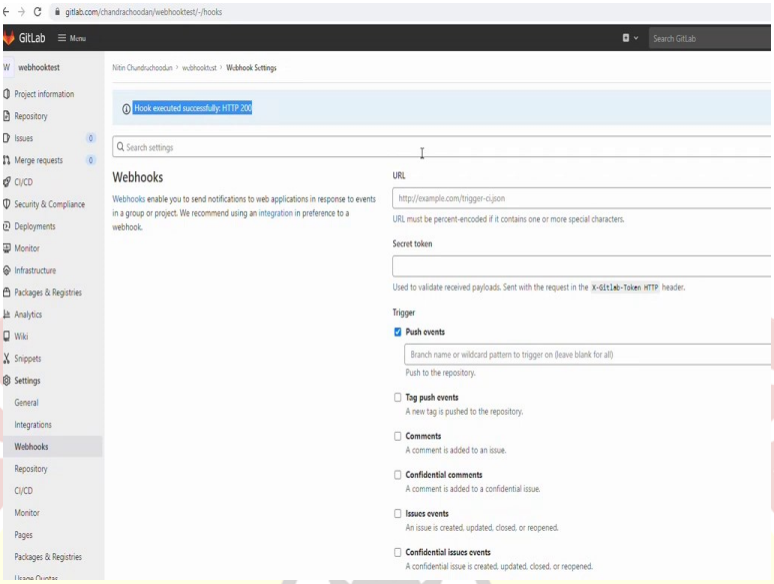
So, now let us go and look once again, a pipe dream, your request bin, and it just says on the left hand side that today, at some particular time, there was an HTTP get request.

(Refer Slide Time: 16:40)



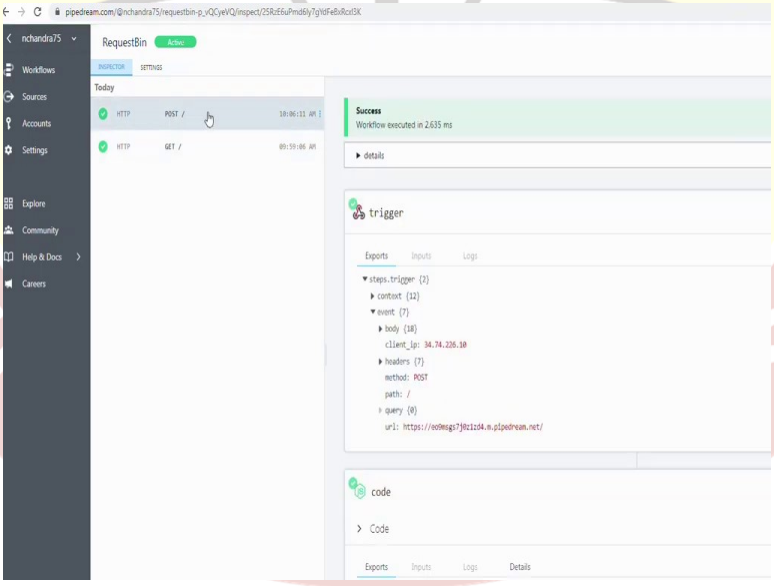
What I am going to do now is to go back to GitLab and I want to test if this particular web hook has been registered. So, what do I do, I can just click on test and generate a push event.

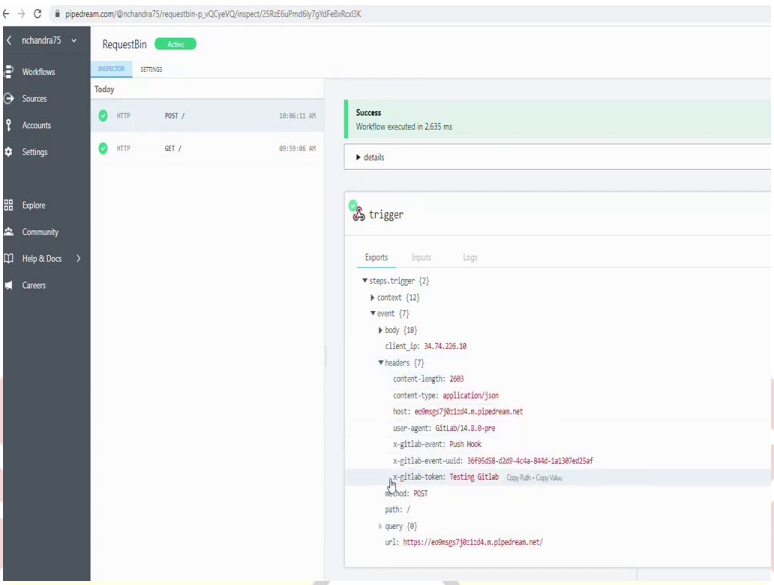
(Refer Slide Time: 16:57)



Now what happens when I generate this push event, you can see for a moment that GitLab sort of goes through and finally it puts a message up here saying hook executed successfully, and that it received an HTTP 200 response to the request. Where did that response come from?

(Refer Slide Time: 17:12)



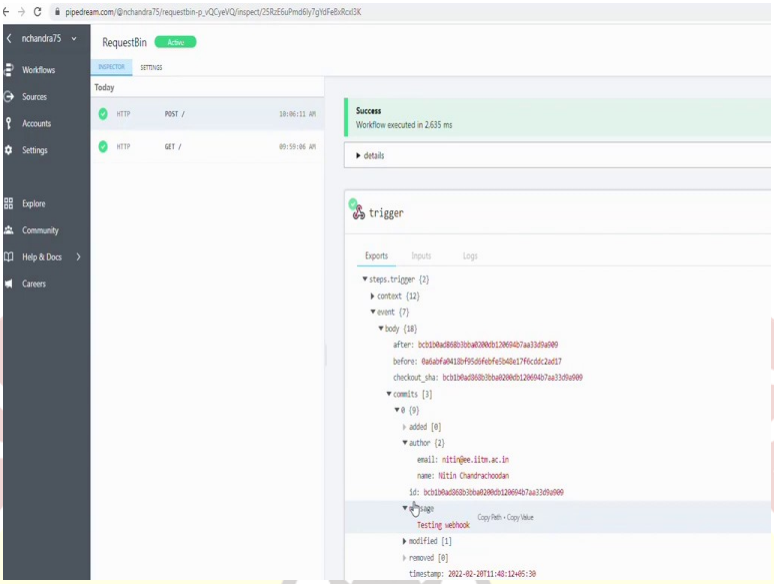


Let us, go back to a request bin and we can see that you now there is an HTTP POST sitting up here. And it was this HTTP POST that basically responded with a 200 code. Look on the right hand side, and it gives you the full details of what exactly was sent as part of the HTTP POST. You can see that this was the trigger and a couple of things to notice.

One is that among the headers, you will find that there is this thing X GitLab token, which is testing GitLab, which is exactly what we had asked it to put in. There are also a few others X GitLab event, So, this your hook, whatever is receiving it could look at this and say, okay this was a push hook, it was not for generate for a comment or an issue.

It was because actually a commit was pushed. In this case, I only just artificially did a test trigger, I did not actually push a commit, you could clone that repository, make some changes, and push the data back and see that the same thing gets called. There is a unique UID also, that is provided here.

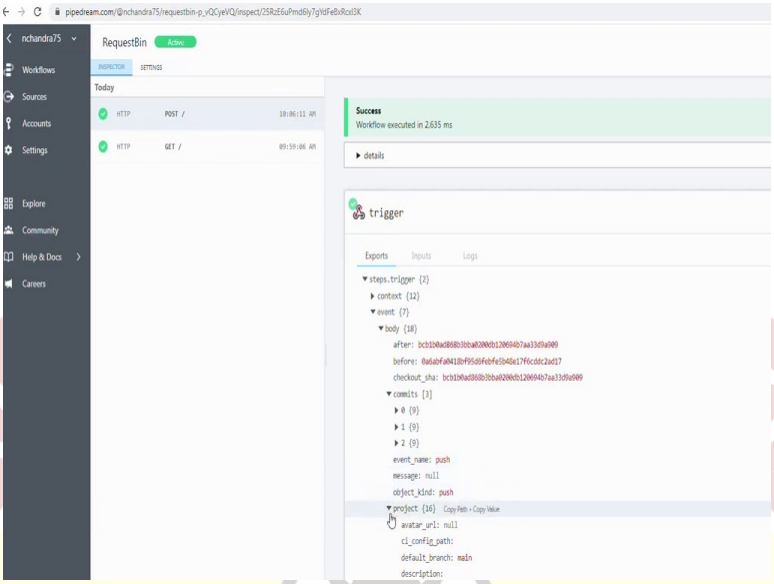
(Refer Slide Time: 18:29)



But now, let us look at the body over here of the trigger. This has a whole lot of useful information, it basically gives you a lot of detail about the present state of the commits. So, this is the final state of the commit the commit hash id from Git. It tells you the number of commits that were there. And in turn those commits commit number 0, it tells you that the author, information about the author, it tells you what was the message.

Now, where did all this come from these commits existed, because while preparing the webhook repository, I pushed two or three commits into this system. If I had not done that, this entire information saying commits, three would essentially probably not have been there, it would not have shown any commits.

(Refer Slide Time: 19:17)

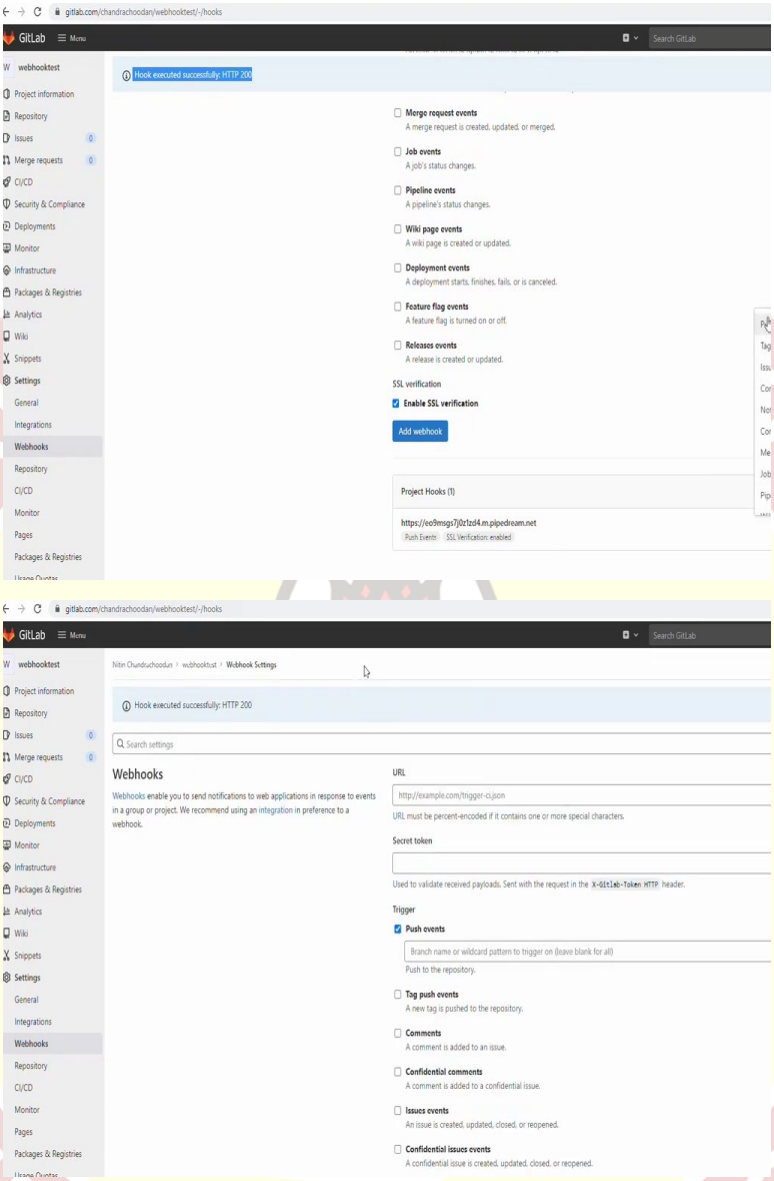


It also gives me details about the project. It tells me about the HTTP URL, the SSH URL, the homepage of the project, a lot of information regarding the project. So, if you look at it, there is actually quite a lot of information even though I have been repeatedly saying that this is supposed to be a lightweight message quite a lot of information has been sent by GitLab to your hook. But if you think about it, it is still just plain text and is not, it is not a huge amount of data. It is not megabytes of data that we are talking about over here. What is useful is all of this information can be parsed out nicely, and I can actually make use of it for something.

Now, one point to note over here is that request bin is actually showing it in very nice sort of structured manner with all these you have the body, and then the commits and project and so on, you might be wondering how exactly is this sent, very simple, it was actually sent as JSON data.

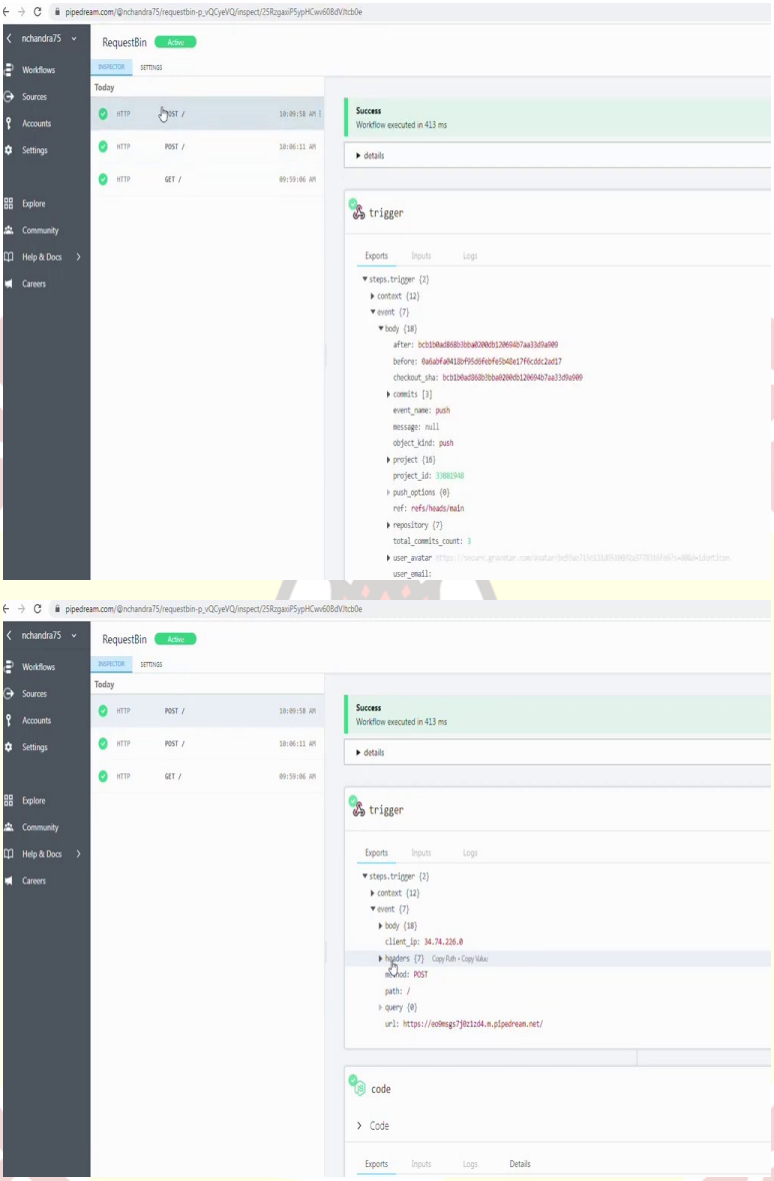
So which means that your actual POST request would be able to just receive a chunk of JSON data, parse it internally. So, Python, JavaScript, whichever back end language you are using, could almost trivially parse the JSON data and collect all of this information, just as it is shown over here and I can do this any number of times I want.

(Refer Slide Time: 20:37)



I can go back and once again, run a test, a push event. And you will see that, you know, once again, executed successfully.

(Refer Slide Time: 20:46)



I have one more POST over here and pretty much exactly the same data after nothing has changed on the repository. So, request bin is useful, just to let us see what is there in the request that came from GitLab. It is very useful. In other words for debugging, it helps you to see what information would be there in the message sent to the webhook. The next question that we might be interested in asking is, how do I make use of it? For that, we could create our own receiver.