

# BUILDING A CISCO SPARK BOT ON THE RASPBERRY PI WITH PYTHON

## CONTENTS

About this Solution .....	2
About This Lab .....	2
Bots .....	2
Raspberry Pi .....	3
Pixel .....	3
Basic Oracle VirtualBox Configuration for Raspberry Pi Pixel .....	4
Configuring Pixel .....	9
Fix the Keyboard .....	9
Setup the Virtual Disk .....	9
USING Github and Grabbing The Scripts .....	13
Ngrok .....	14
Python, PIP and Scripts .....	16
developer.ciscospark.com - create bot and get ACCESS token .....	24
CHALLENGE - Add bot functionality - or Add Alexa to your bot .....	32
Alexa talks to the Pi .....	32
Tropo .....	36
Additional References .....	37

## ABOUT THIS SOLUTION

Cisco Spark is a next generation collaboration platform that can easily integrate with other advanced technologies and platforms through its well defined and highly capable Application Programming Interface (or API). Cisco provides APIs for many of our products and useful development resources at [developer.cisco.com](https://developer.cisco.com). A list of current APIs with relevant links to additional information is maintained at <https://developer.cisco.com/site/devnet/index/index.gsp>.

## ABOUT THIS LAB

The main activities in this lab are:

- Set up a Raspberry Pi (Raspbian Pixel in a Virtual Box VM) using basic Linux commands
- Install Python packages using pip
- Navigate and use Git/Github
- Understand and use Ngrok
- Use some basic Python
- Access and use the Cisco Spark REST API
- Understand and use Webhooks
- Build a functioning Bot

This lab should provide a shortcut method toward building and hosting your first bot.

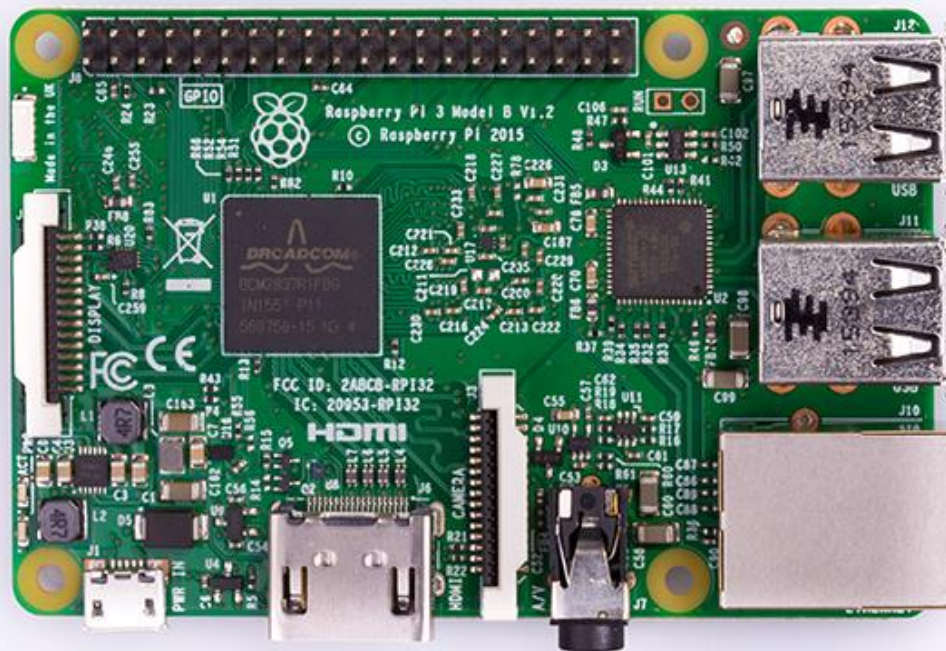
## BOTS

Bots can be powerful resources that can tie into next generation web based services and act as an interactive agent. By providing a natural language based interface and access to multiple applications, data sources, and machine learning or AI engines, Bots are poised to become a quickly growing human interface for information or event notification focused applications. Much as web site growth has been overtaken by the rise of mobile applications, bots are going to proliferate at a rapid pace. When combined with audio and natural speech engines, they become a new interface to displace contact centers, FAQs, event notification or incident response systems. They become a new machine to human and human to machine interface for numerous types of interactions.

Today we will help you build your first bot, so you can understand some of the underlying concepts and components involved.

## RASPBERRY PI

The Raspberry Pi is a Broadcom System on Chip (SoC) based small and inexpensive single board computer utilizing an ARM architecture. It was originally developed with the intention of providing students with a low cost computer for programming and other educational goals. It has risen to become a popular platform in the maker community, and given its support for Linux and low power it is a great platform to run persistent services for very low cost. Additional information can be found at <https://www.raspberrypi.org/>.



To purchase a Raspberry Pi and/or kit we recommend you use Amazon's smile program which will donate a portion of the purchase to a charity of your choice, for instance Children's Healthcare of Atlanta.

<https://smile.amazon.com/> - choose Children's Healthcare of Atlanta

[Purchase a Raspberry Pi3 on Amazon](#)

## PIXEL

The Raspberry Pi foundation recently announced the availability of Pixel, the next version of their Raspberry Pi Linux known as Raspian. The Pixel release is the first to provide an x86 version in addition to the traditional ARM version. This allows us to run the same operating system and many of the same tools in a Virtual Machine on a tradition x86 based laptop for our lab. Before our lab begins, please consider doing the following prework:

Install Pixel into VM - ISO is here: [http://downloads.raspberrypi.org/pixel\\_x86/images/pixel\\_x86-2016-12-13/2016-12-13-pixel-x86-jessie.iso](http://downloads.raspberrypi.org/pixel_x86/images/pixel_x86-2016-12-13/2016-12-13-pixel-x86-jessie.iso)

Follow this guide to build your VM

<http://www.penguintutor.com/raspberrypi/rpi-pixel-virtualbox>

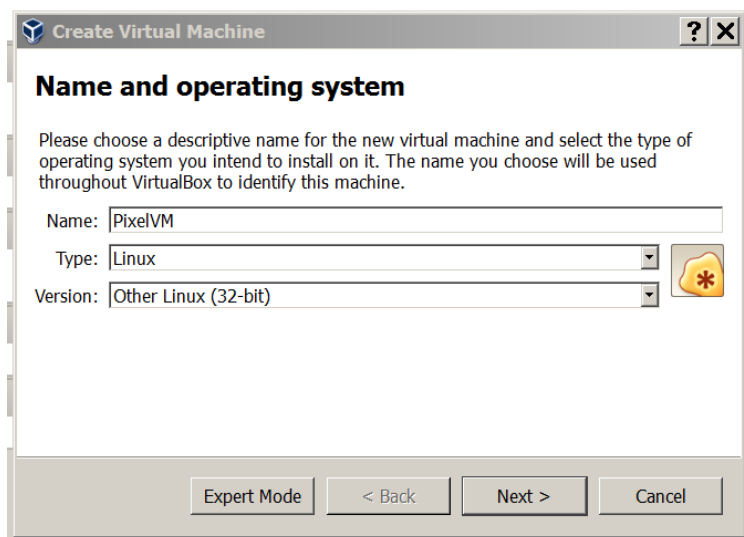
Or this Video

[Install Raspberry Pi Pixel on a VM in Oracle VirtualBox](#)

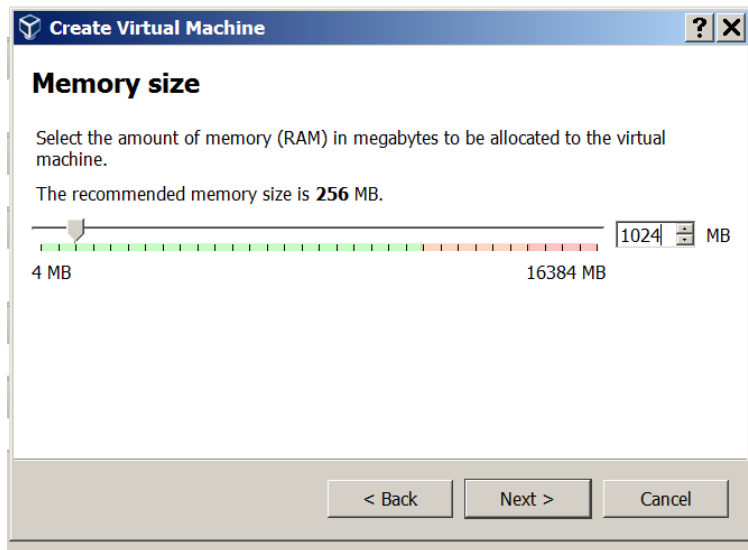
## BASIC ORACLE VIRTUALBOX CONFIGURATION FOR RASPBERRY PI PIXEL

After launching VirtualBox – choose to create a new Virtual Machine(VM) and use the selections in the following screen shots.

Set a Name for your VM, then set Type as Linux and Version as Other Linux(32-bit). Click Next.

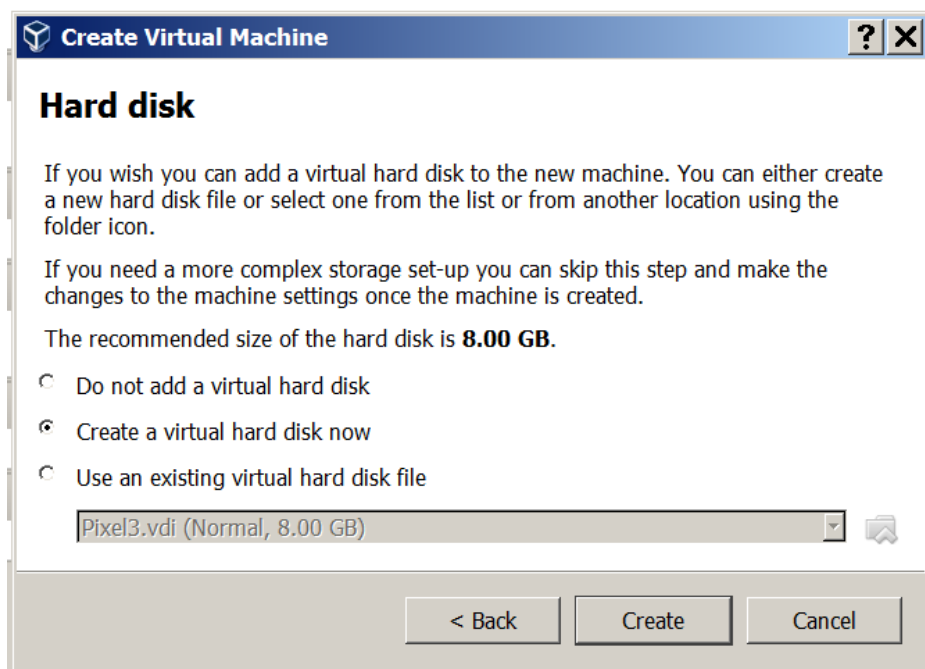


Set the Memory for your VM – 1 GB is recommended but 512 MB will run. Click Next.

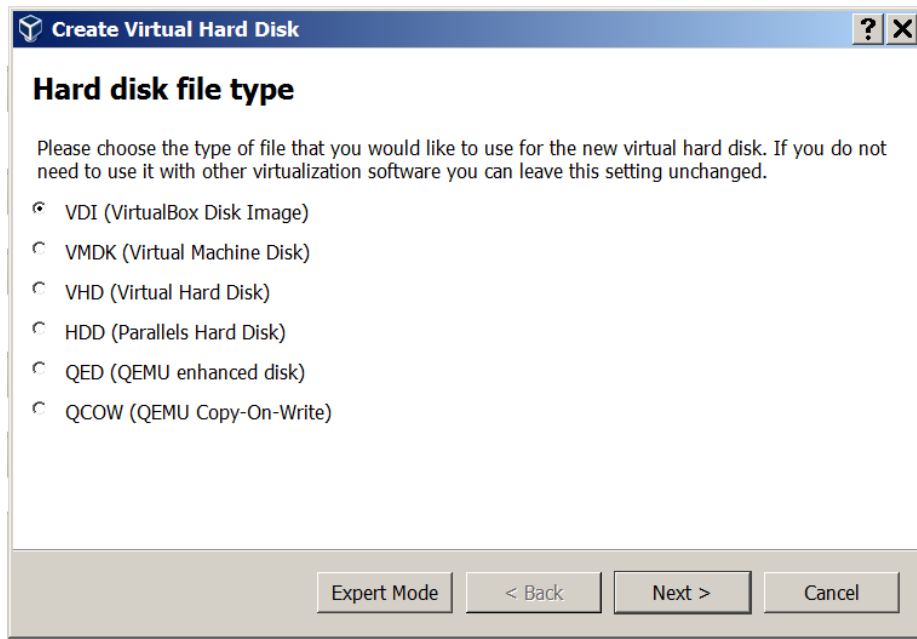


### Step 3 – Create a virtual Hard Disk

For step 3 simply accept the various defaults and then choose a file name and location to store the virtual disk. Since we will select dynamic allocation the actual disk used on your host system will only be around 50 MB to present the 8 GB disk. Click Create.



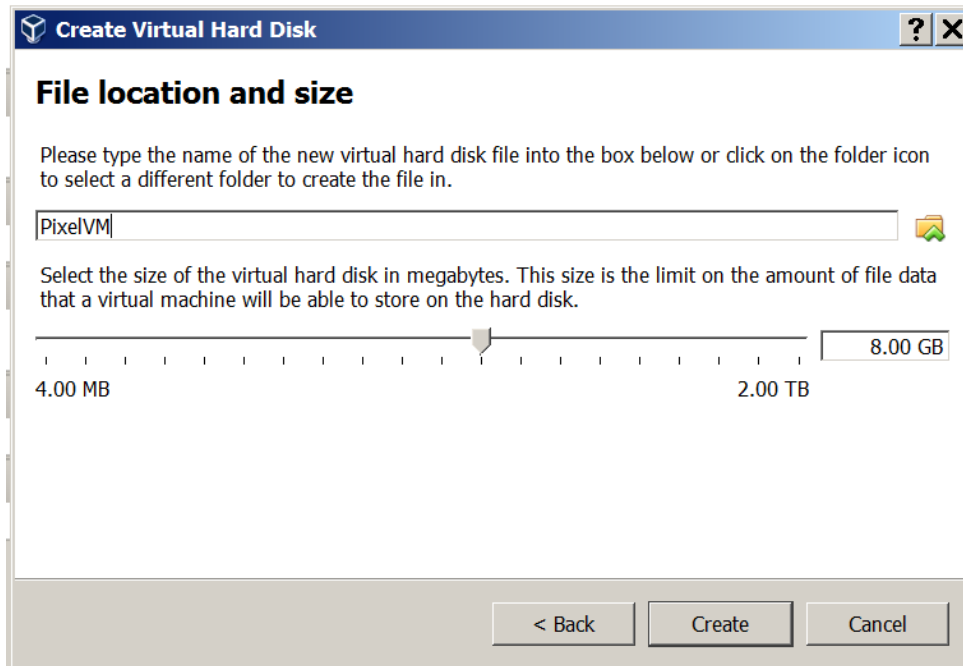
Click Next.



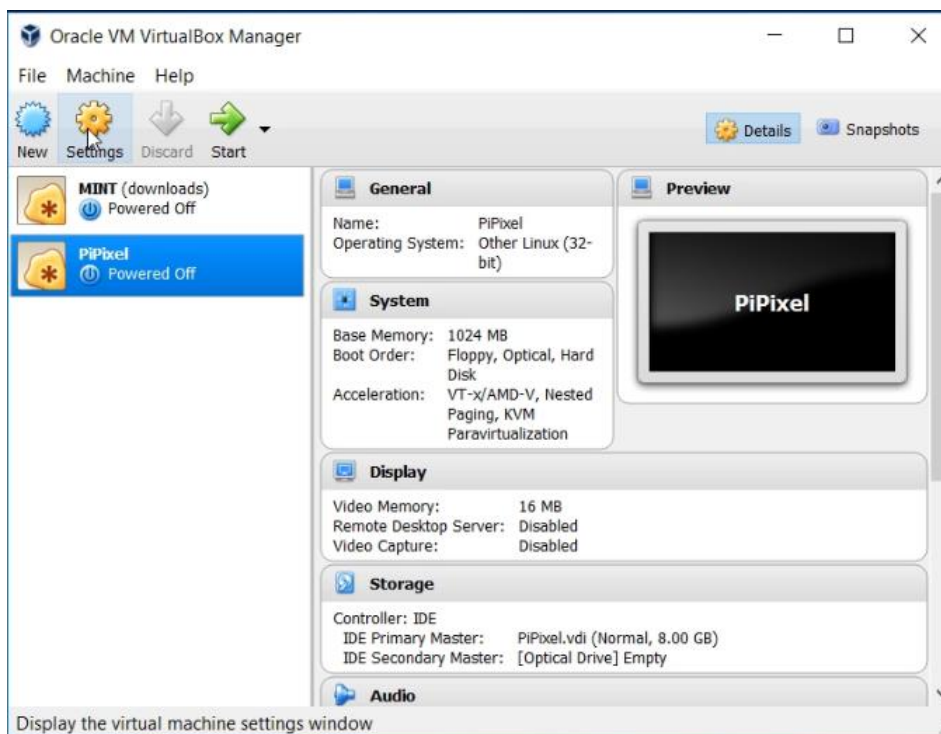
Click Next.



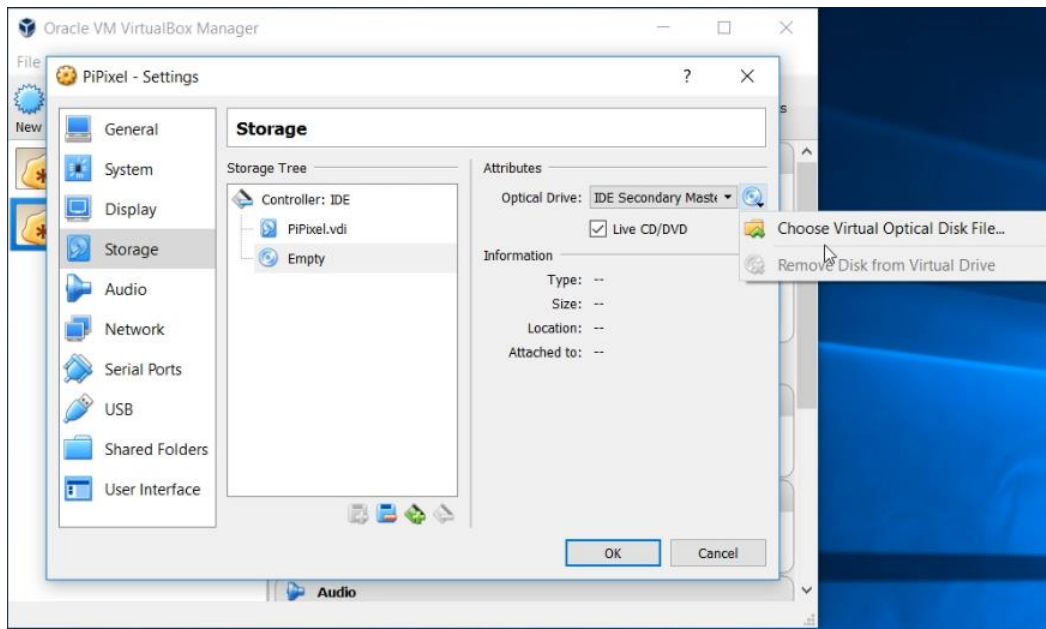
And finally click Create.



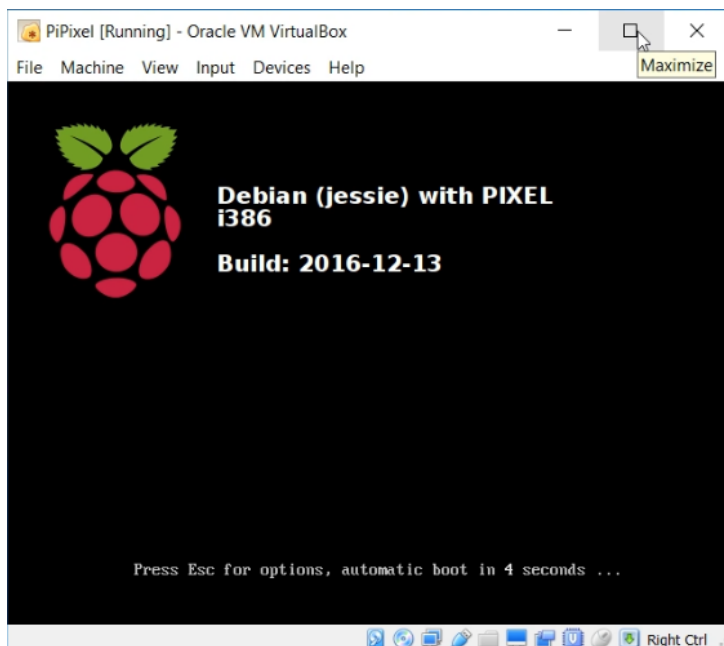
Next we need to create a virtual CD/DVD and point to our downloaded Pi Pixel image. Choose Settings.



Next choose storage, select 'Empty' under Controller:IDE and then click the small DVD image on the top right to select the Pi Pixel ISO you downloaded earlier from where you saved the file.



Once complete simply run the VM from Virtual Box and you will be presented with a Raspberry Pi Pixel splash screen within a few seconds, and a desktop interface with a minute or so.





Choose “Scaled Mode” from the View menu at the top toolbar to stretch the screen if needed.

## CLASSROOM DISCUSSION – LINUX 101

### CONFIGURING PIXEL

Once you are presented with a desktop, you will need to make some changes to Pixel before we begin the lab.

**This first release of Pixel doesn’t include an installer, so if you have to reboot Pixel you will lose all changes you made including any created files or installed packages.** So we have a permanent place to save our work, we are going to partition, format and mount our Virtual Disk created in the earlier steps.

---

### FIX THE KEYBOARD

First we need to change the keyboard from UK to US. Click the Open Applications Menu (the Raspberry on the top left) > Preferences > Mouse and Keyboard Settings

Select the Keyboard Tab then Press the Keyboard Layout button the lower right

Choose United States for Country and English(US) for Variant [scroll to top]

---

### SETUP THE VIRTUAL DISK

Next open a terminal windows so we can install gparted. Gparted is a package that will give us a graphical interface to partition and format the 8 GB virtual disk we created earlier. This will become the location for us to store our scripts so we don’t lose work if we reboot the VM.

Launch a terminal (Click the shortcut at the top of the screen) and enter each of the following commands:

When you see the beginning of a line like this: “pi@raspberrypi:~ \$” please note this is equivalent to the command line you should see in your terminal window. Please note that “~” denotes your home directory (/home/pi/) and “~/code” denotes our working directory (/home/pi/code/).

First we will update our package list. Packages in Linux are effectively programs and all their dependencies that need to be installed to work. These pre-compiled binaries are managed via a package manager. We will be using Apt via the ‘apt-get’ commands to install needed programs. Sudo is used to give us “super-user” rights in Linux. By default Linux is more secure by rigorously implementing user permissions. ‘sudo’ allows us to run commands as the system administrator.

First we update our list of available packages:

```
pi@raspberrypi:~ $sudo apt-get update
```

Then we install gparted:

```
pi@raspberrypi:~ $sudo apt-get install gparted
```

Answer ‘Y’ at the prompt and hit [Enter] to continue

Next we create a directory, “code”:

```
pi@raspberrypi:~ $mkdir code
```

Confirm that “code” is now a directory:

```
pi@raspberrypi:~ $ls -alr
```

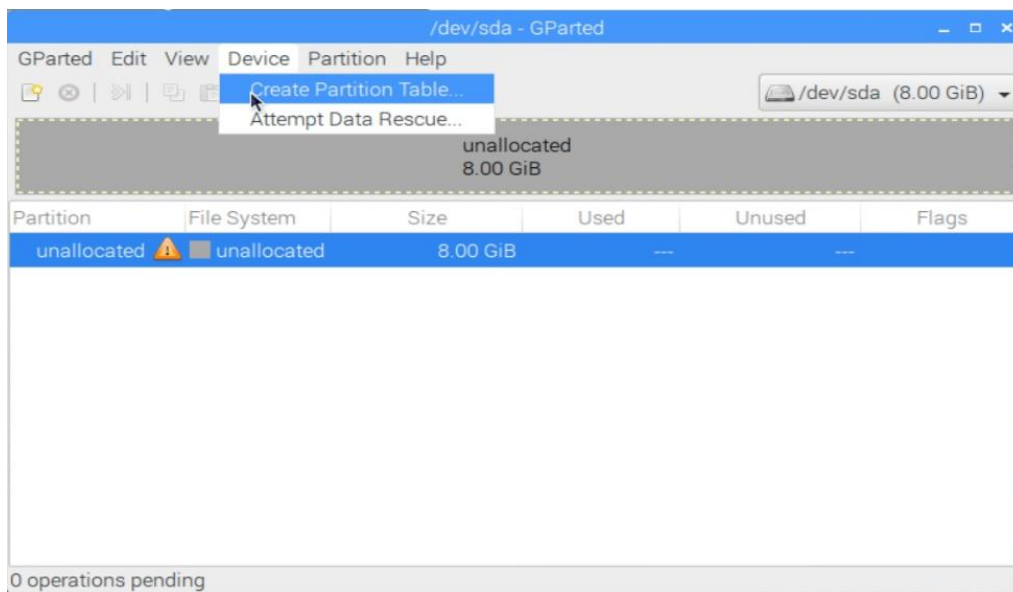
Then launch gparted – this will open a window in the GUI

```
pi@raspberrypi:~ $sudo gparted
```

---

## PARTITION THE HARD DRIVE AND FORMAT WITH GPARTED:

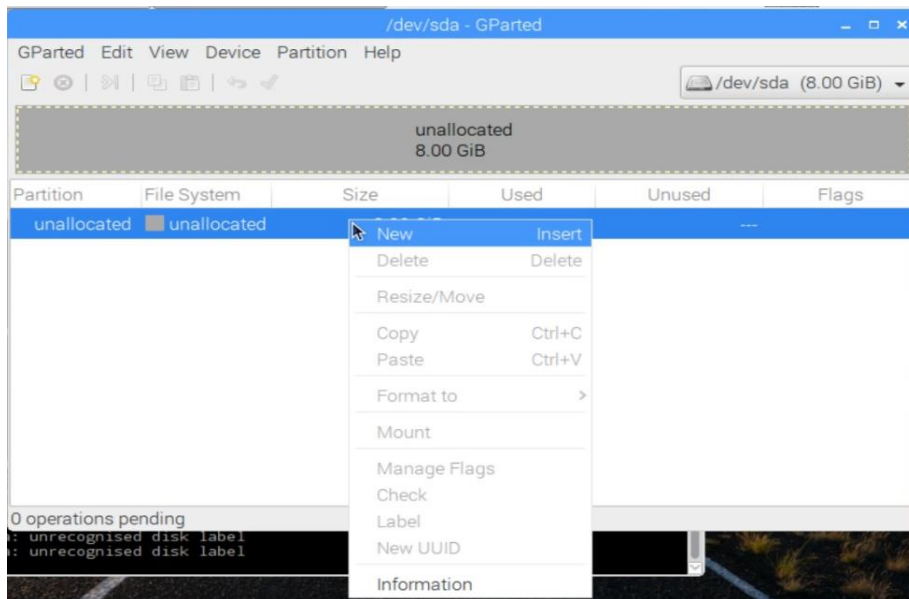
Choose Create Partition Table from the Device Pull Down Menu



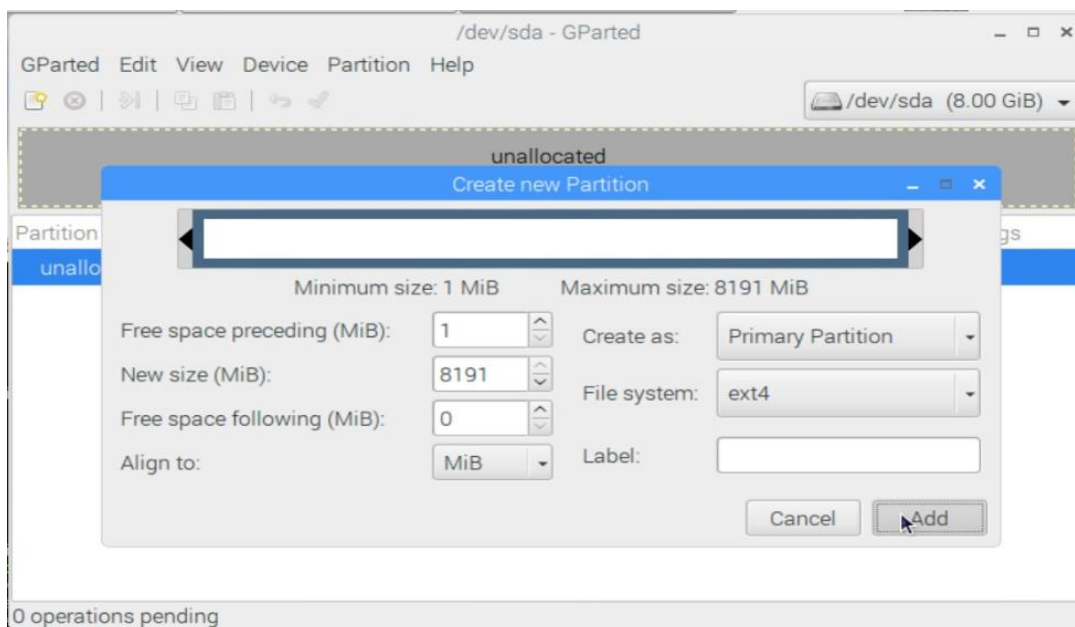
Click Apply



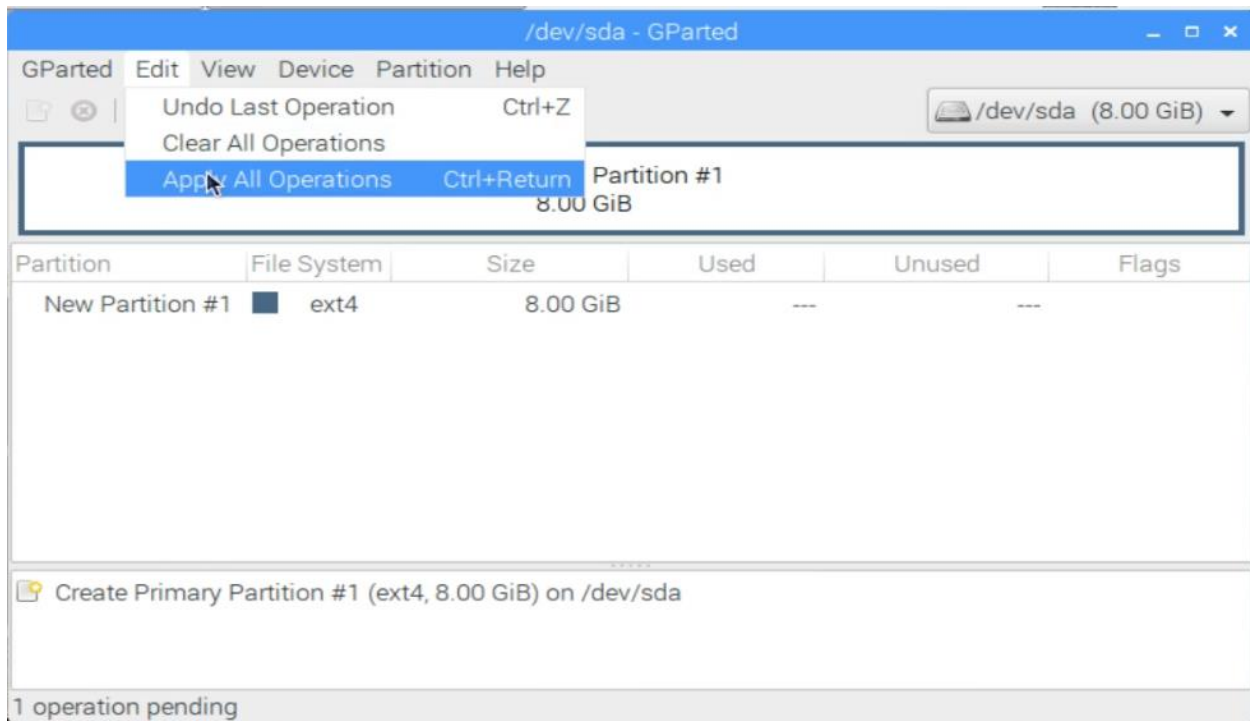
Right click the unallocated partition and choose New from the menu



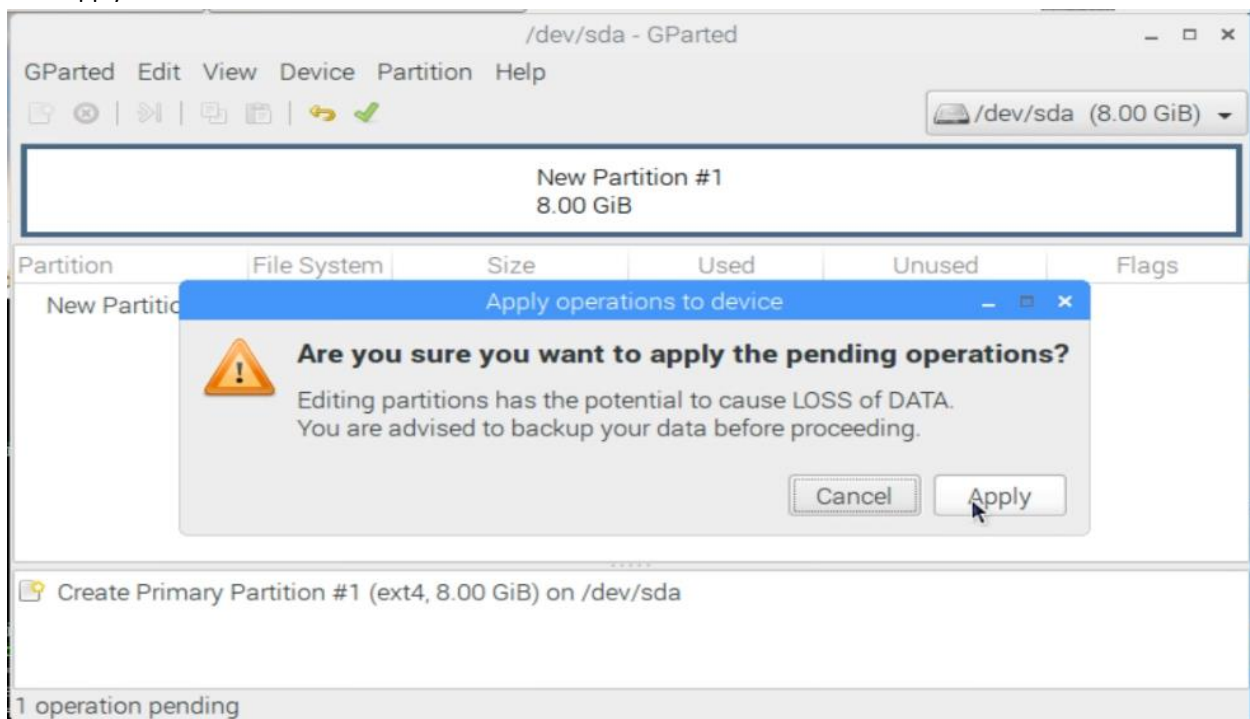
Click Add – accept the defaults



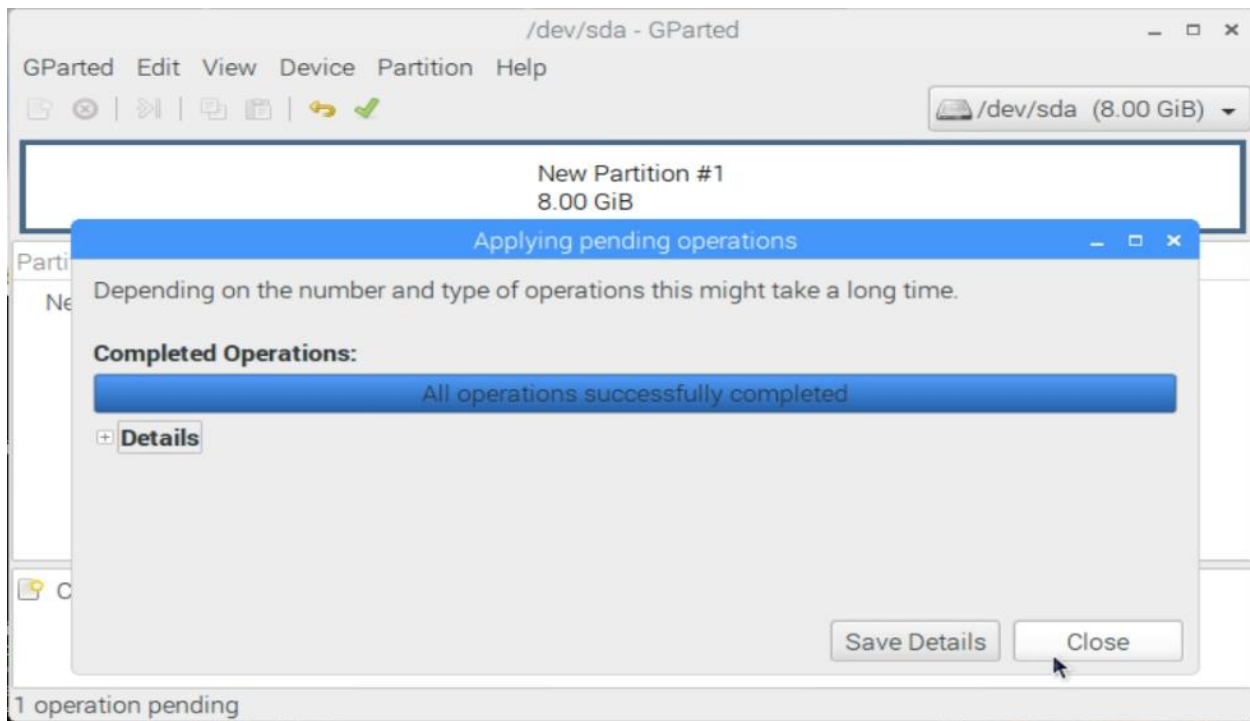
Choose Apply All Operations from the Edit Pull Down Menu



Click Apply



Click Close



## MOUNT, SET THE PERMISSIONS AND OWNERSHIP SO WE CAN WRITE TO THE DISK

Linux requires that a hard drive volume be mounted, and the mount point set in the directory structure. In our case we will mount the hard drive directly to our ~/code directory.

```
pi@raspberrypi:~ $sudo mount /dev/sda1 /home/pi/code/
```

Since we are using the superuser to mount the hard disk, we need to set the proper ownership and permissions to let the default system user, pi, use the resource.

```
pi@raspberrypi:~ $sudo chown -R :users /home/pi/code/
```

```
pi@raspberrypi:~ $sudo chmod -R g+rw /home/pi/code/
```

Use the command 'df' to confirm that the disk is mounted, mounted to the correct location and has available space. The command 'ls -alr' can confirm that the ~/code directory has its ownership and permissions set.

## USING GITHUB AND GRABBING THE SCRIPTS

### COPY THE EXAMPLE PYTHON SCRIPTS WE WILL BE USING

First we need to make sure we are in the right directory where we want to copy the files.

```
pi@raspberrypi:~ $cd code
```

Next we need to copy the files from Devnet's Github project site to our local drive. First open a browser and navigate to the url <https://github.com/CiscoDevNet/ciscosparkapi>. You will see instructions on how to install the DevNet Cisco Spark Python SDK and at the top of the screen you will see multiple files and directories available in

the project repository. If you don't have a Github account you can create one and start building your own repositories for your own projects in the future. Go back to the command line and use the 'git' command to copy the contents of this repository locally:

```
pi@raspberrypi:~/code $git clone https://github.com/CiscoDevNet/ciscosparkapi
pi@raspberrypi:~/code $cp ./ciscosparkapi/examples/botexample.py ./
pi@raspberrypi:~/code $cp ./ciscosparkapi/examples/ngrokwebhook.py ./
```

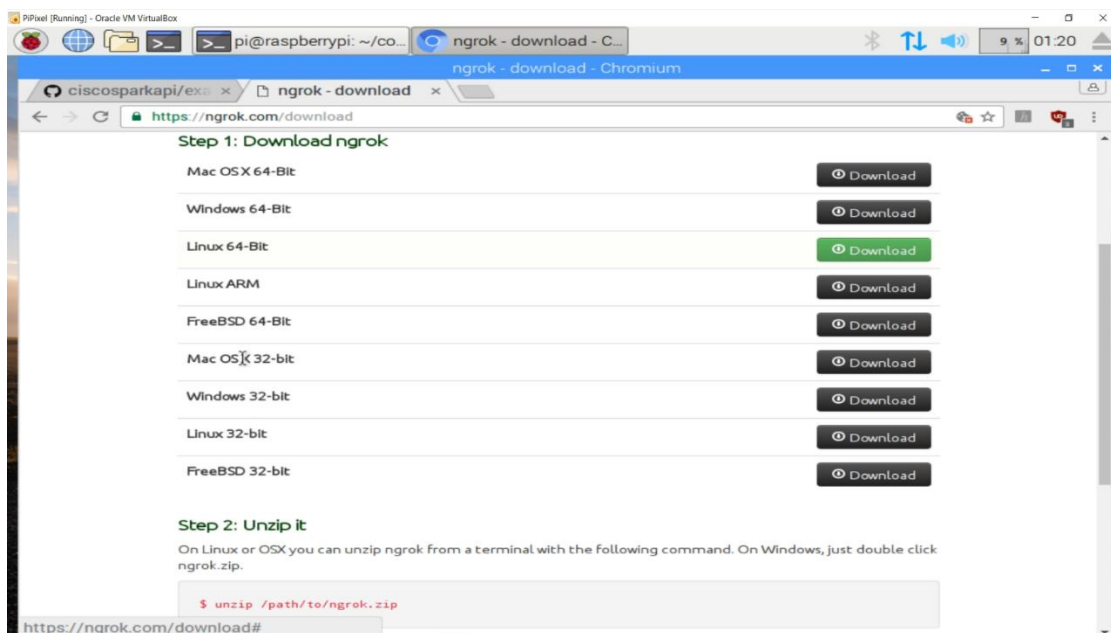
Use 'ls -alr' to verify that the 2 python scripts were copied as expected. We will come back to these scripts once we install Ngrok.

### CLASSROOM DISCUSSION – GIT/GITHUB

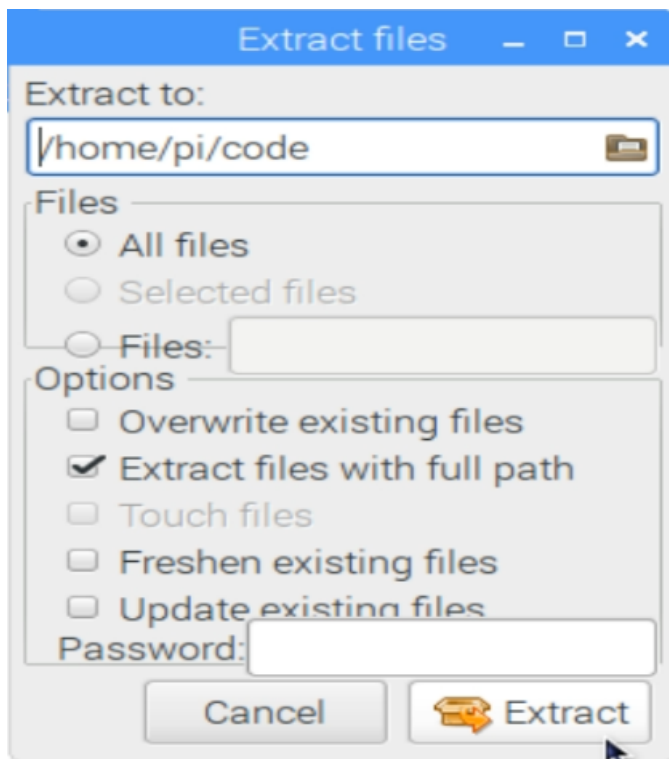
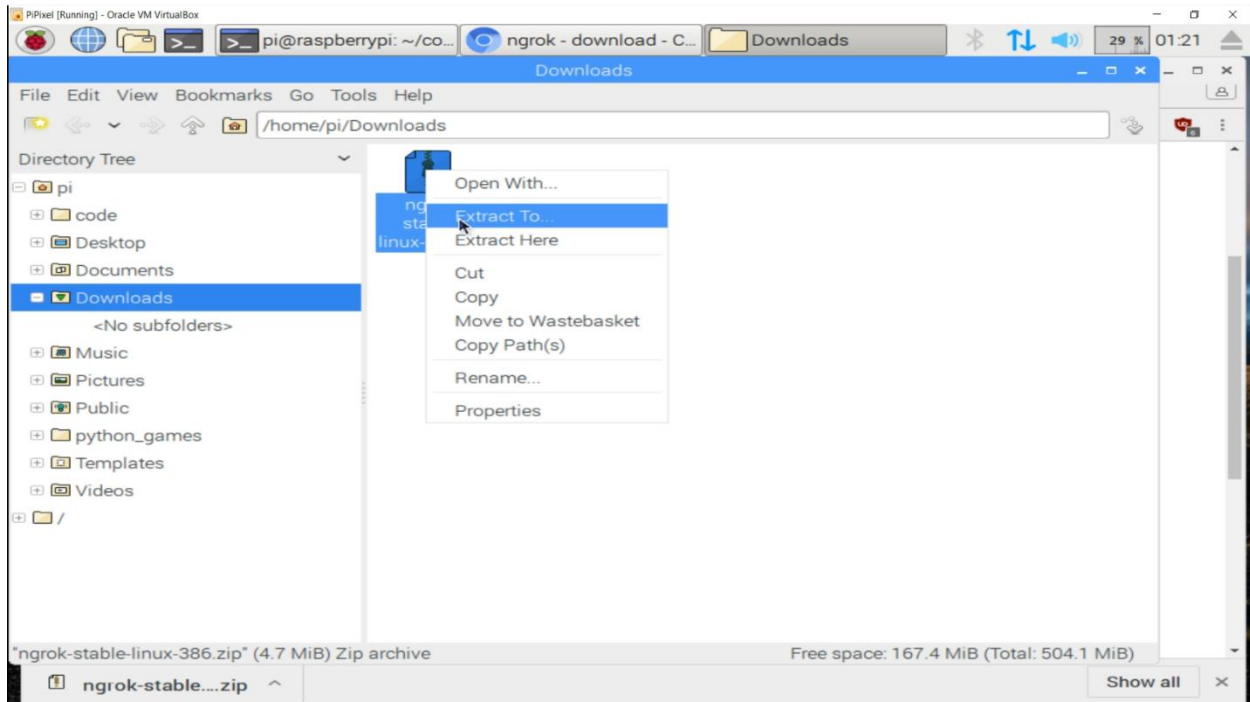
## NGROK

Ngrok will provide Firewall traversal and additional monitoring tools. When new messages are posted in a Cisco Spark room, we can configure a Webhook so that Spark will notify our script so that it might take action. Since our VM (or Pi) sits behind at least one or more firewalls and a NAT, there is no way for us to address the Webhook properly to reach out environment. Ngrok creates a tunnel out to their service, where they host a publicly accessible Internet URI for us to receive those requests from Spark. When received, the request will be relayed back to our VM (or Pi) and then the ngrok process will forward to a running service of our choosing.

Open a browser window and download the relevant version of ngrok from ngrok.com. **If you are running on a physical Raspberry Pi as opposed to the Pixel Pi VM, you will use the ARM version of ngrok. For this lab running in the VM, download the 32-bit Linux version.**



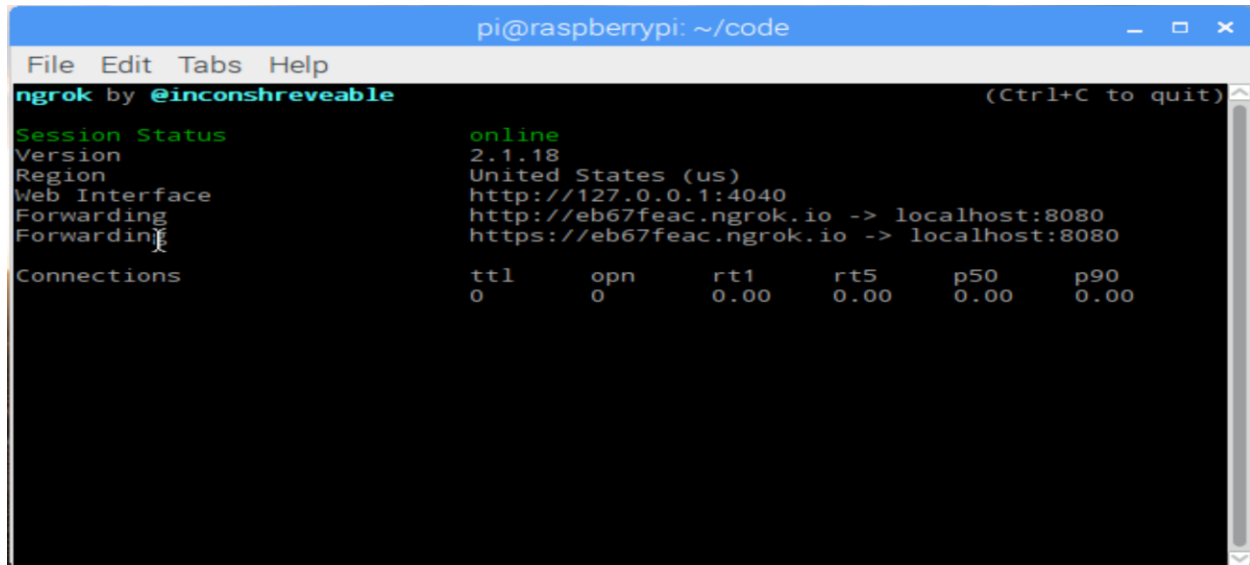
Once downloaded, extract the ngrok binary to the code directory.



Now test ngrok by running

```
pi@raspberrypi:~/code $ngrok http 8080
```

Ngrok will create a tunnel to a random URI and redirect all HTTP traffic to your localhost port 8080.



```
pi@raspberrypi: ~/code
File Edit Tabs Help
ngrok by @inconshreveable (Ctrl+C to quit)
Session Status      online
Version             2.1.18
Region              United States (us)
Web Interface        http://127.0.0.1:4040
Forwarding            http://eb67feac.ngrok.io -> localhost:8080
                     https://eb67feac.ngrok.io -> localhost:8080
Forwarding
Connections
  ttl    opn    rt1    rt5    p50    p90
    0     0     0.00   0.00   0.00   0.00
```

You can further validate ngrok is running by opening a local browser window to 'localhost:4040' to see the web based interface. You can then launch a browser from your phone or the host operating system and navigate to the URI in the ngrok interface. You will receive an error, but can confirm that ngrok sees the HTTP request via the command line or web interface.

#### CLASSROOM DISCUSSION – NGROK

### PYTHON, PIP AND SCRIPTS

#### CLASSROOM DISCUSSION – PYTHON

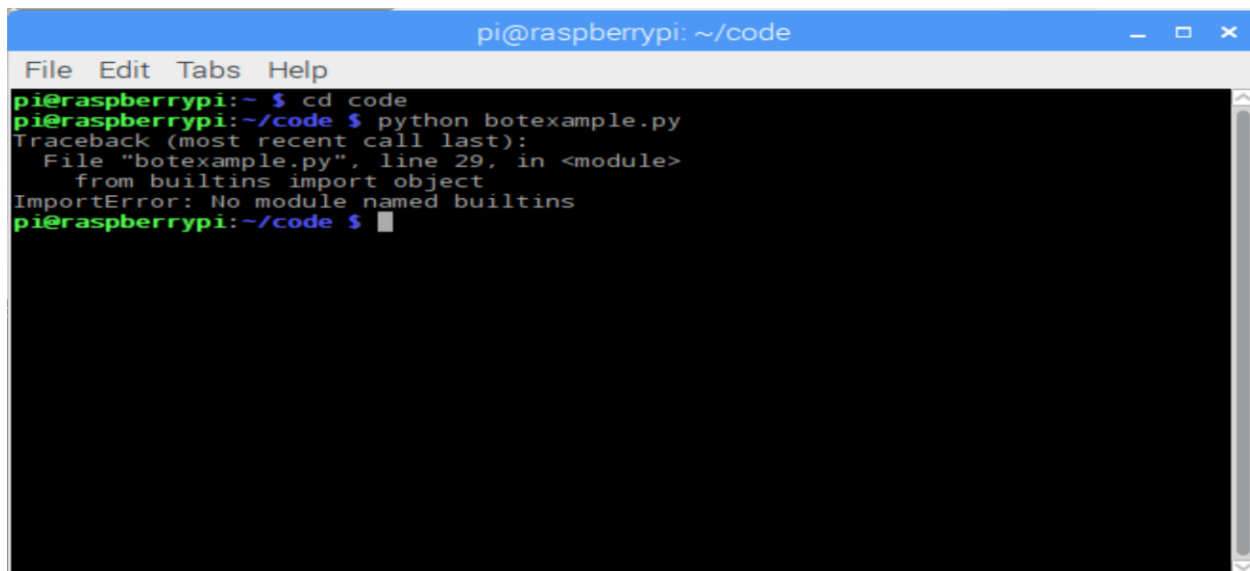
Now let's try to execute our sample script. Open a new Terminal window (the other one is still running ngrok).

```
pi@raspberrypi:~/ $cd code
```

```
pi@raspberrypi:~/code $python botexample.py
```

You will receive errors. When writing new scripts expect to see these error messages on a regular basis. This is how we determine what is wrong with our script, typically syntax, or other mistakes. In our case we are trying to run a script that needs additional libraries installed. Let's break down the error.





```
pi@raspberrypi:~ $ cd code
pi@raspberrypi:~/code $ python botexample.py
Traceback (most recent call last):
  File "botexample.py", line 29, in <module>
    from builtins import object
ImportError: No module named builtins
pi@raspberrypi:~/code $
```

The key is the second line – it mentions line 29, referring to our script. So let's open our script in an editor. Pixel includes Geany which can highlight Python scripts to make them easier to read. Launch the script in Geany and add a '&' to the end of the command so it runs in the background and doesn't take over our Terminal.

```
pi@raspberrypi:~/code $geany botexample.py &
```

You will notice that line 29 is one of a number of lines that include the terms 'import' or 'from', 'import'. These are packages in Python that this script needs to run that we haven't installed. Our next step is to install the missing packages. By default different Python installations may already contain certain packages. In our case we are only missing the web.py and CiscoSparkAPI packages – the other ones are already installed.

We use pip to manage packages (or modules) in Python. Let's install the needed modules:

```
pi@raspberrypi:~/code $sudo pip install web.py
pi@raspberrypi:~/code $sudo pip install ciscosparkapi
```

Now let's look at the script in the text editor and go through each section.

This first section is mostly comments, with the exception of the first line. This is used in Linux to point to the python2 vs python3 binary. Web.py only works with Python 2.x not 3.x.

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""A simple bot script.

This sample script leverages web.py (see http://webpy.org/). By default the
web server will be reachable at port 8080 - append a different port when
launching the script if desired. ngrok can be used to tunnel traffic back to
```

your server if you don't wish to expose your machine publicly to the Internet.

You must create a Spark webhook that points to the URL where this script is hosted. You can do this via the `CiscoSparkAPI.webhooks.create()` method.

Additional Spark webhook details can be found here:

<https://developer.ciscospark.com/webhooks-explained.html>

A bot must be created and pointed to this server in the My Apps section of <https://developer.ciscospark.com>. The bot's Access Token should be added as a

'SPARK\_ACCESS\_TOKEN' environment variable on the web server hosting this script.

NOTE: While this script is written to support Python versions 2 and 3, as of the time of this writing `web.py` (v0.38) only supports Python 2.

Therefore this script only supports Python 2.

"""

The 'from' and 'import' tells the Python interpreter to load modules, or parts of modules. Objects, classes, and methods can be imported specifically. The last line is specifically importing a class called `Webhook` from the `CiscoSparkAPI` so we can use later in the script to associate received data as a `Webhook` and not have to manually parse the JSON returned from the Cisco Spark service to use it in Python.

```
from __future__ import print_function
from builtins import object
import json
import web
import requests
from ciscosparkapi import CiscoSparkAPI, Webhook
```

Here we define a variable that points to the URL to retrieve a cat fact from a public internet service.

```
# Module constants
CAT_FACTS_URL = 'http://catfacts-api.appspot.com/api/facts?number=1'
```

Open a browser and navigate directly to the URL – you will notice that response page has a very specific format. This is JSON. We use specific formats like JSON to make it easier for our programs to parse the data and grab the

pieces we need. JSON is quickly becoming the defacto standard for REST APIs because of its historic widespread use in Java and Javascript plus its easy human readability. If we grab the data and format it by adding some carriage returns, it is easier to see the components:

```
{ "facts": ["Cats with long, lean bodies are more likely to be outgoing, and  
more protective and vocal than those with a stocky build."], "success":  
"true" }
```

```
{  
  
    "facts": ["Cats with long, lean bodies are more likely to be outgoing,  
and more protective and vocal than those with a stocky build."]  
  
    ,  
  
    "success": "true"  
  
}
```

The beginning and ending curl braces are there to make the beginning and end. Next you have a list of name-value pairs. The first name is “facts” and the value of facts is “Cats with long, lean bodies are more likely to be outgoing, and more protective and vocal than those with a stocky build.”. We then have a comma to separate the next name-value pair. What is the next name and next value?

The square brackets denote a list, meaning there could be 1 or more values associated with the name. Look at our original URI now in your browser and change the “number=1” to “number=2” and refresh the page. You will now see 2 facts, separated by a comma. It is important you notice the square brackets and list or you may not parse the JSON correctly. When returning a list item, you have to also tell python the item number in the list (starting your count from 0).

In our script we imported a module called JSON to help us easily parse JSON in Python. We will discuss the details later in the script.

Next we see some more variables. “urls” is going to be used by our web.py module which is defined in app variable on the next line. Notice the syntax “web.”, the ‘.’ states that the application object is part of the web.py module we imported. If we imported a module named “module” then any object or method in the module we wanted to call would follow “module.object” or “module.method()”. To find what objects or methods are available in a module you want to read its documentation. In this case the application object in web.py is what defines the web service we will be running. By creating the app variable we are in effect creating the web service. Notice that we pass the urls as an argument. “urls” is providing the specific address on our webservice and tells app that we expect to receive a webhook at that address.

Last we define the CiscoSparkAPI object. This **is** going to give us an error later because we haven’t provided the object with an Access Token – more on that when we run the script.

```
# Global variables
```

```

urls = ('/sparkwebhook', 'webhook')      # Your Spark webhook should point
to http://<serverip>:8080/sparkwebhook

app = web.application(urls, globals())    # Create the web application
instance

api = CiscoSparkAPI()                    # Create the Cisco Spark API
connection object

```

Next we define a function in python with the “def” keyword. In this case we call the function catfact and use the () to denote that we aren’t expecting any parameters. The “""" defines a comment at the top of our function.

```

def get_catfact():

    """Get a cat fact from appspot.com and return it as a string.

    Functions for Soundhound, Google, IBM Watson, or other APIs can be added
    to create the desired functionality into this bot.

    """

```

Next we use the requests module to do the hard work of actually grabbing the JSON data from the URL we defined earlier. The great thing about modules is that we don’t have to learn how to write a script to manually create and send a HTTP GET request or parse the response and headers. You can search for modules to talk to all sorts of services and via various protocols to make your script development much, much faster. “verify=False” is technically not needed, but is often used in requests if you want to ignore SSL certificate errors.

```

response = requests.get(CAT_FACTS_URL, verify=False)

```

The next line uses the requests module to convert the return text into a Python dictionary. In effect it is parsing the JSON for us.

```

response_dict = json.loads(response.text)

```

Look at this next line and compare to the JSON we saw earlier. Notice the brackets here help us to navigate through the returned name-value pairs and then through the list. So to reiterate the value of the ‘facts’ was a list with a single fact, so we have to add the ‘[0]’ to parse the list and return the first (and only in this case) item.

```

return response_dict['facts'][0]

```

It is important to understand how to navigate through the dictionary as most of the services you will access via REST APIs will have their own name-value pairs and the values may be strings or lists. To get the data you need, you will often print the raw response to the command line and then figure out how to get to the info you want.

The next block of code is probably the most difficult to understand if you don’t have a background in Object Oriented Programming. It defines a class called ‘webhook’ and then defines a single method called POST for itself. Basically this syntax comes from the web.py documentation. It basically defines what we meant by ‘webhook’ earlier when we passed it to app earlier in the script. Go back and look at these 2 lines:

```

urls = ('/sparkwebhook', 'webhook')

```

```
app = web.application(urls, globals())
```

Next we define the webhook method that we associated with our <http://<yourip>:8080/sparkwebhook> as a way to handle post requests to this URI. So we define a function called POST to tell the module what to do when it receives a HTTP POST. Notice we don't define any other common HTTP methods like GET, etc.. so if one hits our script it literally won't know what to do it.

The last line sets a new variable json\_data to the data that is passed from the POST.

```
class webhook(object):  
  
    def POST(self):  
  
        """Respond to inbound webhook JSON HTTP POSTs from Cisco Spark."""  
  
        json_data = web.data() # Get the  
POST data sent from Spark
```

Next we print some info to the screen when a HTTP POST is received by our service:

```
print("\nWEBHOOK POST RECEIVED:")  
  
print(json_data, "\n")
```

Then we use the CiscoSparkAPI's Webhook class to parse the json\_data for us. Notice I didn't have to type ciscosparkapi.Webhook because I used the "from" .. "import" at the beginning of our script.

```
webhook_obj = Webhook(json_data) # Create a  
Webhook object from the JSON data
```

This next line uses the CiscoSparkAPI to actually get a room object that includes all the room details, based on the roomid in the webhook object we received.

```
room = api.rooms.get(webhook_obj.data.roomId) # Get the  
room details
```

Next we do the same for the message details.

```
message = api.messages.get(webhook_obj.data.id) # Get the  
message details
```

And now for the person details.

```
person = api.people.get(message.personId) # Get the  
sender's details
```

We then print some specific pieces of data from the detail objects by using the notation that CiscoSparkAPI defines. So we created a room object, and in that object we store the title of the room, to access it we use the notation "room.title". We will look at the raw JSON data that Cisco Spark returns later in the lab.

```

print("NEW MESSAGE IN ROOM '{}'.format(room.title))

print("FROM '{}'.format(person.displayName))

print("MESSAGE '{}'\n".format(message.text))

```

Next we have a loop prevention section. The idea is simple, if we receive a webhook that points to a message that was written by our bot, we do nothing, otherwise we parse the message to look for commands to our bot and act on those. Without this check we could create a loop where the bot immediately and constantly responds to its own messages. The Cisco Spark platform implements rate controls, but it is still a waste of computing and network resources on both sides to inadvertently create a loop. Always ensure you have a loop control mechanism in place with any script that will create messages or other objects within Cisco Spark or any other web service.

“me” defines the bot in this case since it will be the access token we use for this script. The `message.personID` is a unique encoded identifier for the person that created the message. The `me.id` is that same identifier specific to the bot. If they are equal we know the bot is the message creator. We can then exit via the “return”.

“else” defines what to do when the id’s don’t match. Here we look in the message text for the string “/CAT” and if we see it we provide some output to the command line and then use our `get_catfact()` function to grab an actual fact. We then print that fact to the local command line.

```

# This is a VERY IMPORTANT loop prevention control step.

# If you respond to all messages... You will respond to the messages
# that the bot posts and thereby create a loop condition.

me = api.people.me()

if message.personId == me.id:

    # Message was sent by me (bot); do not respond.

    return 'OK'

else:

    # Message was sent by someone else; parse message and respond.

    if "/CAT" in message.text:

        print("FOUND '/CAT'")

        cat_fact = get_catfact()

# Get a cat fact

print("SENDING CAT FACT '{}'.format(cat_fact))

```

Next we create a new message in the Spark room where we received the original message. Notice we pass the `room.id` and the text we returned from our `cat_fact` function. We then use “return” to exit.

```

        response_message = api.messages.create(room.id,
text=cat_fact)      # Post the fact to the room where the request was received

    return 'OK'

```

Technically these 3 lines are our program, there is just a lot going on behind the scenes with the `app.run()` line. The first line is seen commonly in Python scripts, it basically says that if you are running this script directly then do the following. That way if we used `import` to pull this script in as a module, the script we pulled this into wouldn't automatically run this portion.

```

if __name__ == '__main__':

    # Start the web.py web server

    app.run()

```

Now that we have gone through and discussed each section of the script, let's try to run it:

```
pi@raspberrypi:~/code $python botexample.py
```

....and BOOM! – an error, as promised:

```

pi@raspberrypi:~ $ cd code
pi@raspberrypi:~/code $ python botexample.py
Traceback (most recent call last):
  File "botexample.py", line 46, in <module>
    api = CiscoSparkAPI()
ciscosparkapi.exceptions.ciscosparkapiException: You must provide an Spark access token to interact with the Cisco Spark APIs, either via a SPARK_ACCESS_TOKEN environment variable or via the access_token argument.
pi@raspberrypi:~/code $

```

Look at line 46 in your editor –

```
api = CiscoSparkAPI()
```

This is where we are instantiating the `CiscoSparkAPI`. Notice the error on the command line – we didn't pass it a Spark Access Token. We need to go get one and then find a way to pass it to the object in the script. First though, let's add a mechanism to read the Access Token from a local file. By default the `CiscoSparkAPI` can read the Access Token from a Global Variable, the problem with this is when we want to run multiple bots from the same script or platform, it is helpful to be able to point to local files instead to keep track of our Access Tokens.

Add the following script to `botexample.py` above the '# Global Variables' line:

```

#grab the at from a local at.txt file instead of global variable

fat=open ("at.txt","r+")

at=fat.readline().rstrip()

```

```
fat.close
```

The above script creates a variable that points to a file object. The file object is created using the integrated Python `open()` function. We open a file in the local directory call “at.txt” and in read mode “r+” since we don’t need to write to the file. Next we set our ‘at’ variable to the string in the first line of the file using the integrated file buffer function `readline()`. The integrated string function `.rstrip()` will remove any extra whitespace. Alternatively we could have broken this into multiple lines of script, instead we use the ‘.’ to call methods related to the previous object. Last we close the file with `fat.close`.

Next we edit the `CiscoSparkAPI` instantiation line to include this ‘at’ variable:

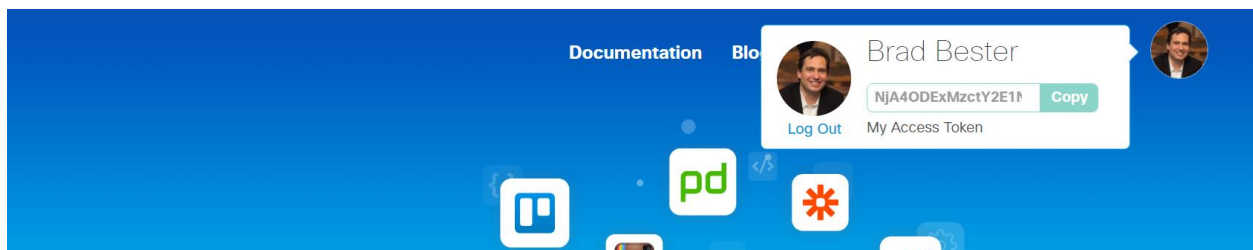
```
api = CiscoSparkAPI(at)
```

Now we need to get a bot access token and place into a file in the code directory name ‘at.txt’.

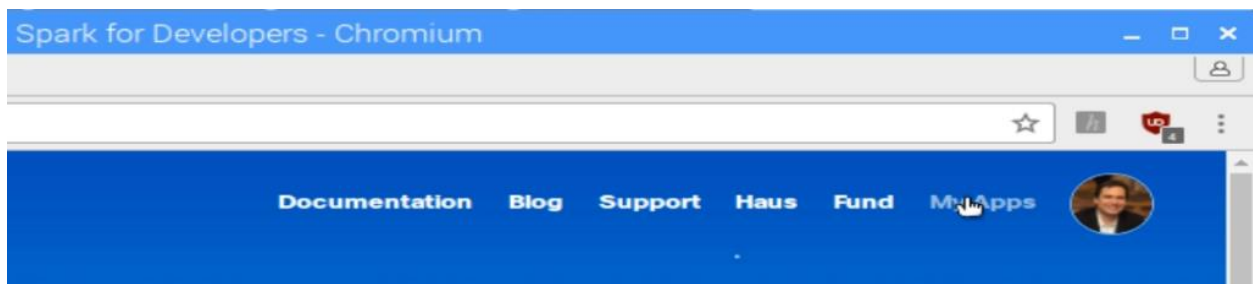
## DEVELOPER.CISCOSPARK.COM - CREATE BOT AND GET ACCESS TOKEN

Navigate your VM’s browser to `developer.ciscospark.com`. If you haven’t created a login do so now.

You will notice your profile picture in the top right of the screen once you have logged in.

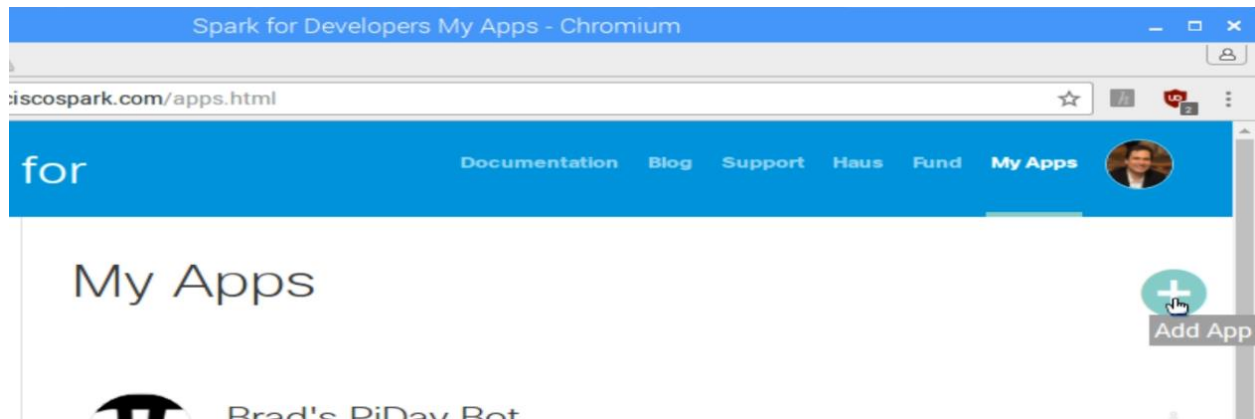


If you hover your cursor you will see your personal user Access Token. DO NOT USE THIS ACCESS TOKEN. You can use it in scripts, but then posted messages will come from you, not your bot. Instead click “My Apps”.

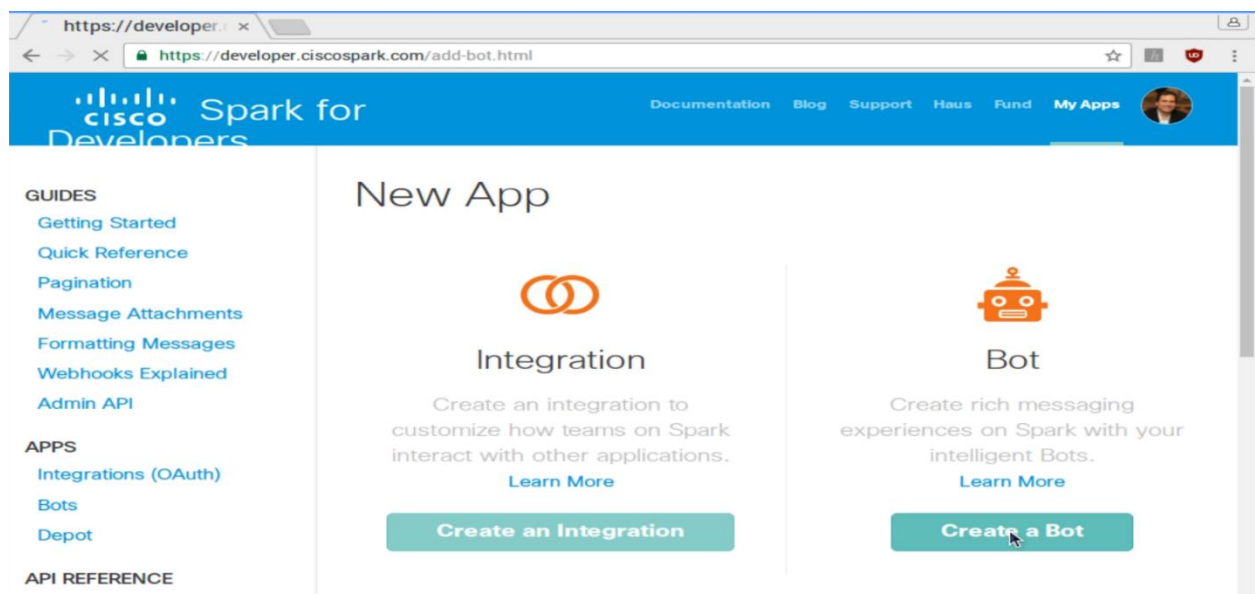


Next Add an App by clicking the large plus sign on the top right of the screen.





Next choose “Create a Bot”



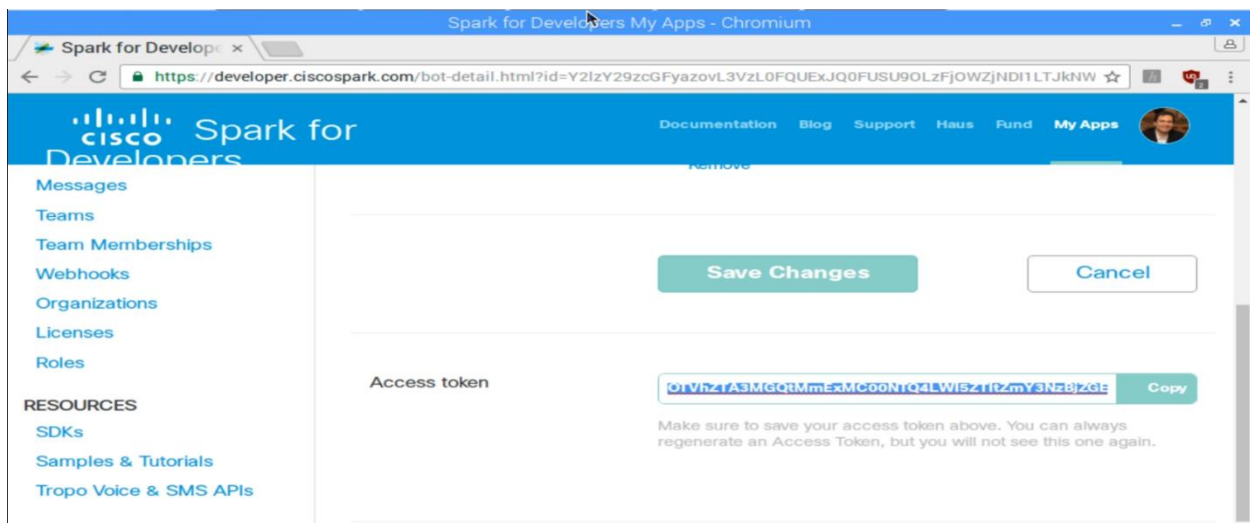
And then give your bot a unique name. Use a combination of your email address (to the left of the @) and Piday to endure your bot's name is unique – use it in both the Display Name and Bot Username fields. Remember your bot's name so you can add it to the group Spark room later.

The screenshot shows the 'New Bot' form in the Spark for Developers web application. The browser address bar shows the URL `https://developer.ciscospark.com/add-bot.html`. The page has a blue header with the Cisco Spark logo and navigation links: Documentation, Blog, Support, Haus, Fund, and My Apps. A left sidebar contains a 'GUIDES' section with links like 'Getting Started', 'Quick Reference', and 'Admin API', as well as 'APPS' and 'API REFERENCE' sections. The main content area is titled 'New Bot' and contains three input fields: 'Display Name' (with a placeholder 'I'), 'Bot Username' (with a placeholder '@sparkbot.io'), and 'Icon' (with a placeholder URL). Each field has a descriptive text below it. The 'Bot Username' field is marked as 'Cannot be changed later.'

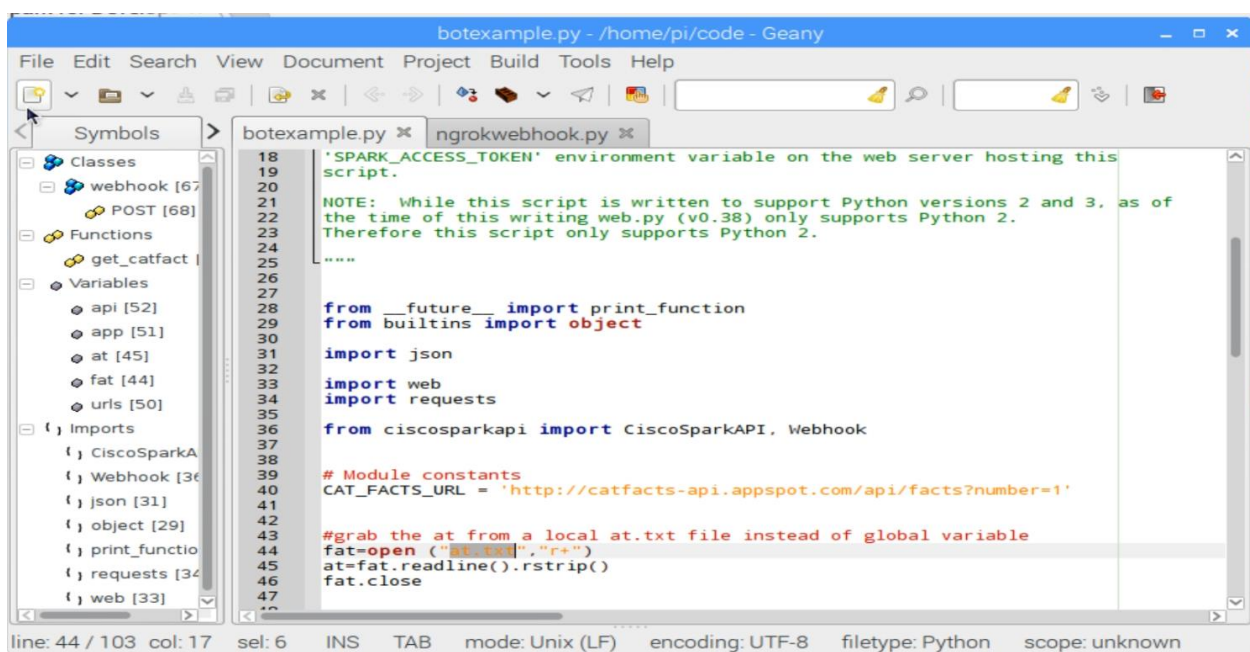
Next point to the URL of an image. A quick trick is to use Google Images to find a URL to an image 512x512 – try to use creative commons or other free images. Once finished choose “Add Bot”.

This screenshot shows the same 'New Bot' form, but now the 'Icon' field contains the URL `http://riv.zcache.com/black_and_white_pi_symbol_coa`. The 'Add Bot' button is highlighted with a mouse cursor, and the 'Cancel' button is also visible. The 'Bot Username' field still shows the 'Cannot be changed later.' message.

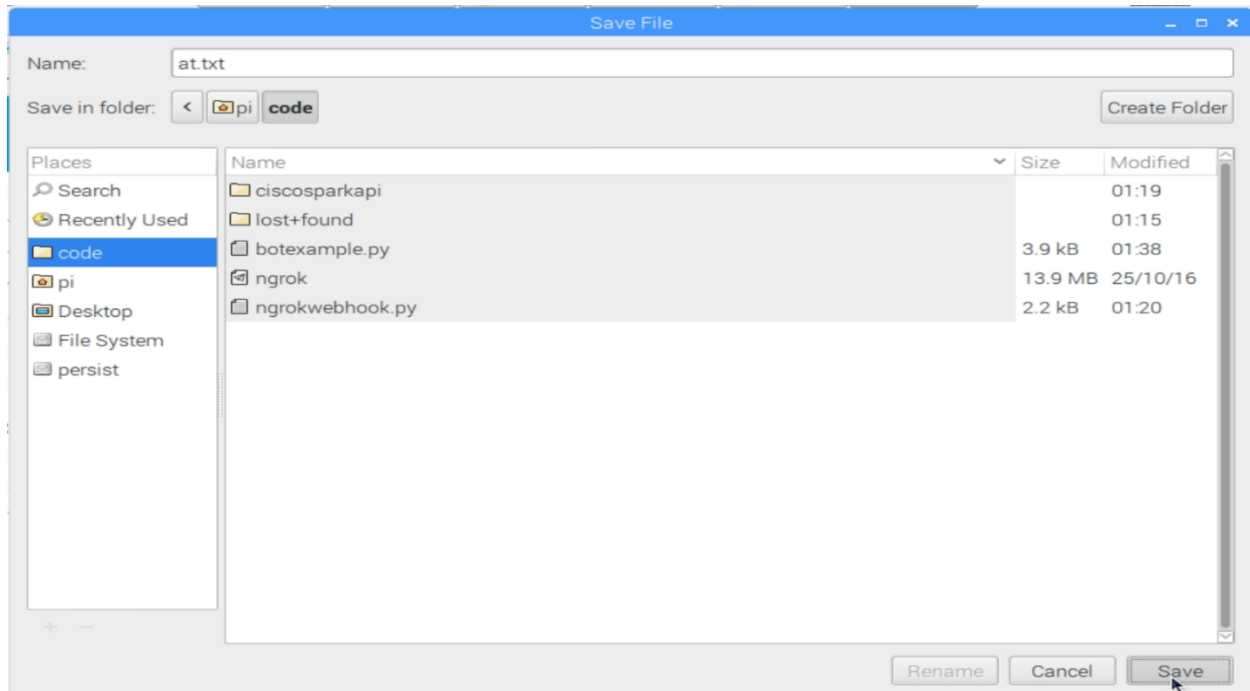
Once your bot is saved, you should be redirected to a screen that includes the access token for your bot. Press the copy button and go back to the Geany editor.



Create a new file in Geany and paste the Access Token into the first line.



Now save the file as "at.txt" in the /home/pi/code/ directory.



Now that we have the access token entered properly we can go ahead and get our bot going. Run the `botexample.py` script by opening a terminal and typing

```
pi@raspberrypi:~/ $cd code
pi@raspberrypi:~/code $python botexample.py
```

Next you we need to connect the ngrok tunnel to Cisco Spark's cloud platform by enabling a Webhook within Cisco Spark. To do this we have provided a script that will automatically read the ngrok address and create a Webhook in Spark. Here is a copy of that script - `ngrokwebhook.py`. Read through it and try to understand how it works....

```
#sample script that reads ngrok info from localhost:4040 and create Cisco
Spark Webhook

#typicall ngrok is called "ngrok http 8080" to redirect localhost:8080 to
Internet

#accesible ngrok url

#

#To use script simply launch ngrok, then launch this script. After ngrok is
killed, run this

#script a second time to remove webhook from Cisco Spark
```

```

import requests

import json

import re

import sys

import requests.packages.urllib3

requests.packages.urllib3.disable_warnings()

from ciscosparkapi import CiscoSparkAPI, Webhook


def findwebhookidbyname(api, webhookname):

    webhooks = api.webhooks.list()

    for wh in webhooks:

        if wh.name == webhookname:

            return wh.id

    return "not found"


#Webhook attributes

webhookname="testwebhook"

resource="messages"

event="created"

url_suffix="/sparkwebhook"


#grab the at from a local at.txt file instead of global variable

fat=open ("at.txt","r+")

at=fat.readline().rstrip()

fat.close


api = CiscoSparkAPI(at)

```

```

#go to the localhost page for ngrok and grab the public url for http
try:

    ngrokpage = requests.get("http://127.0.0.1:4040").text
except:

    print ("no ngrok running - deleting webhook if it exists")

    whid=findwebhookidbyname(api, webhookname)

    if "not found" in whid:

        print ("no webhook found")

        sys.exit()

    else:

        print (whid)

        dict=api.webhooks.delete(whid)

        print (dict)

        print ("Webhook deleted")

        sys.exit()

for line in ngrokpage.split("\n"):

    if "window.common = " in line:

        ngrokjson = re.search('JSON.parse\(\\"(.+)\\"\\);',line).group(1)

        ngrokjson = (ngrokjson.replace('\\"', ''))

print (ngrokjson)

Url = (json.loads(ngrokjson) ["Session"] ["Tunnels"] ["command_line
(http)"] ["URL"])+url_suffix

print (Url)


#check if the webhook exists by name and then create it if not
whid=findwebhookidbyname(api, webhookname)

```

```

if "not found" in whid:

    #create

    print ("not found")

    dict=api.webhooks.create(webhookname, Url, resource, event)

    print (dict)
else:

    #update

    print (whid)

    dict=api.webhooks.update(whid, name=webhookname, targetUrl=Url)

    print (dict)

```

From the above script the following section really defines the attributes for the webhook:

```

#Webhook attributes

webhookname="testwebhook"

resource="messages"

event="created"

url_suffix="/sparkwebhook"

```

It basically translates to “send a webhook to <http://<our ngrok address>/sparkwebhook> whenever a new message is created”. Cisco Spark bots are designed so they only see new messages that are directed to them. This means we can place our bot in any room and when someone directs a message to the bot this webhook will forward that notification to our running bot code.

Run the ngrokwebhook.py script and it will alternatively create, update or delete the webhook based on whether ngrok is running, or if the ngrok address has changed between runs. When you are done with the bot, please terminate ngrok with Ctrl-C and then run the ngrokwebhook.py again to delete the webhook.

Open a terminal and run the ngrokwebhook.py script:

```

pi@raspberrypi:~/ $cd code
pi@raspberrypi:~/code $python ngrokwebhook.py

```

Now open your phone or a browser on the HOST (not the VM) and login into Cisco Spark via the app or the website ([ciscospark.com](http://ciscospark.com)). Go into our lab room and add your bot. Now test you bot by typing the ‘@’ sign followed by your bot’s name – it should appear in the dropdown – just click. Next type /CAT as your message to your bot.

You should see a response in the Spark room. Take a look at the output in your open terminal windows and in the <http://127.0.0.1:4040> ngrok web interface. You see 2 webhooks – 1 when your bot received a message from you and 1 when your bot posted its message to the room.

## CHALLENGE - ADD BOT FUNCTIONALITY - OR ADD ALEXA TO YOUR BOT

Now that you have a working bot, what else can you do with it?

By defining new functions and searching for new key terms in the message, you can literally make your bot do just about anything you want. Do a search of the web for APIs and you find literally 10s of thousands – with new ones showing up daily. Some great resources to start your journey:

<https://market.mashape.com/explore> - Mashape Marketplace, a directory of APIs with great examples of how to access

<https://www.programmableweb.com/> - A listing of over 16,000 APIs and growing – search for things that interest you

What if you want to talk to Cisco devices just like you can talk to these web service?

<https://developer.cisco.com/site/devnet/index/index.gsp> contains a listing and references to dozens of Cisco APIs.

Here is a sample function to go to the Internet Chuck Norris Database, grab a random Chuck Norris joke, and then change “Chuck” to “Steve” and “Norris” to “Koenig” before returning. What other work would you need to do to integrate into our existing botexample.py script?

```
def get_steve():
    URL = 'http://api.icndb.com/jokes/random?exclude=[explicit] '
    print "URL:"+URL+"\n"
    resp = requests.get(URL, verify=False)
    resp_dict = json.loads(resp.text)
    joke = resp_dict["value"]["joke"]
    joke = joke.replace ("Chuck", "Steve")
    joke = joke.replace ("Norris", "Koenig")
    joke = joke.replace ('"', '"')
    return joke
```

Many sites will require you create and maintain login credentials to ensure you aren’t over using their APIs. From a security perspective it is best to keep the credentials outside of your script using environment variables or the file technique we used in our example script.

---

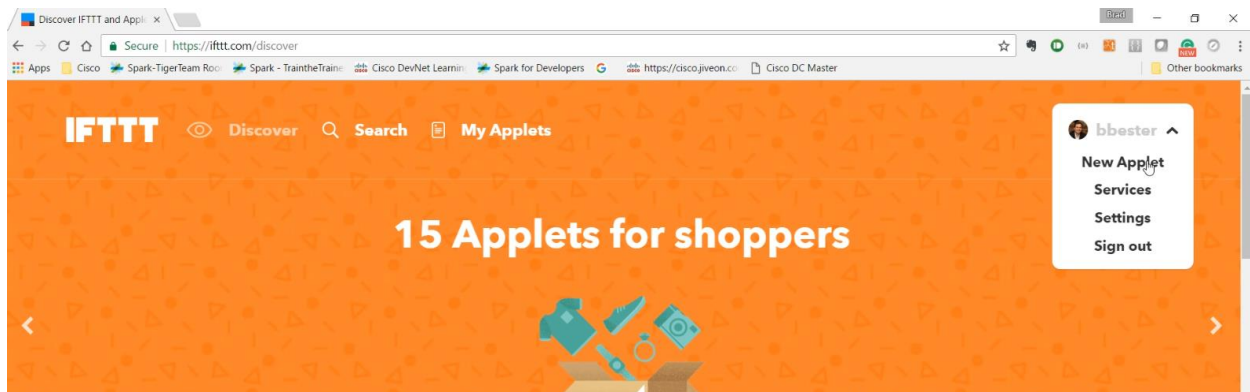
## ALEXA TALKS TO THE PI

How about adding voice commands via Alexa and IFTTT to your bot? Here are simple instructions to get IFTTT to trigger a webhook to your bot.

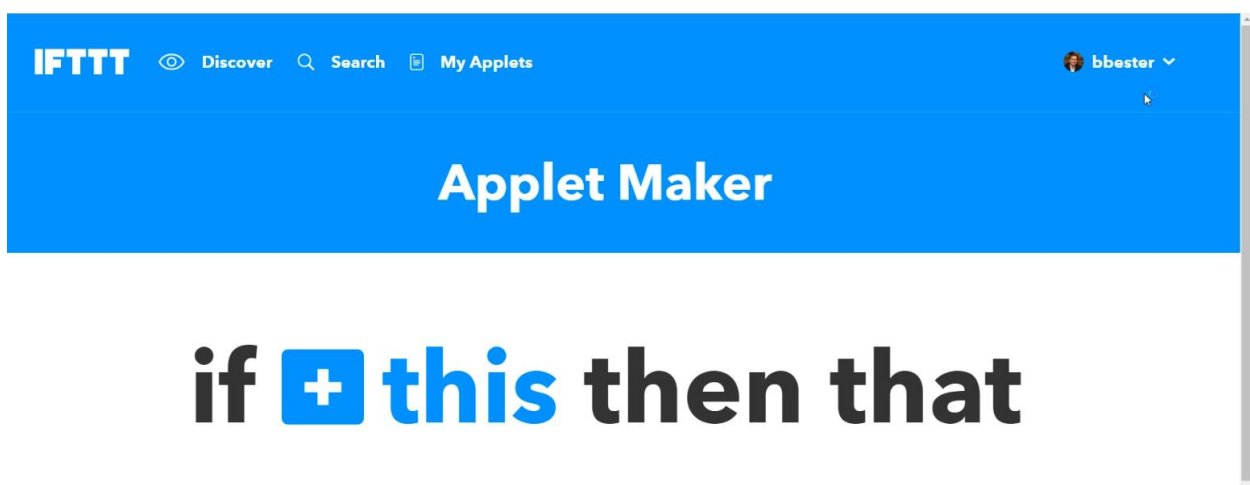
If you don’t own an Amazon Echo or Dot, you can try out echosim.io (an Echo simulator) for free.

Go to IFTTT and create a new Applet (sign up for an account if you don’t have one, it’s free):

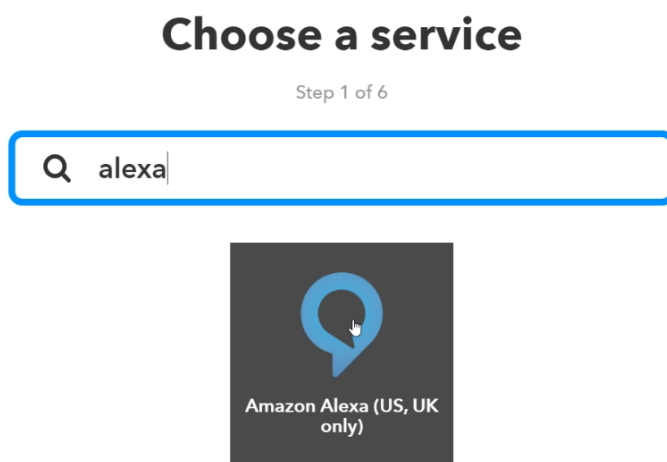




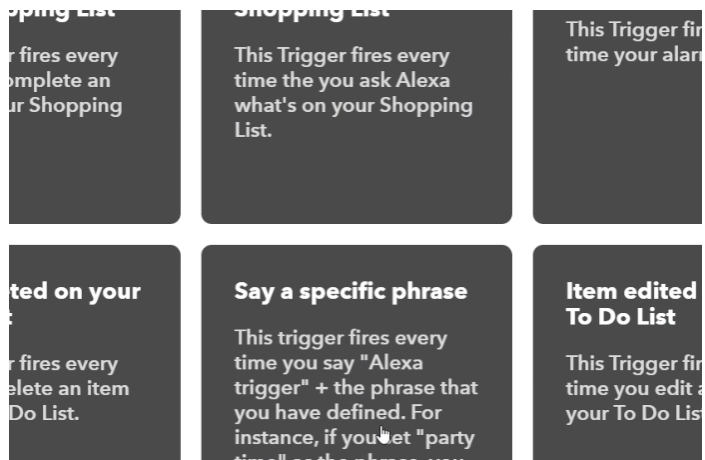
Next click 'this'.



Search for the Amazon Alexa service and click the icon.



Find and click “say a specific phrase”:



Set the phrase – “spark” in our case:

The image shows the 'Say a specific phrase' trigger configuration screen. The text reads:

**Say a specific phrase**

This trigger fires every time you say "Alexa trigger" + the phrase that you have defined. For instance, if you set "party time" as the phrase, you can say "Alexa trigger party time" to have your lights loop colors. Please use lower-case only. Neither German characters (Umlaute/Eszet) nor their long-form equivalents (ae, oe, etc.) are currently supported – support is coming soon.

What phrase? \*

spark

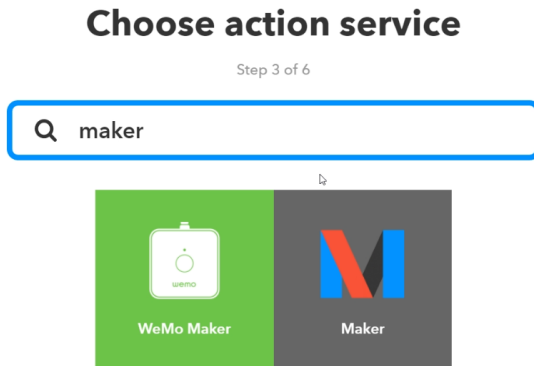
Use lower-case characters only

Create trigger

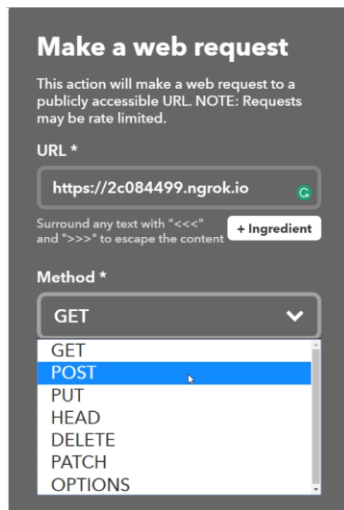
Now let's set “that” to the channel Maker and define our JSON webhook we want IFTTT to send. Click “that”.



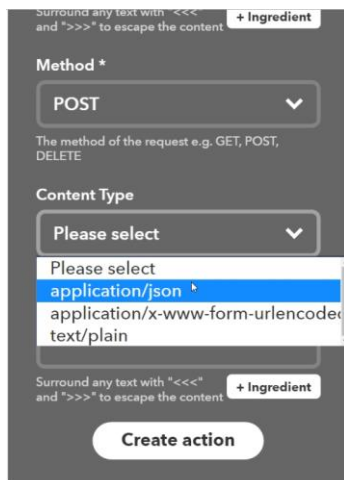
Search for and select 'Maker':



Now you can set your Ngrok URL and the method to POST to mirror a webhook.



Select Applications/JSON for content type.



Last we define the data we want this trigger to post. That way you can make unique triggers for as many actions as you want your bot to perform.



Surround any text with "<<<" and ">>>" to escape the content. + Ingredient

**Method \***

POST

The method of the request e.g. GET, POST, DELETE

**Content Type**

application/json

Optional

**Body**

`{"ifttt": "true", "text", "spark"}`

Surround any text with "<<<" and ">>>" to escape the content. + Ingredient

Create action

Once you create this action edit your botexample.py to handle this webhook. (Hint: create an if statement that checks for certain fields in the incoming webhook to differentiate between Spark and IFTTT POSTs)

---

## TROPO

<https://www.tropo.com/>

Tropo can be used to integrate your bot into traditional calling or SMS testing as well, either in conjunction or separate from Cisco Spark. Try to create a function in your script that can call your phone and read any posted message to your bot (toll charges may apply).

## ADDITIONAL REFERENCES

- Additional Labs and Resources
  - <https://developer.cisco.com/>
- Linux References
  - <http://lifehacker.com/5633909/who-needs-a-mouse-learn-to-use-the-command-line-for-almost-anything>
  - [http://linuxcommand.org/learning\\_the\\_shell.php](http://linuxcommand.org/learning_the_shell.php)
- Python Reference sites
  - [http://www.davekuhlman.org/python\\_101.html#operators](http://www.davekuhlman.org/python_101.html#operators)
  - <https://wiki.python.org/moin/BeginnersGuide/Programmers>
  - <https://www.codecademy.com/learn/python>
- Infrastructure/Network Scripting with Python Info
  - Cisco
    - Cisco Github for Data Center <https://github.com/datacenter>
    - UCS <http://www.cisco.com/c/en/us/support/servers-unified-computing/ucs-director/products-programming-reference-guides-list.html>
    - Spark <https://developer.ciscospark.com/quick-reference.html>
    - Call Manager <https://pypi.python.org/pypi/pyaxl/1.0.1>
    - Network Gear
      - <https://pynet.twb-tech.com/blog/automation/netmiko.html>
      - <https://pynet.twb-tech.com/blog/python/ciscoconfparse.html>
  - VMWare <https://github.com/vmware/pyvmomi>
  - KVM/libvirt <https://libvirt.org/index.html>



**Americas Headquarters**  
Cisco Systems, Inc.  
San Jose, CA

**Asia Pacific Headquarters**  
Cisco Systems (USA) Pte. Ltd.  
Singapore

**Europe Headquarters**  
Cisco Systems International BV Amsterdam,  
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at [www.cisco.com/go/offices](http://www.cisco.com/go/offices).

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: [www.cisco.com/go/trademarks](http://www.cisco.com/go/trademarks). Third party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1110R)