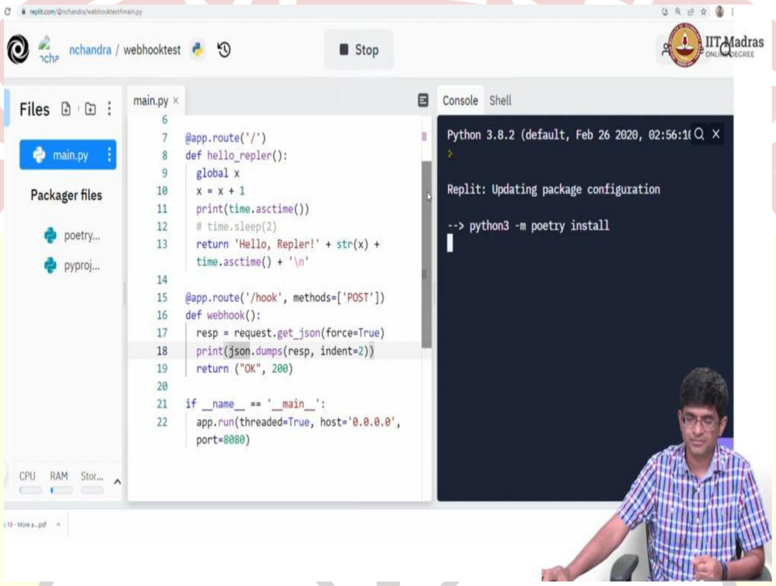
INDIAN INSTITUTE OF TECHNOLOGY MADRAS
सिद्धिर्भवति कर्मजा

IIT Madras
ONLINE DEGREE

Modern Application Development – II
Professor Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology, Madras
More about Webhooks

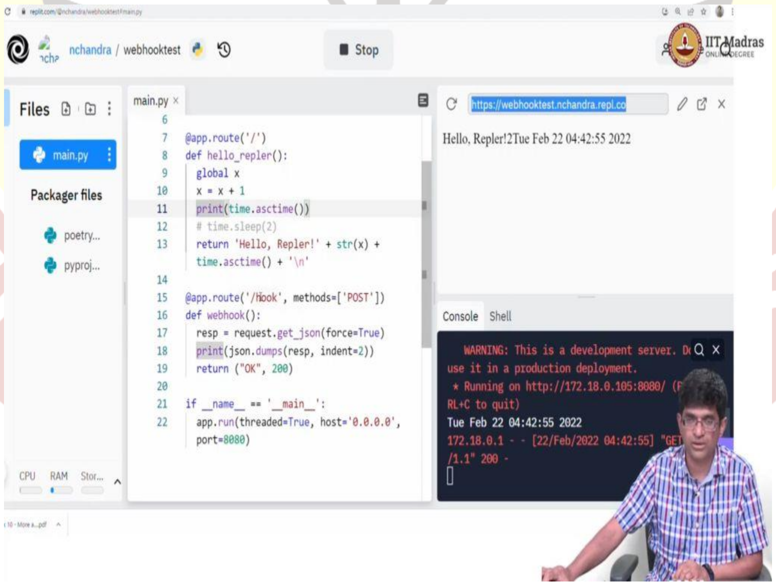
Hello everyone, welcome to Modern Application Development part 2.

(Refer Slide Time: 00:15)



```
@app.route('/')
def hello_repler():
    global x
    x = x + 1
    print(time.asctime())
    # time.sleep(2)
    return 'Hello, Repler!' + str(x) +
    time.asctime() + '\n'

@app.route('/hook', methods=['POST'])
def webhook():
    resp = request.get_json(force=True)
    print(json.dumps(resp, indent=2))
    return ("OK", 200)

if __name__ == '__main__':
    app.run(threaded=True, host='0.0.0.0',
    port=8080)
```

```
Python 3.8.2 (default, Feb 26 2020, 02:56:1(
>
Replit: Updating package configuration

--> python3 -m poetry install
```



```
https://webhooktest.nchandra.repl.co

Hello, Repler!2Tue Feb 22 04:42:55 2022
```

```
   WARNING: This is a development server. D(
use it in a production deployment.
 * Running on http://172.18.0.105:8080/ (P
RL+C to quit)
Tue Feb 22 04:42:55 2022
172.18.0.1 - - [22/Feb/2022 04:42:55] "GET
/1.1" 200 -
```

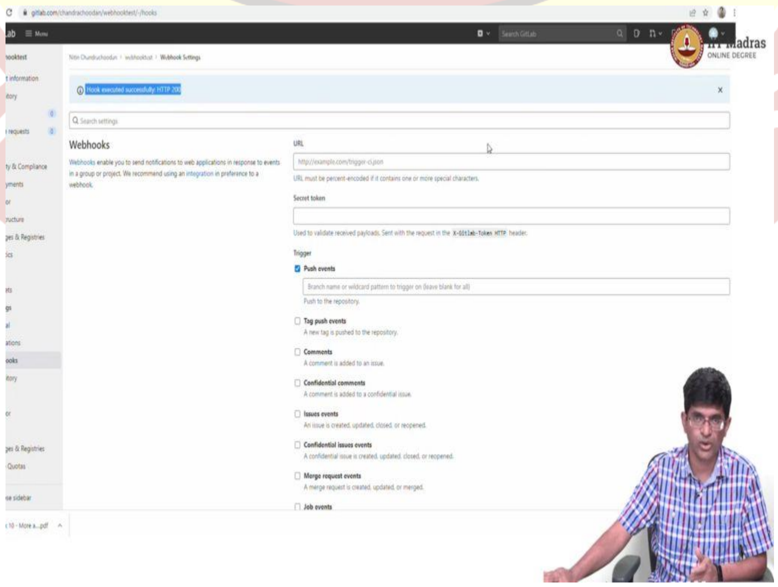
So, I have created another Replit in ripple in Replit; that essentially, is a simple flask application. What is a flask application do? I still have some of this; you know the standard template data that I had in order to demonstrate some of the information about, how flask responds to delayed actions and so on. But, what is important over here is this new route that I have added. I have basically created a new route slash hook, and I have specified that it has to be called with a POST method.

What does it do when I get the POST method? It basically gets the request json out; this get json with force equal to true will force the flask to pull the data out in json format. And I can then print it out, sort of pretty print it over here as a string. Now, why am I printing it out and where will it go when I print it out? What will happen when I print it out? Is that it will actually get printed on the debug logs, or the console of where I am running the server. It does not in particular, go back to the client. What goes back to the client? Just a simple message, OK; and the HTTP code 200.

So, why is that? Because in general, remember that a webhook is not expecting data in response; it is only expecting a status code. So, I need to make sure that whatever I sent back as the response to the webhook is kept minimal. So now, the interesting thing is I can run this, it takes a moment because it needs to install flask. And you can see that once flask is up and running, it has essentially created this URL out here on the right. It has automatically made a request to the get slash which of course in this case; all that it did was print some information out there.

But, I am not really interested in the get slash; what I am interested in is this URL domain. And what I need to do after that is to add this slash hook.

(Refer Slide Time: 02:28)


gitlab.com/chandrachoodan/webhooktest/-/hooks

Nitin Chandrachoodan › webhooktest › Webhook Settings

Hook executed successfully: HTTP 200

Webhooks

Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an integration in preference to a webhook.

URL

http://example.com/trigger-ci.json

URL must be percent-encoded if it contains one or more special characters.

Secret token

Used to validate received payloads. Sent with the request in the `X-Gitlab-Token HTTP` header.

Trigger

- [✓] Push events

  Branch name or wildcard pattern to trigger on (leave blank for all)

  Push to the repository.
- [ ] Tag push events

  A new tag is pushed to the repository.
- [ ] Comments

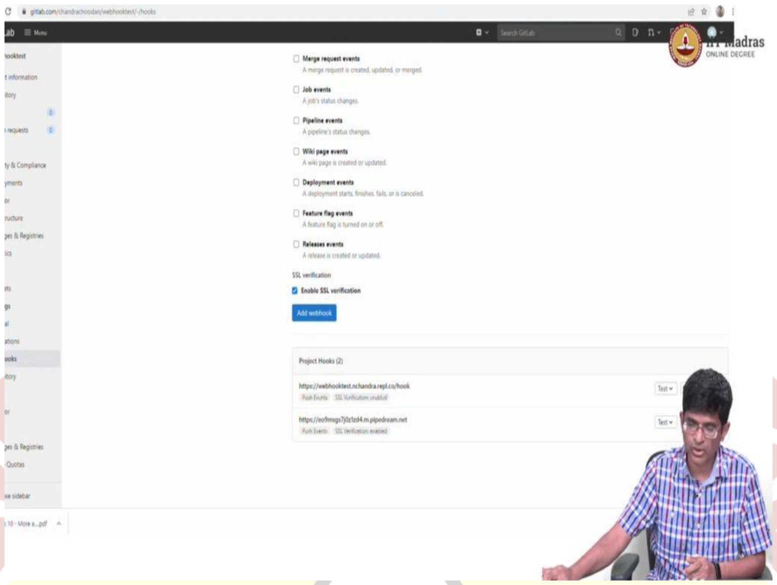
  A comment is added to an issue.
- [ ] Confidential comments

  A comment is added to a confidential issue.
- [ ] Issues events

  An issue is created, updated, closed, or reopened.
- [ ] Confidential issues events

  A confidential issue is created, updated, closed, or reopened.
- [ ] Merge request events

  A merge request is created, updated, or merged.
- [ ] Job events



- Merge request events
  A merge request is created, updated, or merged.
- Job events
  A job's status changes.
- Pipeline events
  A pipeline's status changes.
- Wiki page events
  A wiki page is created or updated.
- Deployment events
  A deployment starts, finishes, fails, or is canceled.
- Feature flag events
  A feature flag is turned on or off.
- Releases events
  A release is created or updated.

SSL verification

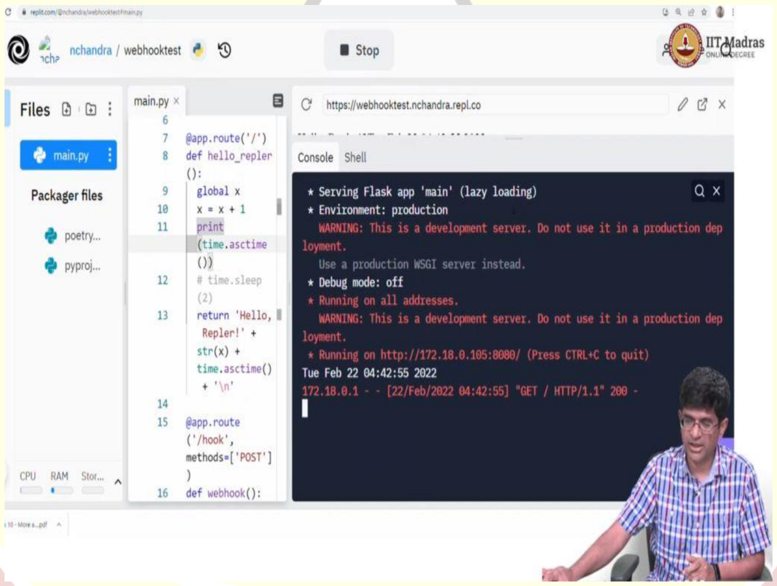
- [x] Enable SSL verification

Add webhook

Project Hooks (2)

https://webhooktest.nchandra.repl.co/hook
Push Events · SSL Verification: enabled

https://eo9msgs7j0z1zd4.m.pipedream.net
Push Events · SSL Verification: enabled



https://webhooktest.nchandra.repl.co

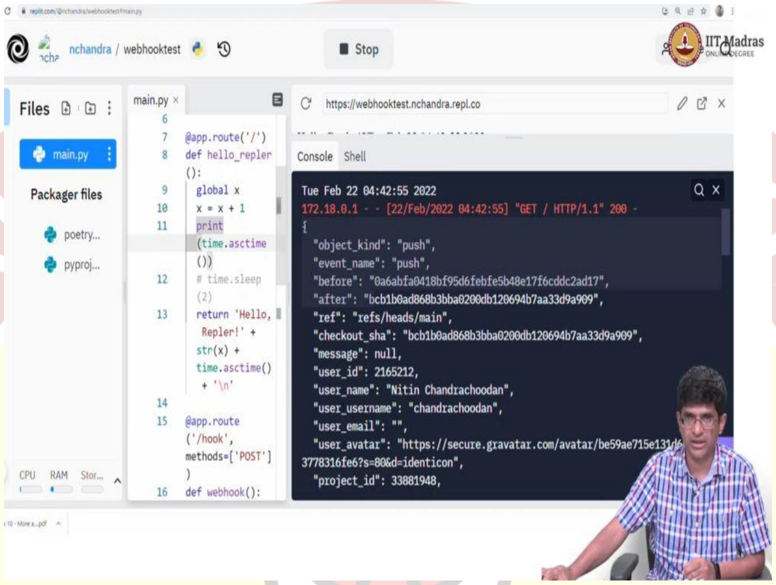
```python
@app.route('/')
def hello_repler():
    global x
    x = x + 1
    print(time.asctime())
    # time.sleep(2)
    return 'Hello, Repler!' + str(x) + time.asctime() + '\n'

@app.route('/hook', methods=['POST'])
def webhook():
```

```
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production dep
loyment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production dep
loyment.
 * Running on http://172.18.0.105:8080/ (Press CTRL+C to quit)
Tue Feb 22 04:42:55 2022
172.18.0.1 - - [22/Feb/2022 04:42:55] "GET / HTTP/1.1" 200 -
```

So, let me go back to my gitlab and say that I am going to register this slash hook as my URL. Once again, I will give another secret token out here; and once again call it for push events. What do I do? I add the webhook. Now, if I look at it, I find that there are two project webhooks, which I can test independently. The interesting thing is when an actual push event happens on the repository; both the hooks will be called. But right now, I can test them independently. What I will do here is I will just test webhook test dot nchandra dot repl dot co slash hook.

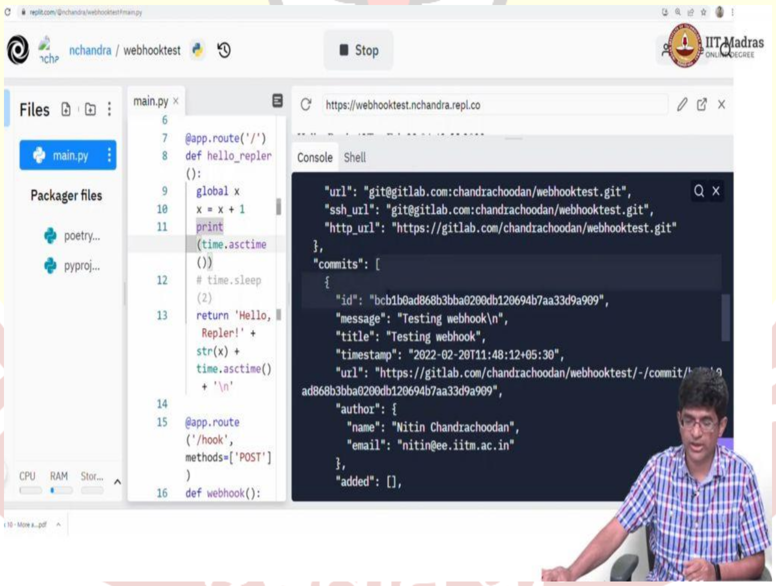
First, let me go back and take a look at what the logs over here look like. You can see the bottom right hand side; I will just make that a little bit bigger. And you can see that the messages are being printed out there. Let us go and test that particular hook; I test push events. And I find that the hook executed successfully and got back in HTTP 200. You will notice that the OK that was being sent back was actually ignored. It did not show that in any case; it only cared about the HTTP 200 that was sent back.

(Refer Slide Time: 03:50)



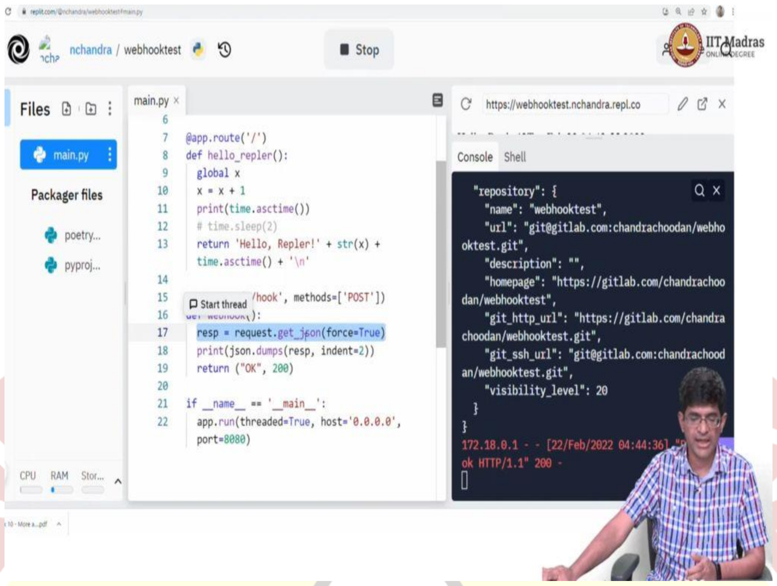
```python
@app.route('/')
def hello_repler():
    global x
    x = x + 1
    print(time.asctime())
    # time.sleep(2)
    return 'Hello, Repler!' + str(x) + time.asctime() + '\n'

@app.route('/hook', methods=['POST'])
def webhook():
```

```
Tue Feb 22 04:42:55 2022
172.18.0.1 - - [22/Feb/2022 04:42:55] "GET / HTTP/1.1" 200 -
{
  "object_kind": "push",
  "event_name": "push",
  "before": "0a6abfa0418bf95d6febfe5b48e17f6cddc2ad17",
  "after": "bcb1b0ad868b3bba0200db120694b7aa33d9a909",
  "ref": "refs/heads/main",
  "checkout_sha": "bcb1b0ad868b3bba0200db120694b7aa33d9a909",
  "message": null,
  "user_id": 2165212,
  "user_name": "Nitin Chandrachoodan",
  "user_username": "chandrachoodan",
  "user_email": "",
  "user_avatar": "https://secure.gravatar.com/avatar/be59ae715e131d6
3778316fe6?s=80&d=identicon",
  "project_id": 33881948,
```



```python
@app.route('/')
def hello_repler():
    global x
    x = x + 1
    print(time.asctime())
    # time.sleep(2)
    return 'Hello, Repler!' + str(x) + time.asctime() + '\n'

@app.route('/hook', methods=['POST'])
def webhook():
```

```
    "url": "git@gitlab.com:chandrachoodan/webhooktest.git",
    "ssh_url": "git@gitlab.com:chandrachoodan/webhooktest.git",
    "http_url": "https://gitlab.com/chandrachoodan/webhooktest.git"
  },
  "commits": [
    {
      "id": "bcb1b0ad868b3bba0200db120694b7aa33d9a909",
      "message": "Testing webhook\n",
      "title": "Testing webhook",
      "timestamp": "2022-02-20T11:48:12+05:30",
      "url": "https://gitlab.com/chandrachoodan/webhooktest/-/commit/b[illegible]0
ad868b3bba0200db120694b7aa33d9a909",
      "author": {
        "name": "Nitin Chandrachoodan",
        "email": "nitin@ee.iitm.ac.in"
      },
      "added": [],
```



```python
@app.route('/')
def hello_repler():
  global x
  x = x + 1
  print(time.asctime())
  # time.sleep(2)
  return 'Hello, Repler!' + str(x) +
  time.asctime() + '\n'

                /hook', methods=['POST'])
            ():
  resp = request.get_json(force=True)
  print(json.dumps(resp, indent=2))
  return ("OK", 200)

if __name__ == '__main__':
  app.run(threaded=True, host='0.0.0.0',
  port=8080)
```

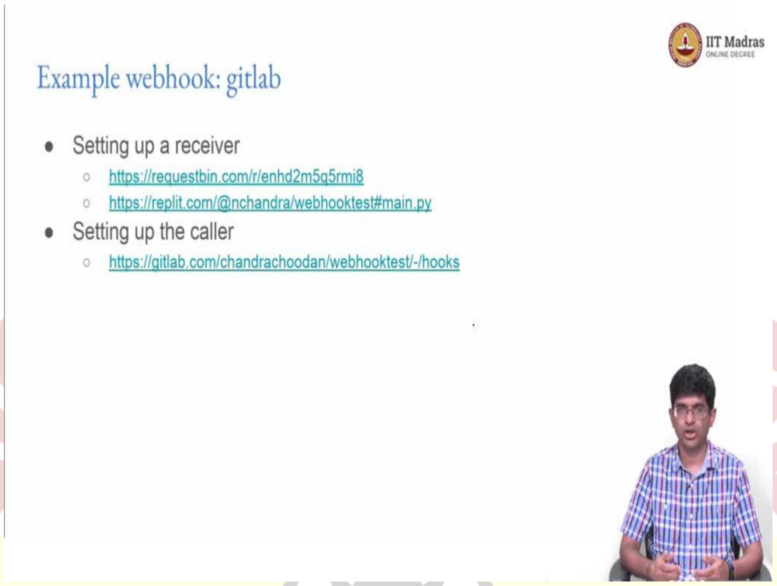
```
  "repository": {
    "name": "webhooktest",
    "url": "git@gitlab.com:chandrachoodan/webho
oktest.git",
    "description": "",
    "homepage": "https://gitlab.com/chandrachoo
dan/webhooktest",
    "git_http_url": "https://gitlab.com/chandra
choodan/webhooktest.git",
    "git_ssh_url": "git@gitlab.com:chandrachood
an/webhooktest.git",
    "visibility_level": 20
  }
}
172.18.0.1 - - [22/Feb/2022 04:44:36] "
ok HTTP/1.1" 200 -
```

Go back to my ripple and I find that it in fact has printed out a whole lot of information out here. So, the fact that it got that slash hook out there. From there, everything has been printed out in the form of json data. Pretty printed according to the json dot dump s command that I have in my ripple code; but all the same information that you saw on request bin. So, it has the project ID the name, the description; it has the commits, which are once again an array of commits over here. So, you can see commit 0, commit 1, commit 2; it has the author's name for each of the commits; exactly the same information that you saw, in the case of the request bin.

The good thing is since this is flask code; unlike instead of just printing out the data out here, I could have processed it. I could have written my code to do anything I wanted. Pulled out only useful information that I need, constructed a small message that needs to be sent to chat and posted it onto a Google Chat, or onto a Slack channel. So, all of that information now becomes trivially possible, because I am running full-blown Python code out here at the backend. All that I have done is I have received my some request, which came from the webhook.

I can extract only the part that I care about from the webhook and send out, and do anything that I want with it. I could log it, I could post a message to a chat. I could increment certain counters that tell me who has committed how many messages, or how many lines of code into different parts of the system. And keep track of maybe, who gets star for being the best performer in terms of committing lines of code. So, all of that now becomes possible because I am running arbitrary Python code out here.

(Refer Slide Time: 05:45)

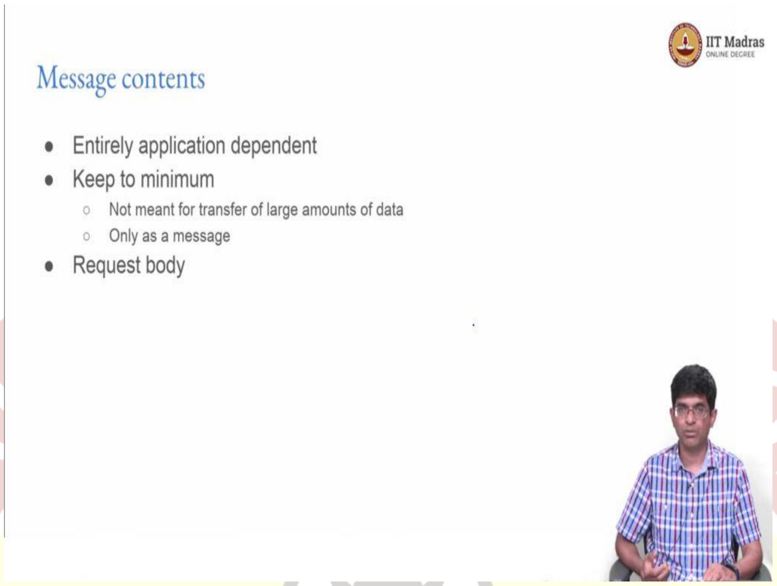

Example webhook: gitlab

- Setting up a receiver
  - https://requestbin.com/r/enhd2m5q5rmi8
  - https://replit.com/@nchandra/webhooktest#main.py
- Setting up the caller
  - https://gitlab.com/chandrachoodan/webhooktest/-/hooks

So, the purpose of the previous examples was to show how you can set up a receiver for a web hook, this is arbitrary. In general, you could set up any kind of receiver, just by saying that I am willing to accept HTTP requests. And how do you set up the caller? That is more specific to gitlab; because I needed to specify to gitlab saying use this URL, use this token if necessary. I mean, that is just a sort of simple, trivial way of establishing security between the caller and the callee. And of course, gitlab has certain data that it sends along with each webhook request, which you can then learn to make use of.

So, you can see how you could have set this up in any way that you wanted? Anybody could set up a webhook receiver, and say, look call me when you have any useful method to useful information to send. And anybody could say I can call a webhook if needed, any service that you provide. You could also provide webhooks, where you tell the person look if you want useful information real time updates on something, you can just register your own webhooks with me. And I will send out a call to them when the a particular type of event occurs.

(Refer Slide Time: 07:07)

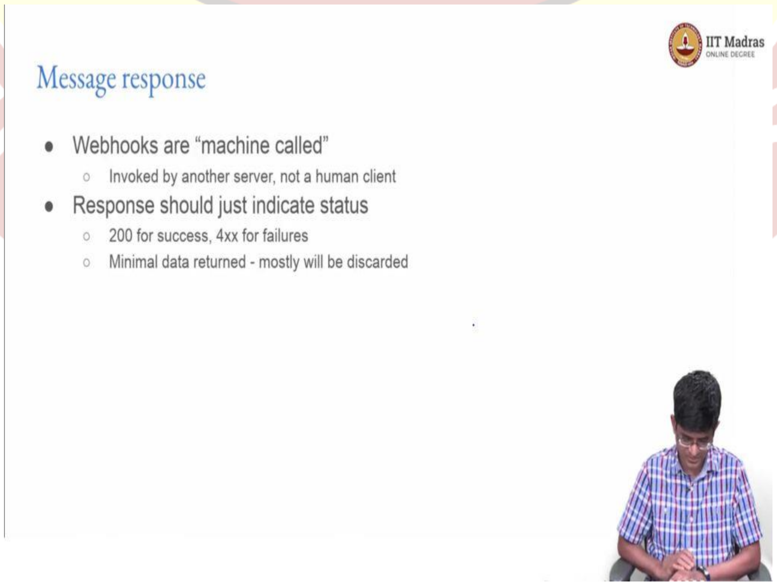

IIT Madras ONLINE DEGREE

Message contents

- Entirely application dependent
- Keep to minimum
  - Not meant for transfer of large amounts of data
  - Only as a message
- Request body

So, generally speaking, the message content that is sent as part of a webhook is entirely application dependent. There are no standards, there is no (con), there is there are few conventions that sort of tell you to use json, and a few things of that sort. But apart from that, nothing really telling you what should be or what should not be in the message content. But, the idea is you keep it to a minimum; it is not meant to transfer a huge amount of data, it is meant just for conveying a message. And the message itself is usually sent as part of the request body, which means that it can easily be transferred and controlled as json.

(Refer Slide Time: 07:44)


IIT Madras ONLINE DEGREE

Message response

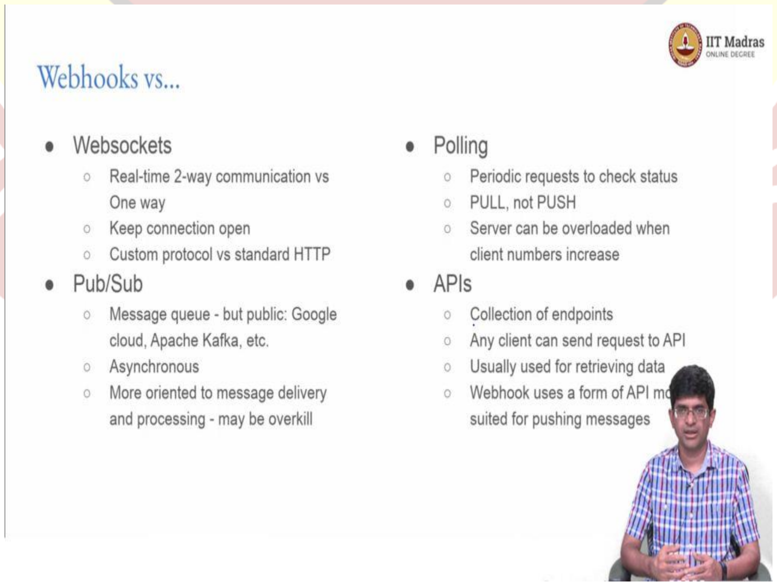
- Webhooks are "machine called"
  - Invoked by another server, not a human client
- Response should just indicate status
  - 200 for success, 4xx for failures
  - Minimal data returned - mostly will be discarded

And that is as far as the message request is concerned. What about the message response? Remember that a webhook is being called by another machine; it not by a human, not by a web front end; which means that is being invoked by another server, which is usually not sort of waiting for a detailed response. At best, if you are sending something back, you should just be sending back something which says a minimum amount of json data, which might be used by the webhook caller. Gitlab, for example, is pretty much going to ignore any data that you send back.

But, you might have, you might create your own webhook caller, which says that when I am done with it, if you can send me back some information in JSON; that will be good. I will use that for my own logs. But, that is to be avoided as far as possible, the whole idea for webhook, is it is a one way message. And the response should pretty much just indicate the status; as long as you send back a 200 response code, you are done. It means that the webhook succeeded. If you send back something in the 400 range, it usually means that there was a failure, something went wrong.

And that is enough for the caller to know that whether a webhook succeeded or not. What does the caller do in general to keep track of webhooks, if it finds that they are repeatedly failing? Then there is a chance that it might sort of stop calling the webhook after a while; but that is all that it really cares about; it only cares about the final status code.

(Refer Slide Time: 09:09)



Webhooks vs...

- Websockets
  - Real-time 2-way communication vs One way
  - Keep connection open
  - Custom protocol vs standard HTTP
- Pub/Sub
  - Message queue - but public: Google cloud, Apache Kafka, etc.
  - Asynchronous
  - More oriented to message delivery and processing - may be overkill
- Polling
  - Periodic requests to check status
  - PULL, not PUSH
  - Server can be overloaded when client numbers increase
- APIs
  - Collection of endpoints
  - Any client can send request to API
  - Usually used for retrieving data
  - Webhook uses a form of API mo
  - suited for pushing messages

So, all this while we have been discussing webhooks, but these are not the only means of doing this kind of lightweight communication, or communication in general between different systems. There are a number of other variants that you will see in different places. Rather than going into details on all of those, I am just going to sort of mention them. And sort of leave them as material for you to follow up and learn about on your own.

One example, which once again is very good for communication, direct real time communication between two services; or could be even between a client and a server is what is called websockets. Now, websockets are fundamentally different from HTTP. They do not have the so called stateless nature that where the server says, I really do not know what the client is doing, you just send me a request and I will send you back a response.

A websocket is basically creating a real time two way communication channel. It is a two way channel, which means that I can send messages to the server; I can also receive messages back from the server. At any point, the server can send information back to me. The connection therefore, between the two between the client and the server, or between the two servers has to be kept open; so that either side can initiate a data transfer at any point in time, which is different from HTTP.

HTTP fundamentally assumes that once you have made the request and you have got a response, you can at any point close the connection. You may for certain kinds of optimizations keep the connection open for a while longer; but generally speaking, the expectation is that you should be allowed to close the connection at any point you want. Websockets use a custom protocol, they are not using HTTP; they build on top of TCP, the Transmission Control Protocol. So, it is still using the basic internet protocol standards.

But apart from that, the headers the entire mechanism for setting up a connection, the fact that it is a two way real time communication. All of that means that there are significant differences in how the thing operates. So, you could use websockets for a number of especially real time kind of applications, things like chat applications.

There are certain actual terminal servers that you could open that allow you to directly type in, as your got a command line interface. So, the Google cloud shell, for example, where you can type in commands in a web browser, and it directly executes them on your virtual machine somewhere in the Google cloud. That is an example of something which is actually using a websocket to establish a real time communication between your browser and the server at the other end.

Another alternative to websockets is Pub/Sub, which I have sort of been talking about. Ultimately, Pub/Sub is a form of message queue. It is not exactly the same because this is more the publish subscribe, rather than just being a dedicated queue. The point that I am trying to make over here is that the Pub/Sub instances that we are looking at could potentially be public instances; and there are a number of public Pub/Sub services. There is the Google cloud messaging, there is Apache has a project called Kafka. There are a number of things that you could either set up on your own and expose to the cloud, or to the internet.

Or, you could even just purchase; you pay for a certain amount; and they allow you to send a certain number of publish and subscribe messages to a public service. Now this, of course, has all the traditional properties of the message queues. It is asynchronous; it is more oriented to message delivery and processing. It in general depends on the use case. If you really want to have that kind of publish subscribe kind of behavior, then of course you need to use it. But for the instances, the examples that we have considered so far, it is probably overkill. It is too complicated to register and set up a complete Pub/Sub mechanism just to sort of send back a message from gitlab to someone saying that, somebody committed few more lines of code.

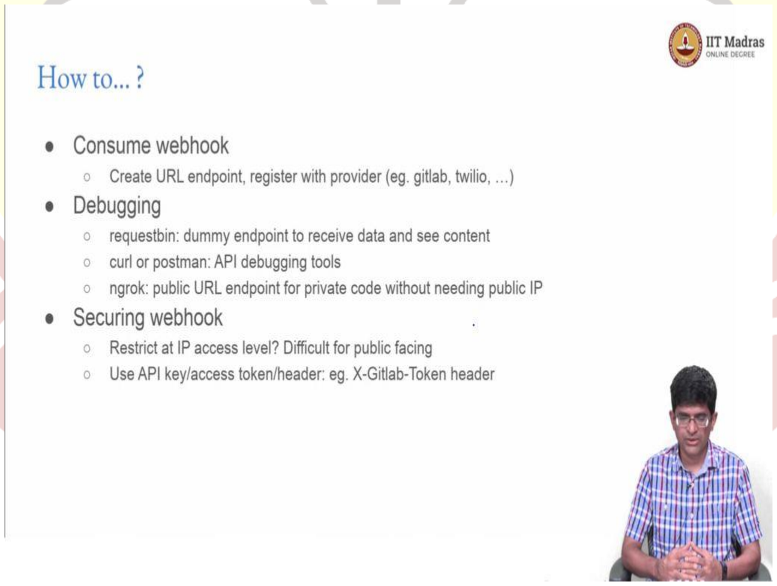
Now, webhooks are a way by which you want to communicate from server A to server B; and server B then calls back to server A saying that, something happened, some job got completed or somebody posted some message or whatever it is. Instead, server A could also have repeatedly kept asking server B. Do you have any information for me? Do you have any updates? So that is another form of communication between A and B, which is usually called polling. A keeps on polling B, it connects to B in order to check whether there is an update.

You can do this at a regular interval. It is what is called a PULL mechanism. A is repeatedly pulling data from B in order to find out if there is an update. You might know that for example, when you are checking mail on a phone, there is this thing where you keep sort of swiping down and it refreshes; or you are on a browser and Gmail, and you click on the Refresh button. You want to check whether there is new mail. Effectively, what is happening there is that you are polling the system and saying, is there anything new for me?

On the other hand, even if you do not do anything; at some point, the server when it gets a new message, will usually be able to push a push a message onto your front end; whether it is the web based front end or onto your phone. Now, what happens with polling is that if I have a large number of clients that are polling the same server, they can potentially just flood the server with too many requests; and potentially that s even worse of a problem. Then if the server says look, I will let you know when I have a message or when I have an update. It depends as usual on the use case.

Now finally, webhooks versus APIs, we have already discussed this. An API is ultimately just a collection of endpoints and any client can send a request to an API. The primary sort of reason why we make a distinction at all between webhooks and APIs is that traditional APIs are meant more for retrieving data, more for not only for. But, webhooks are almost entirely used for pushing messages. But as we already saw, there are people who call webhooks just reverse API's. Or even a limited form of a lightweight API just use in order to communicate to a server. So, you cannot really say that they are completely distinct from each other. Webhooks, in other words, use a form of API to communicate between services.

(Refer Slide Time: 15:51)



How to... ?

- Consume webhook
  - Create URL endpoint, register with provider (eg. gitlab, twilio, ...)
- Debugging
  - requestbin: dummy endpoint to receive data and see content
  - curl or postman: API debugging tools
  - ngrok: public URL endpoint for private code without needing public IP
- Securing webhook
  - Restrict at IP access level? Difficult for public facing
  - Use API key/access token/header: eg. X-Gitlab-Token header

So, there are a number of issues that come up when you are implementing webhooks. One of them is for example, how do I consume a webhook? We already saw an example of how you could write code, the replete that I showed you. All that you need to do is write a piece of Python code, and register it with a server like flask. Make sure you have an a URL that can be accessed from whoever it is that is trying to send you information. If you are trying to run it on your own local machine that does not have a public IP address. There are services like ngrok that would allow you to provide a temporary public name; that can be used in order to connect to your system.

So, as far as consuming the webhook is concerned, you create your URL endpoint and registrate with whoever the provider is. But, this is webhooks can be a bit hard to debug; and as you can see, the reason is quite simple. The problem that is happening is that you know there is a lot of information in the webhook that might be sent by whoever is calling the webhook. And you need to know all of that information before you can do something useful with it. That is where things like request bin are exceptionally useful. They allow you to just get complete dumps of your data, which you can then look at and say okay, am I processing this correctly?

Of course, if you do not have if you can not use request bin, you could still create dumps of your data on your own server, as I showed you with the replete; but that is probably going to be even harder. You can make simple changes just to monitor what the data is so easily. There are other things curl, the command line URL access command, can be used as a very powerful API debugging tool. It can be used to post information to webhooks, or to other kinds of URL endpoints.

And generally speaking, allow you to debug not just webhooks, but any kind of APIs. And as I already mentioned, ngrok is something that allows you to get a publicly accessible URL, when in fact, you are trying to develop your flask code internally. So, it is something temporary, you cannot rely on it being there over a long duration; unless, you sign up for a paid plan with ngrok. But, once you have that you can get a tunnel through from that public domain into your local system, without needing to have a public IP address of your own. And along with consuming and debugging the webhook, one of the things that you need to be careful about is how do I secure it?

How do I make sure that it does not get abused by people who just keep posting random information to it? One thing you could do is restrict down to the IP access level. Let us say that I know that I only expect gitlab to call my webhook. I could even say that I am going to restrict the IPs that are allowed to connect to my webhook endpoint; and say that only the gitlab IPs are allowed to connect, that is reasonably good.

The problem with it is that what if gitlab changes their IPs at some point? I need to keep track of that and keep updating it. The alternative is that a register a secret token, tell it to gitlab; and say or tell gitlab OK, send it along with each request that you send. And that is sufficient, because all I need to do is on my receiver, I look for the token. If it is not there, I discard the request. I still have the problem that I might get bombarded with too many requests that I have to discard, but at least I know what to discard.