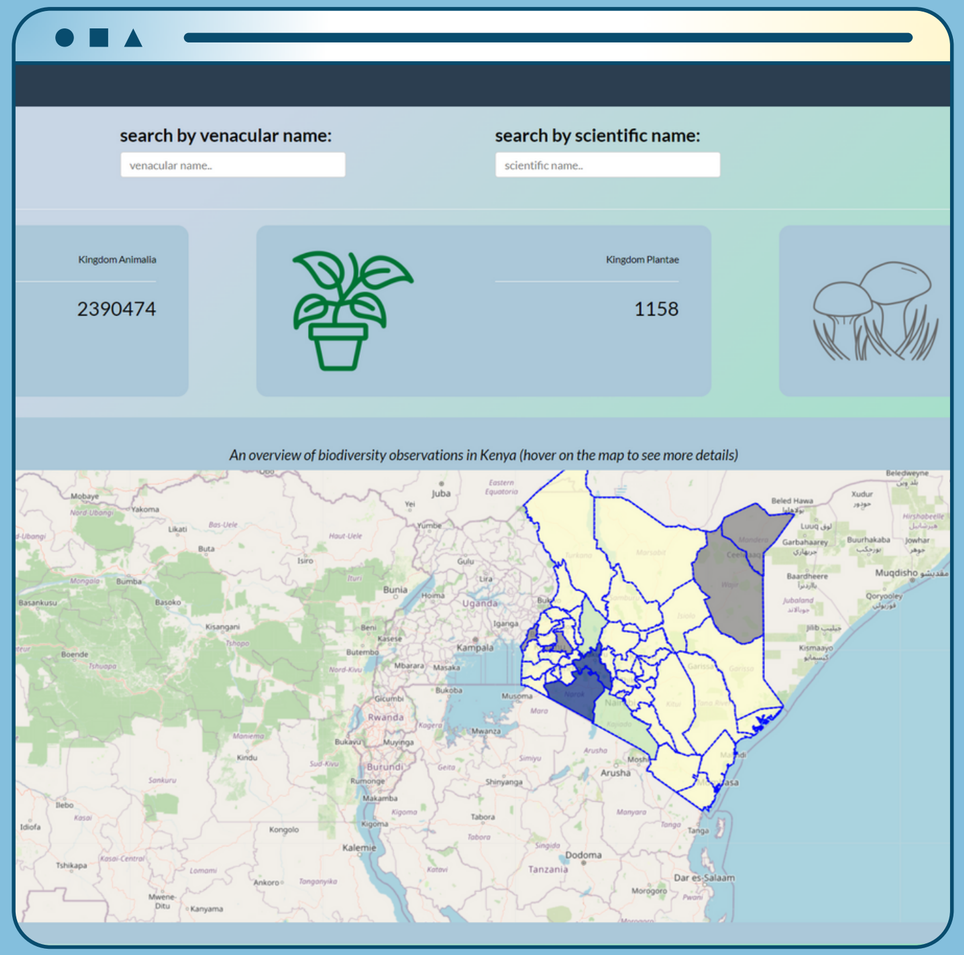
AUTOMATING UPDATES TO SHINY DASHBOARDS ON SHINYSERVER

Presented by: Clinton David

- Shiny is a framework in R for building web applications.
- The development can be done on your laptop using GUI tools such as RStudio or VScode
- Depending on your use case, it may be necessary to ‘expose’ your application to the web for remote access - we call this deployment



DEPLOYMENT

There are a number of ways for deploying your applications;

- shinyapps.io
- shinyproxy
- docker on cloud platforms
- Posit
- shinyserver

Some of the above solutions are open source while others are enterprise.
For this presentation we'll be looking at shinyserver (open source)

SHINYSERVER

- This is the open source backend program that you can use to expose your application to the internet.
- We can host multiple applications on shinyserver but each app will be assigned a web address such that whenever someone visits that address, the respective app will be spinned up.

```
 |- srv/
 |---- shiny-server/
 |------- index.html
 |------- sample-apps/
 |----------- hello
 |----------- rmd
```

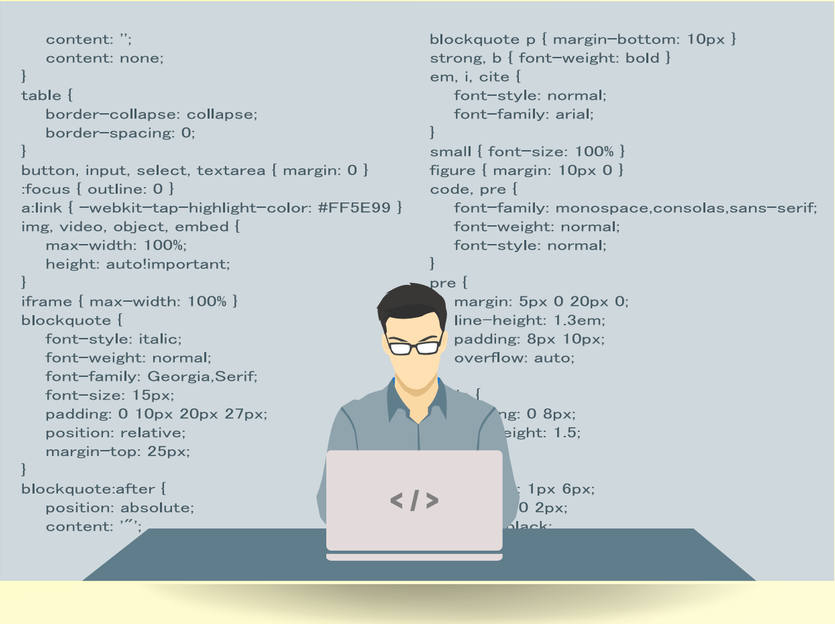
Adding app on shinyserver

shinyserver+git+github

The use of git and github gives you the priviledge of;

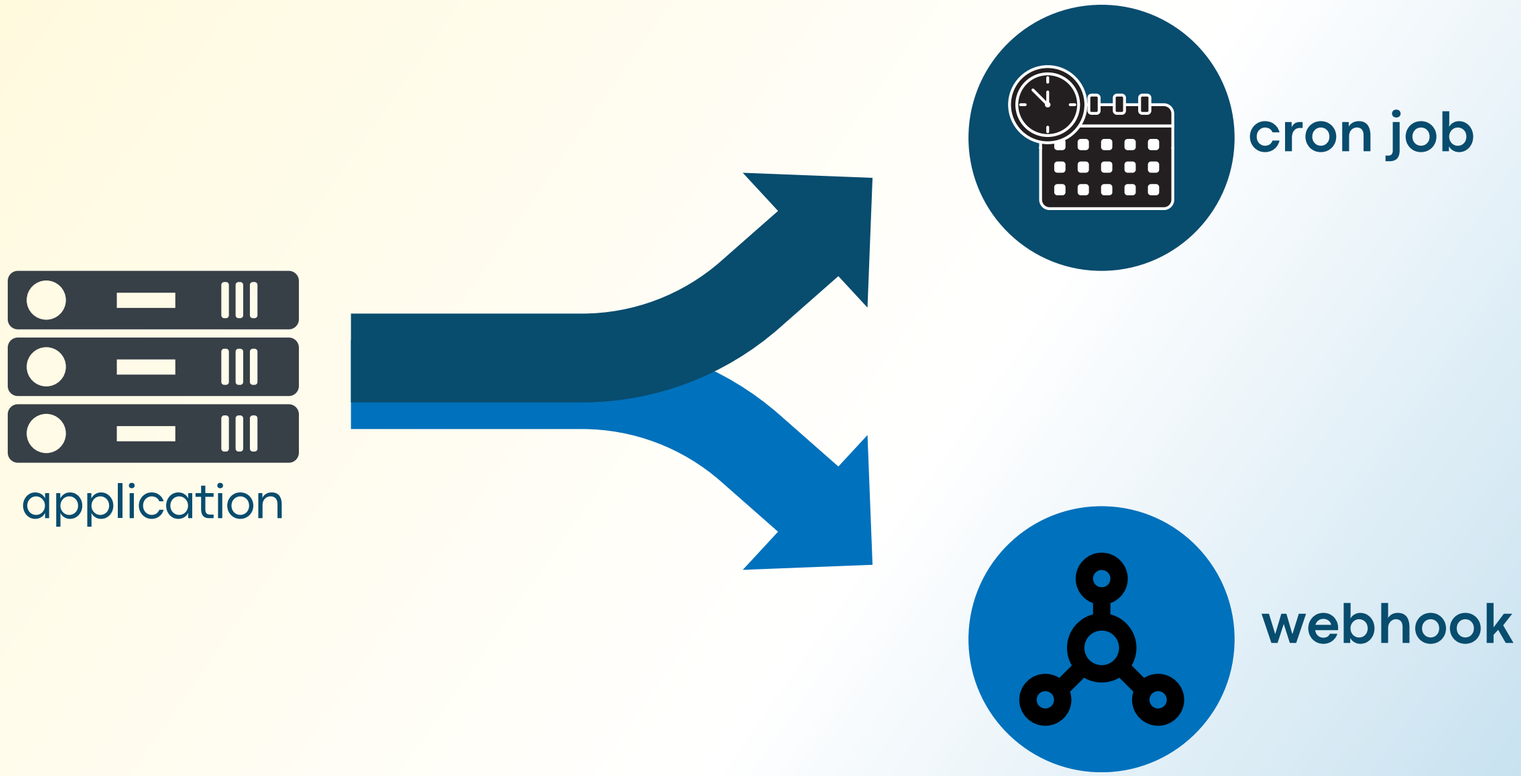
- Collaborating with others
- Code security
- The possibility of setting up an automated workflow

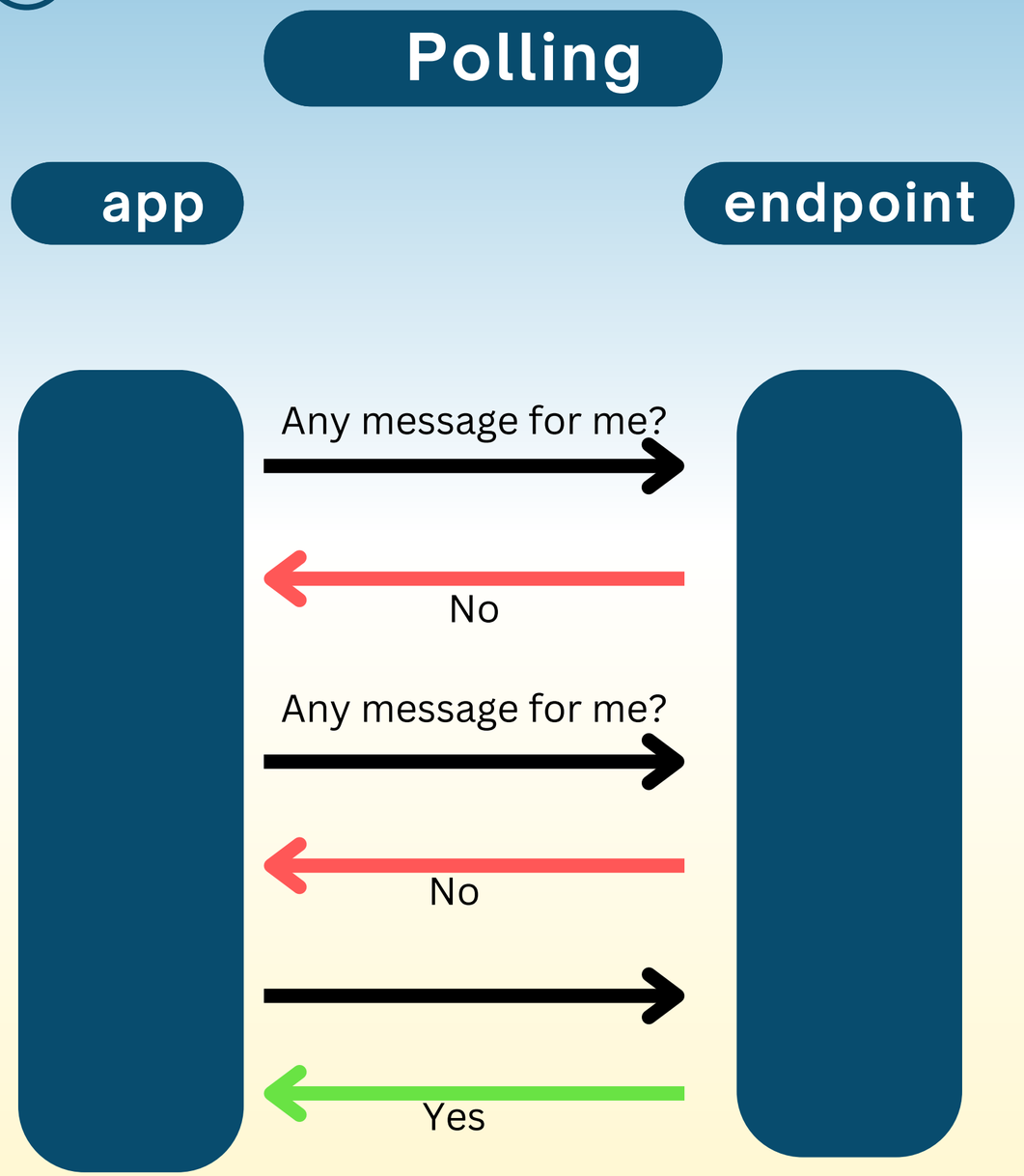
Hands-on..



- Create a github repo
- Use the version control option to create a project on Rstudio
- Create a shiny app
- Push to github

AUTOMATION




Polling
app
endpoint
Any message for me?
No
Any message for me?
No
Yes


webhook
app
endpoint
Let me know whenever there is a new message for me
You have a new message

github webhooks

Github webhooks provide a way for notifications to be delivered to an external web server whenever certain events occur on github.

For this to be set up you need;

- Endpoint to which the payload will be delivered
- The trigger/event to invoke the payload

Endpoint

This can be an API such as Django Rest API, flask or FastAPI. which you can configure on your deployment server

Once you have your Django API up and running on the server, its advisable to have a web server such as nginx to reverse proxy calls to the API, otherwise you can just use the server address and the port number on which the API is running - something like ***http://xxx.xx.xx.xx:8000***

THE DJANGO APP..

- Once you install you django rest api your folder structure should be something like..

```
|-myvenv
|---bin
|---lib/python3.10/site-packages
|---pyvenv.cfg
|-project_folder
|---_pycache_
|---_init_.py
|---asgi.py
|---settings.py
|---urls.py
|---wsgi.py
|-app_folder
|---_pycache_
|---migrations
|---_init_py
|---admin.py
|---apps.py
|---models.py
|---tests.py
|---urls.py
|---views.py
|-www
|-manage.py
```

settings.py of the project

```
ALLOWED_HOSTS = ['*']



# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'restapi_app',
]
```

urls.py of the project

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',include('restapi_app.urls'))
]
```

urls.py of the app

```
from django.urls import path
from . import views
from django.conf import settings

urlpatterns = [
    path('', views.gitpull)
]
```

views.py of the app

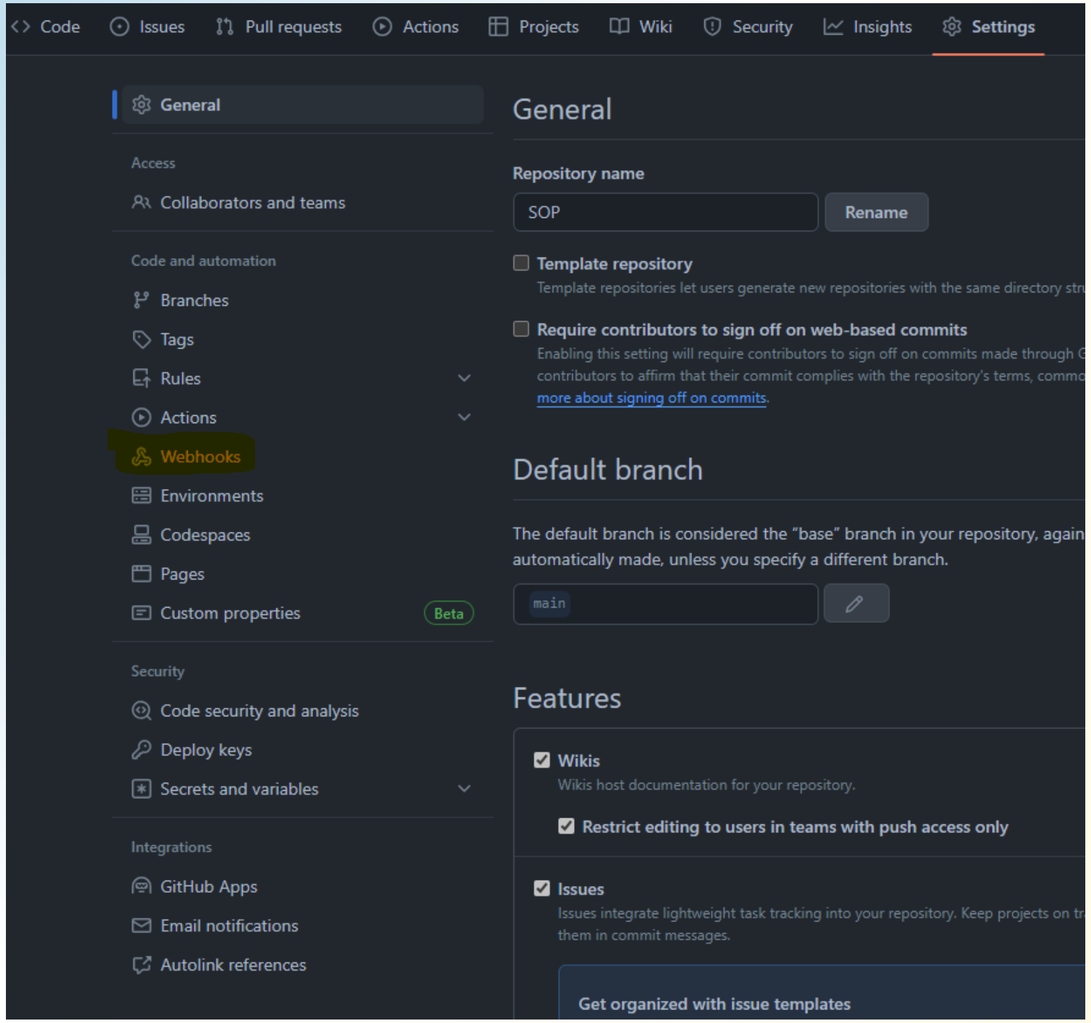
```python
from django.shortcuts import render
from rest_framework.response import Response
from rest_framework.decorators import api_view
import subprocess
from django.http import HttpResponse
from dotenv import load_dotenv
import os

load_dotenv()



# Create your views here.
@api_view(['POST'])
def gitpull(request):
    process = subprocess.run([os.getenv("SCRIPT_LOC")])
    if process.returncode == 0:
        print(process.returncode)
        return HttpResponse("success")
    else:
        return HttpResponse("fail")
```

The trigger

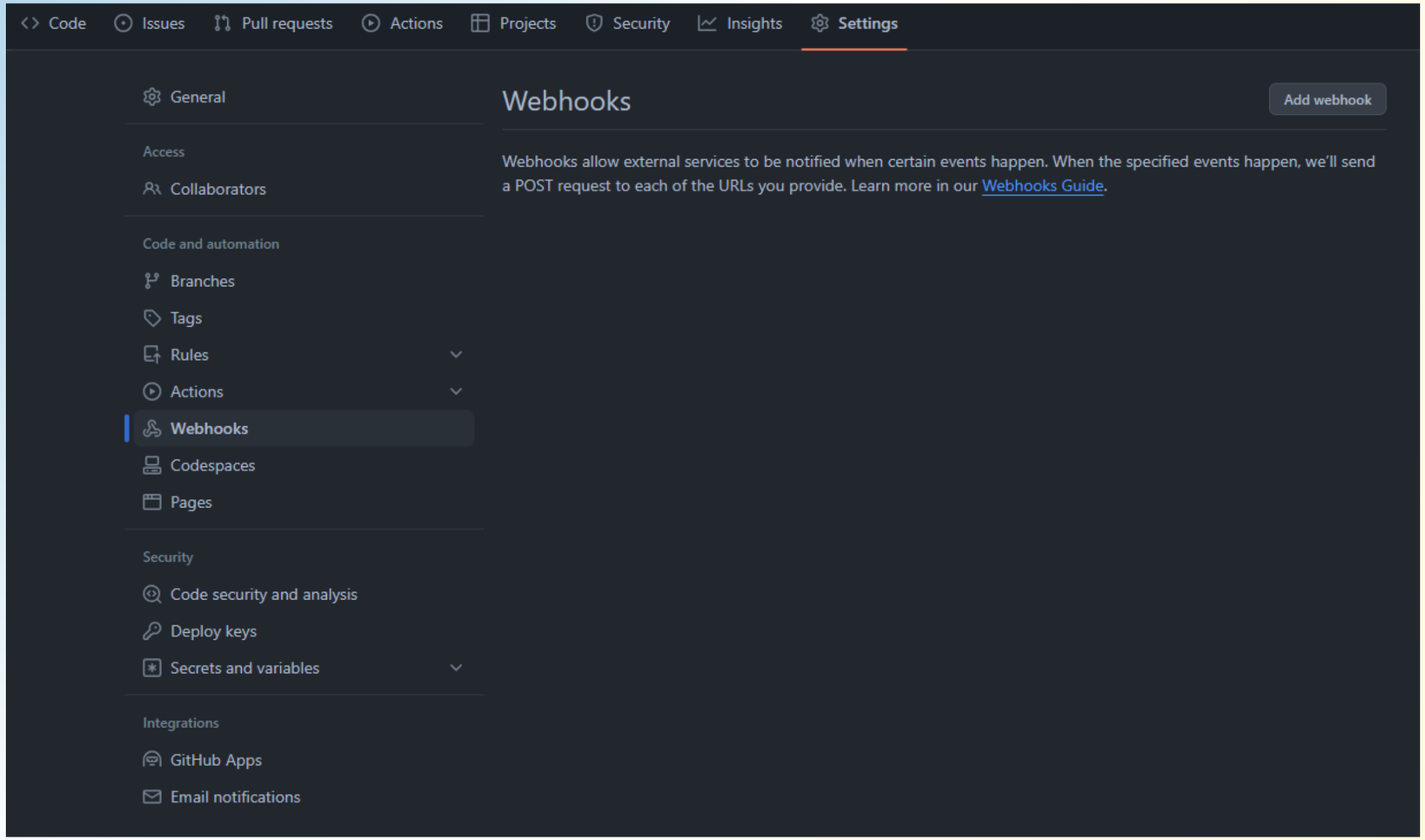
- When setting up the webhook, you need to specify what event will trigger the payload to be sent to the endpoint.

- For this case, a push event to the master branch sounds a good fit given that the repo could be having several branches expecially in a set up where there is collaboration.

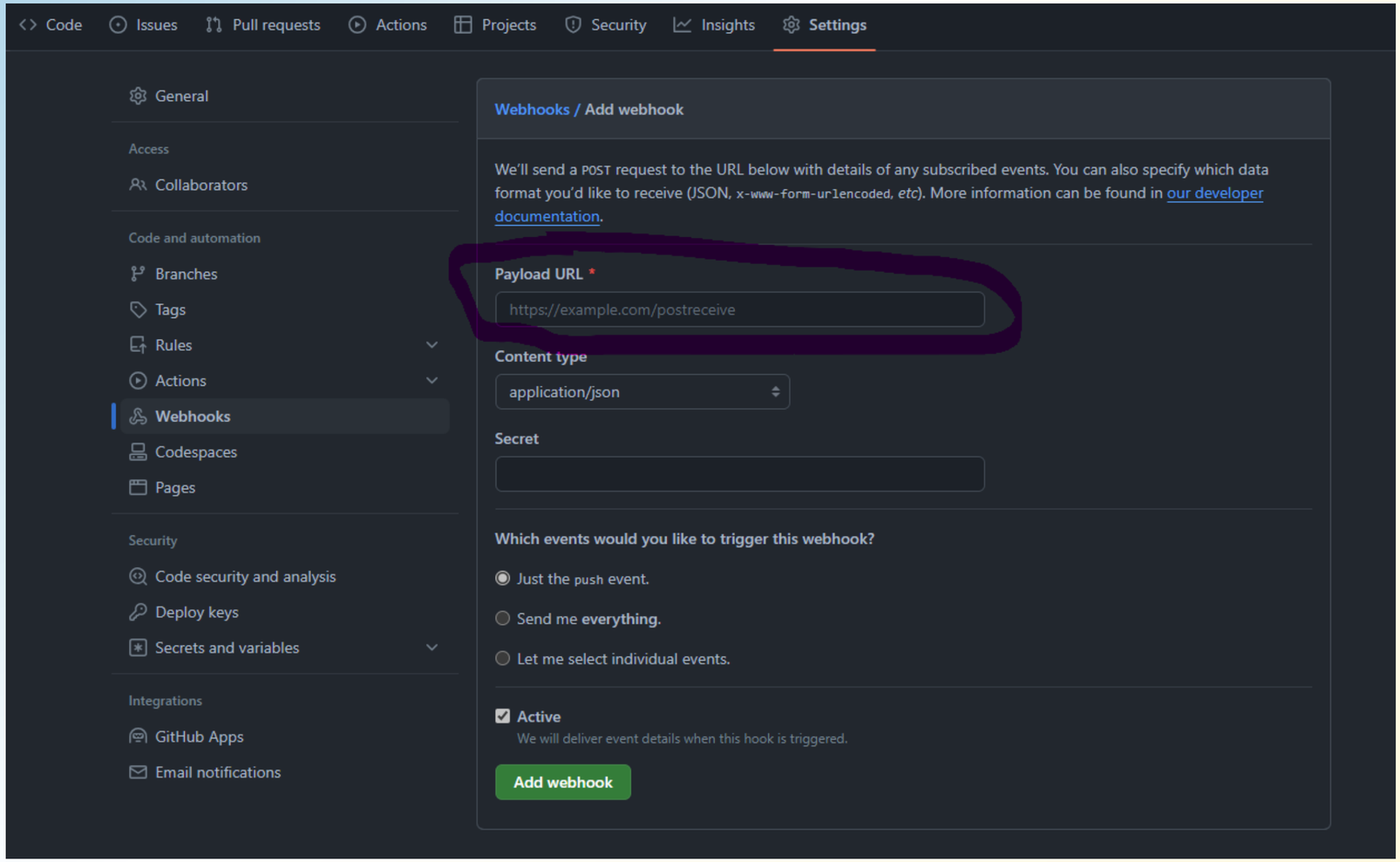
Setting up the webhook



add a webhook



register the webhook



bash

- Our intention was to automate a git pull to update the code on shinyserver .

- To achieve that we'll have a bash script that simply runs git pull from the repo when the payload is delivered from github

your bash script

```bash
#!/bin/bash
# Calculate the sum of two integers with pre initialize
LOG_LOCATION=/path/to/logfiles/folder/
exec > $LOG_LOCATION/bashscript_log_file.log 2>&1


################
# Uncomment if you want the script to always use the scripts
# directory as the folder to look through
#REPOSITORIES="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"
REPOSITORIES=`pwd`

IFS=$'\n'

for REPO in `ls "$REPOSITORIES/"`
do
  if [ -d "$REPOSITORIES/$REPO" ]
  then
    echo "Updating $REPOSITORIES/$REPO at `date`"
    if [ -d "$REPOSITORIES/$REPO/.git" ]
    then
      cd "$REPOSITORIES/$REPO"
      git status
      echo "Pulling"
      git pull
    else
      echo "Skipping because it doesn't look like it has a .git folder."
    fi
    echo "Done at `date`"
    echo
  fi
done
```

Remember to make the bash script executable

Ultimate system architecture