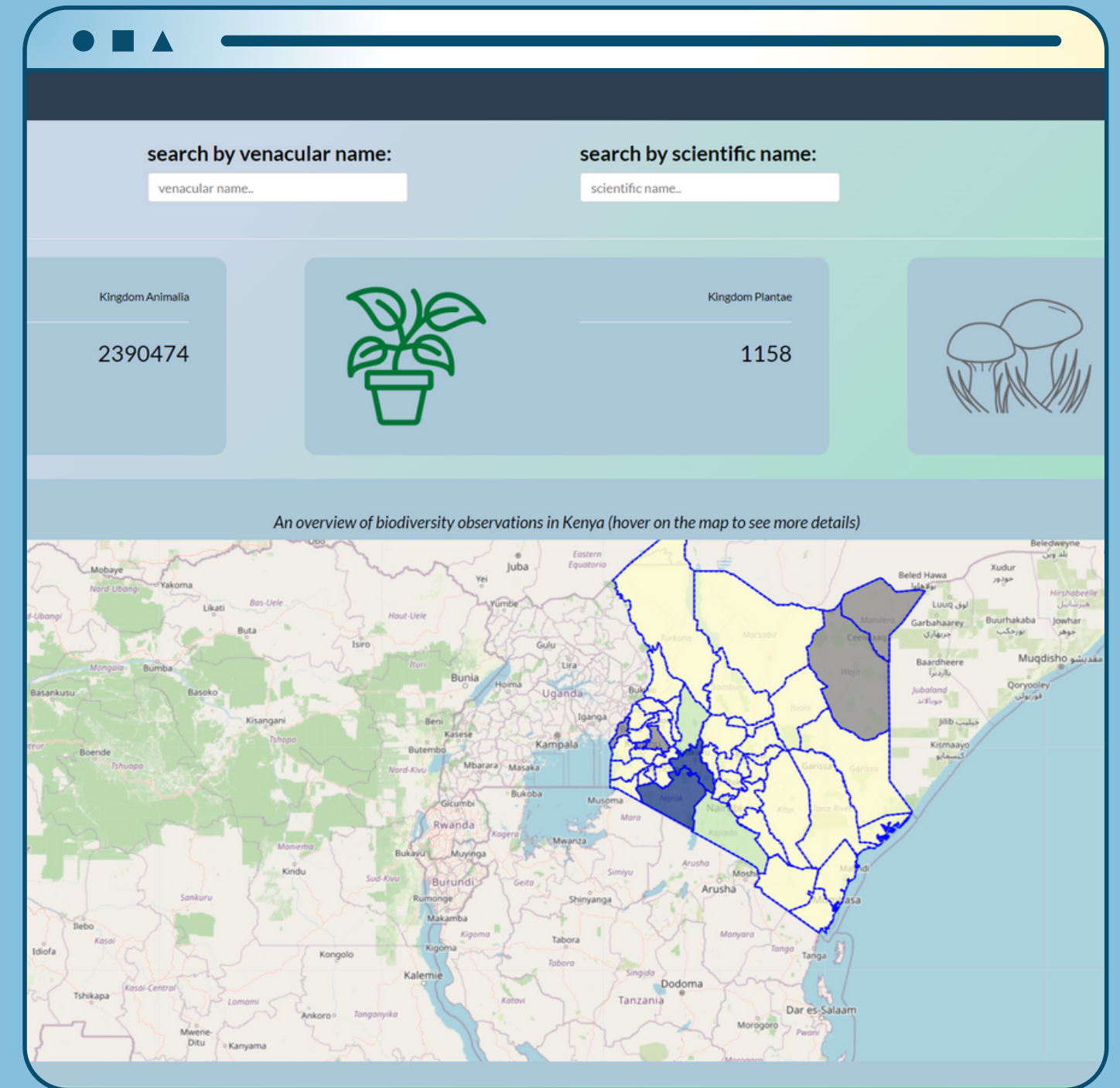


AUTOMATING UPDATES TO SHINY DASHBOARDS ON SHINYSERVER

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





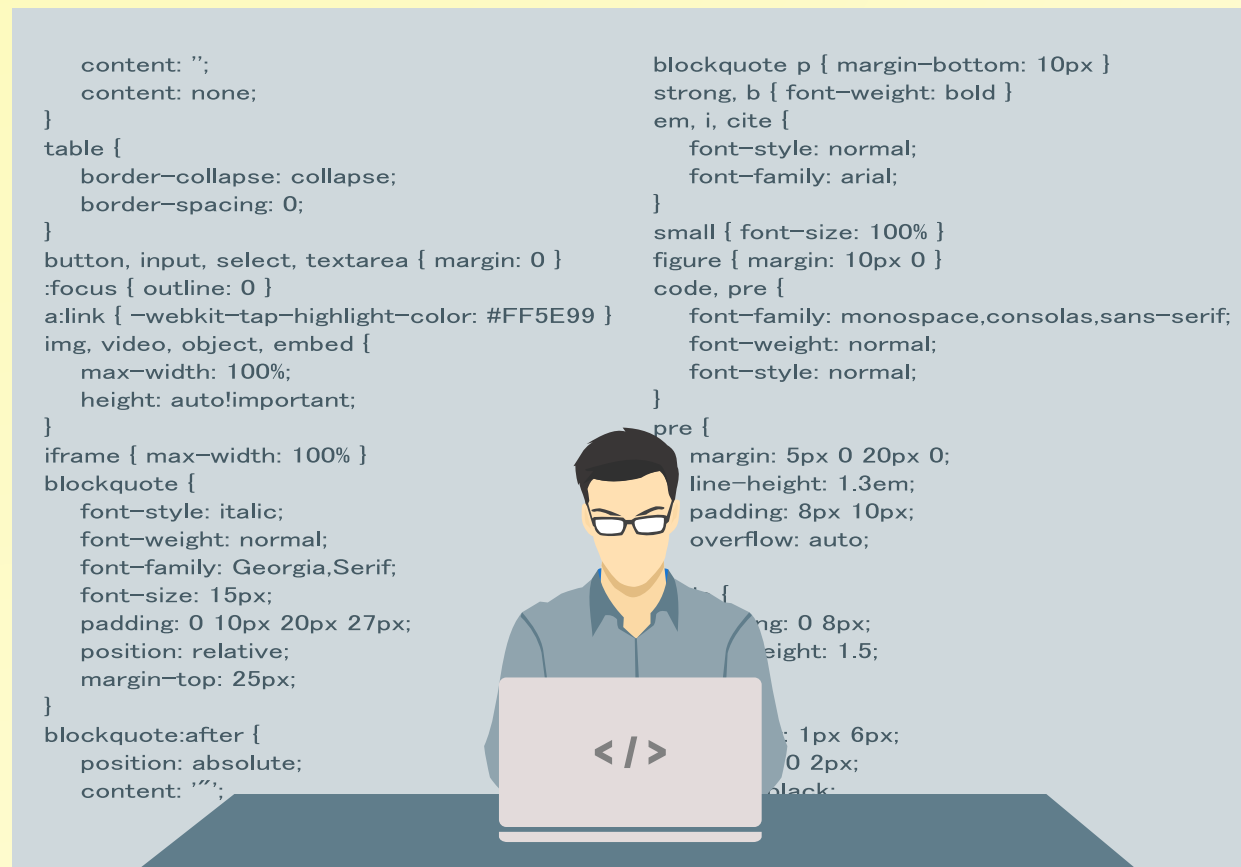
shinyserver+git+github

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

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



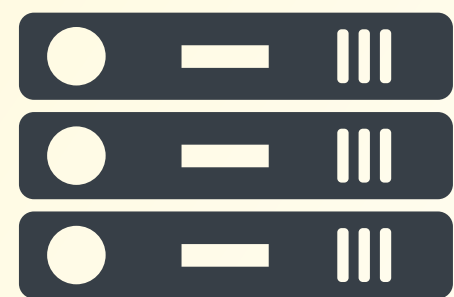
Hands-on..



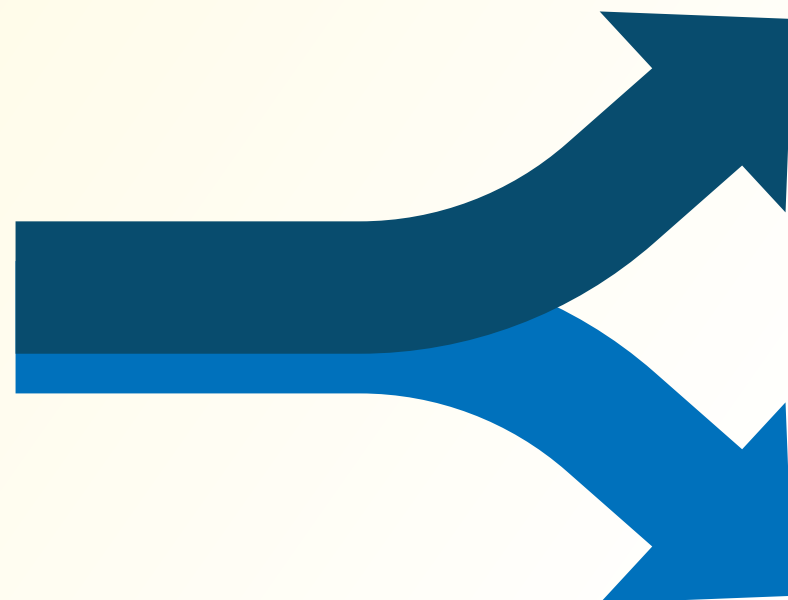
- ✓ Create a github repo
- ✓ Use the version control option to create a project on Rstudio
- ✓ Create a shiny app
- ✓ Push to github



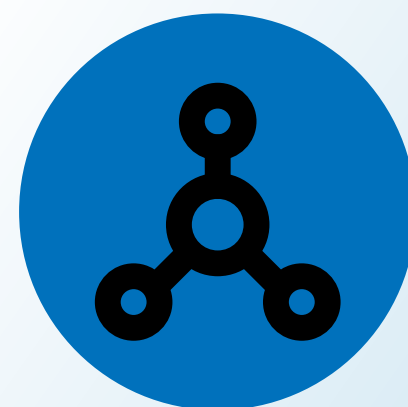
AUTOMATION



application



cron job



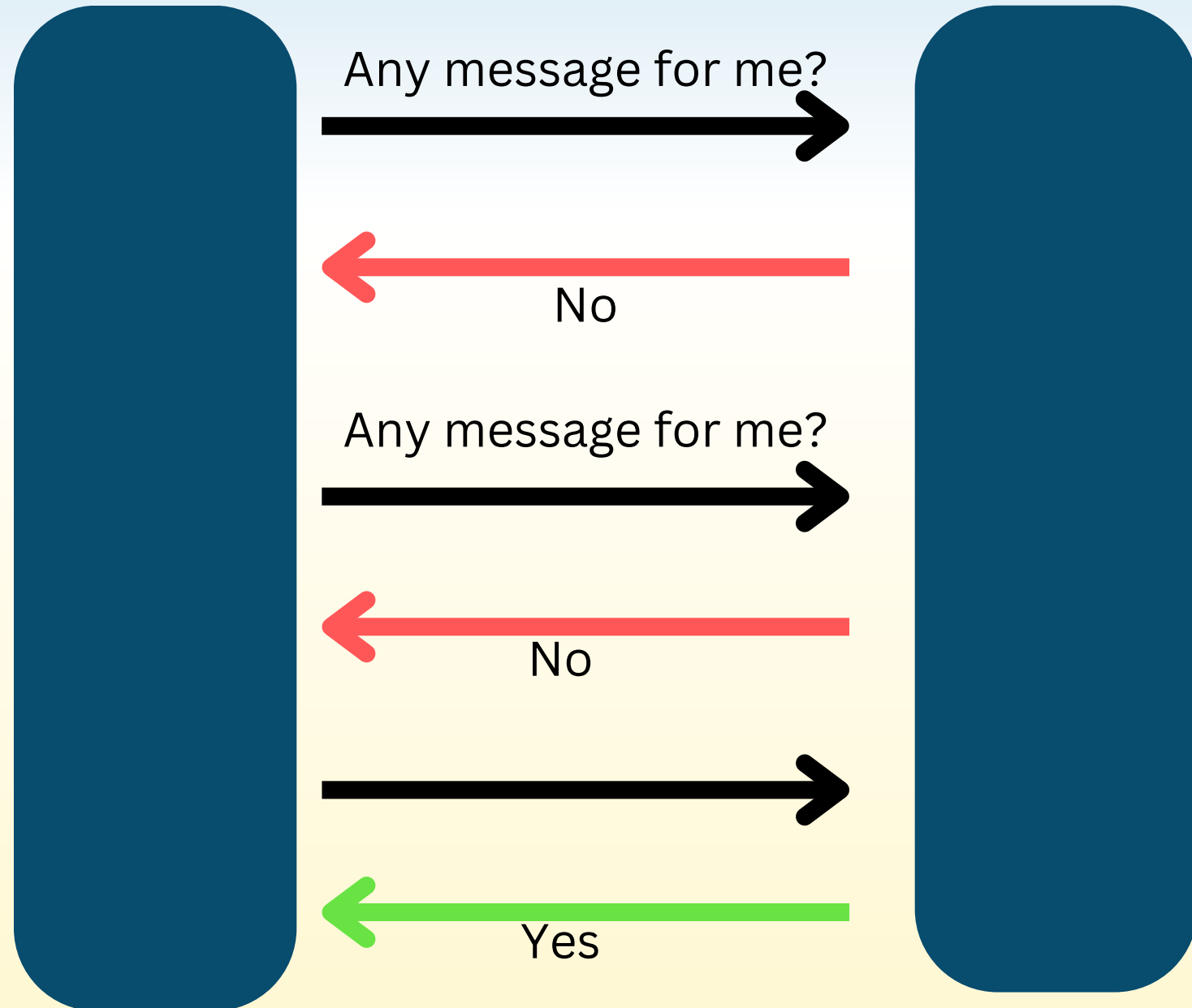
webhook



Polling

app

endpoint



webhook

app

endpoint





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

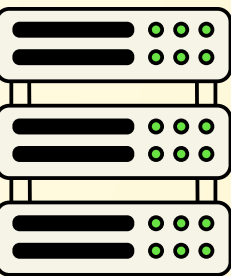
Once you have your Django API up and running on the server, its advisable to have a web server such as nginx to reverse proxy calls to the API, otherwise you can just use the server address and the port number on which the API is running - something like ***http://xxx.xx.xx.xx:8000***

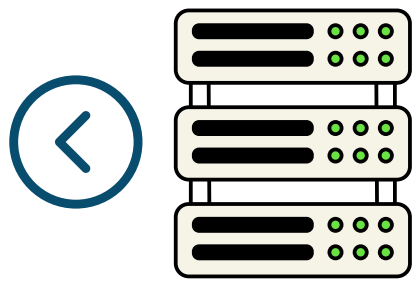


THE DJANGO APP..

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

```
...
|-myenv
|---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
...
```



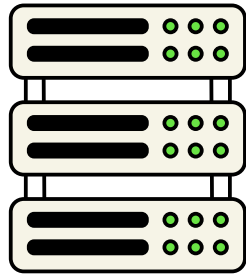


settings.py of the project

```
ALLOWED_HOSTS = ['*']

# Application definition

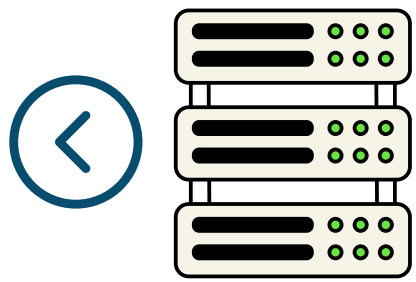
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'restapi_app',
]
```

urls.py of the project

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

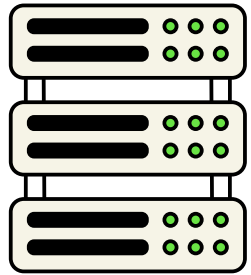
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('restapi_app.urls'))
]
```



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

```
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 especially in a set up where there is collaboration.



Setting up the webhook

The screenshot displays the GitHub repository settings interface. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The left sidebar lists various settings categories: General, Access, Collaborators and teams, Code and automation, Branches, Tags, Rules, Actions, Webhooks (highlighted with a yellow background), Environments, Codespaces, Pages, Custom properties (marked as Beta), Security, Code security and analysis, Deploy keys, Secrets and variables, Integrations, GitHub Apps, Email notifications, and Autolink references. The main content area is titled 'General' and shows the 'Repository name' as 'SOP' with a 'Rename' button. Below this are checkboxes for 'Template repository' and 'Require contributors to sign off on web-based commits'. The 'Default branch' section shows 'main' as the default. The 'Features' section has checkboxes for 'Wikis', 'Restrict editing to users in teams with push access only', and 'Issues'. A button at the bottom right says 'Get organized with issue templates'.



add a webhook

<> Code

Issues

Pull requests

Actions

Projects

Security

Insights

Settings

General

Access

Collaborators

Code and automation

Branches

Tags

Rules

Actions

Webhooks

Codespaces

Pages

Security

Code security and analysis

Deploy keys

Secrets and variables

Integrations

GitHub Apps

Email notifications

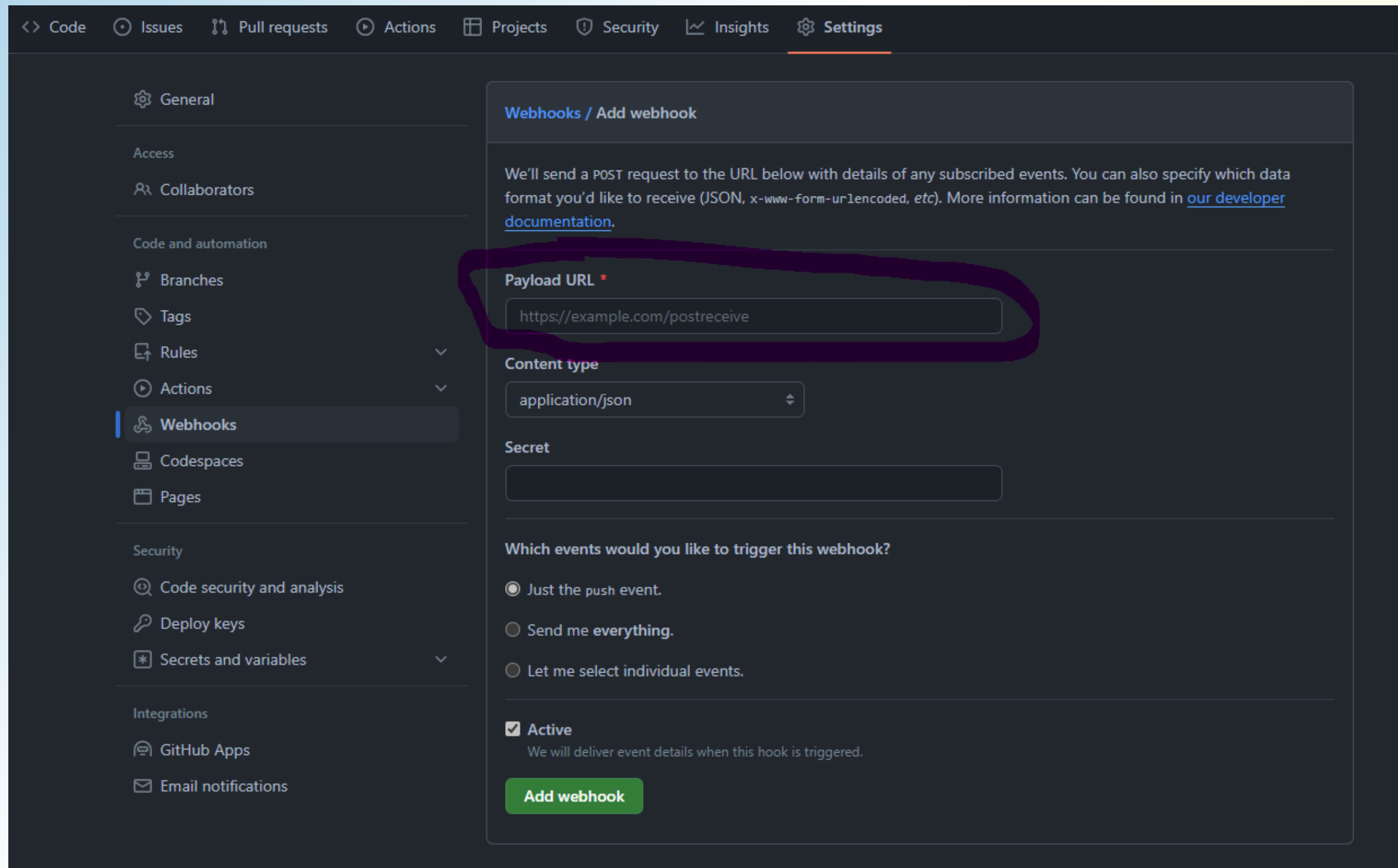
Webhooks

Add webhook

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our [Webhooks Guide](#).



register the webhook



The screenshot shows the GitHub 'Settings' page for a repository, specifically the 'Webhooks' section. The left sidebar contains various settings categories: General, Access, Collaborators, Code and automation (with sub-items: Branches, Tags, Rules, Actions, and Webhooks), Codespaces, Pages, Security (with sub-items: Code security and analysis, Deploy keys, and Secrets and variables), and Integrations (with sub-items: GitHub Apps and Email notifications). The 'Webhooks' section is currently selected. The main content area is titled 'Webhooks / Add webhook'. It includes a descriptive paragraph about sending POST requests. The 'Payload URL' field is highlighted with a red circle and contains the text 'https://example.com/postreceive'. Below this, the 'Content type' is set to 'application/json'. There is an empty 'Secret' field. Under the heading 'Which events would you like to trigger this webhook?', three radio button options are listed: 'Just the push event.' (selected), 'Send me everything.', and 'Let me select individual events.'. At the bottom, the 'Active' checkbox is checked, with a note 'We will deliver event details when this hook is triggered.'. A green 'Add webhook' button is at the bottom right.

<> Code Issues Pull requests Actions Projects Security Insights Settings

General

Access

Collaborators

Code and automation

Branches

Tags

Rules

Actions

Webhooks

Codespaces

Pages

Security

Code security and analysis

Deploy keys

Secrets and variables

Integrations

GitHub Apps

Email notifications

Webhooks / Add webhook

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in [our developer documentation](#).

Payload URL *

https://example.com/postreceive

Content type

application/json

Secret

Which events would you like to trigger this webhook?

☒ Just the push event.

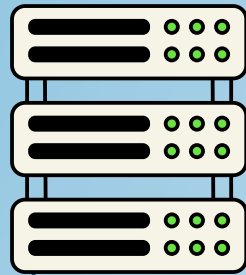
☐ Send me everything.

☐ Let me select individual events.

☒ **Active**

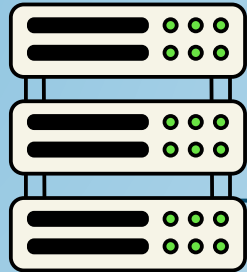
We will deliver event details when this hook is triggered.

Add 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



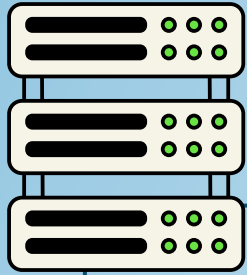
your bash script

```
#!/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

A stylized illustration of a laptop computer with a black screen and a silver base. The screen displays the command `sudo chmod +x script.sh` in red text.

```
sudo chmod +x script.sh
```


Ultimate system architecture

