DevOps Workshop

Part 2 coming soon!

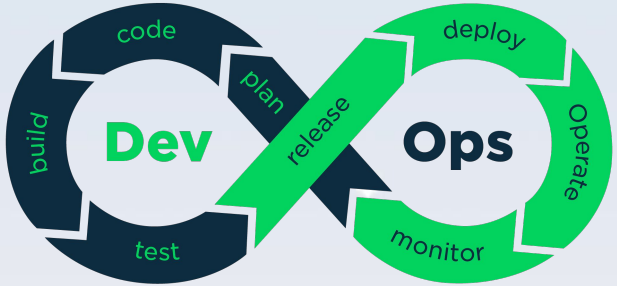
Today’s Agenda

- What is **DevOps**?
- What is CI/CD?
- How to implement CI/CD using **GitHub Actions**?
- A brief introduction to Firebase
- And much more!

By the end of it, you'll have your **own "*DevOps-ified* " *React*/ Firebase project!**

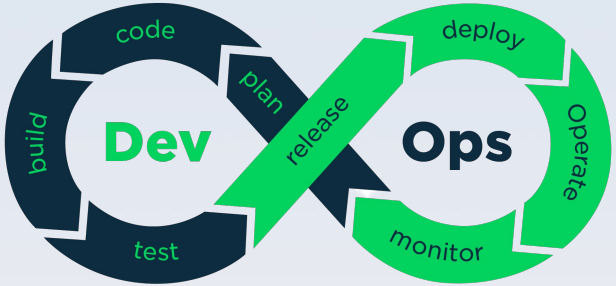
You can then edit the React App's code and turn it into your own thing (ex. Personal website, etc.)

So, what is DevOps?



- Set of **cultural philosophies, practices, and tools** that increases an organization's ability to **deliver applications and services at high velocity**
- Primarily through **automating** and **streamlining** development and infrastructure management processes

Benefits of DevOps



- Speed/Agility
- Rapid Deployment
- Quality and reliability
- Improved Collaboration
- And much more!

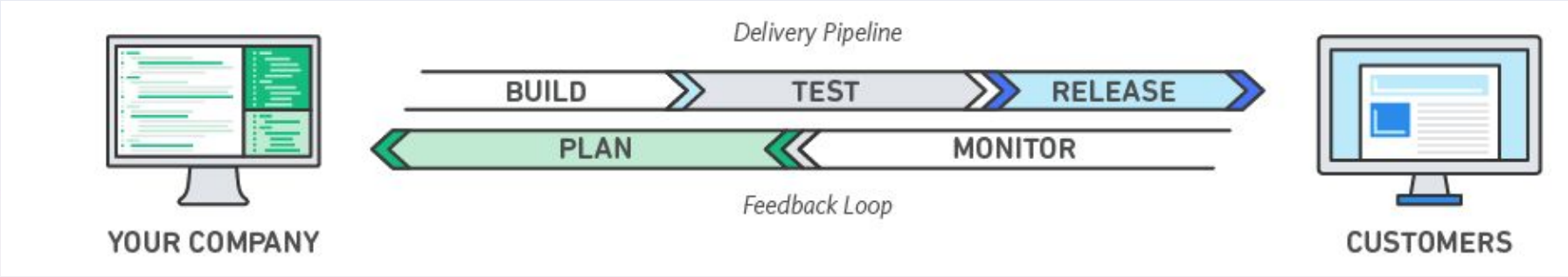
DevOps Best Practices

- Continuous Integration
- Continuous Delivery
- Microservices
- Infrastructure as Code
- Monitoring and Logging
- Communication and Collaboration

DevOps Best Practices

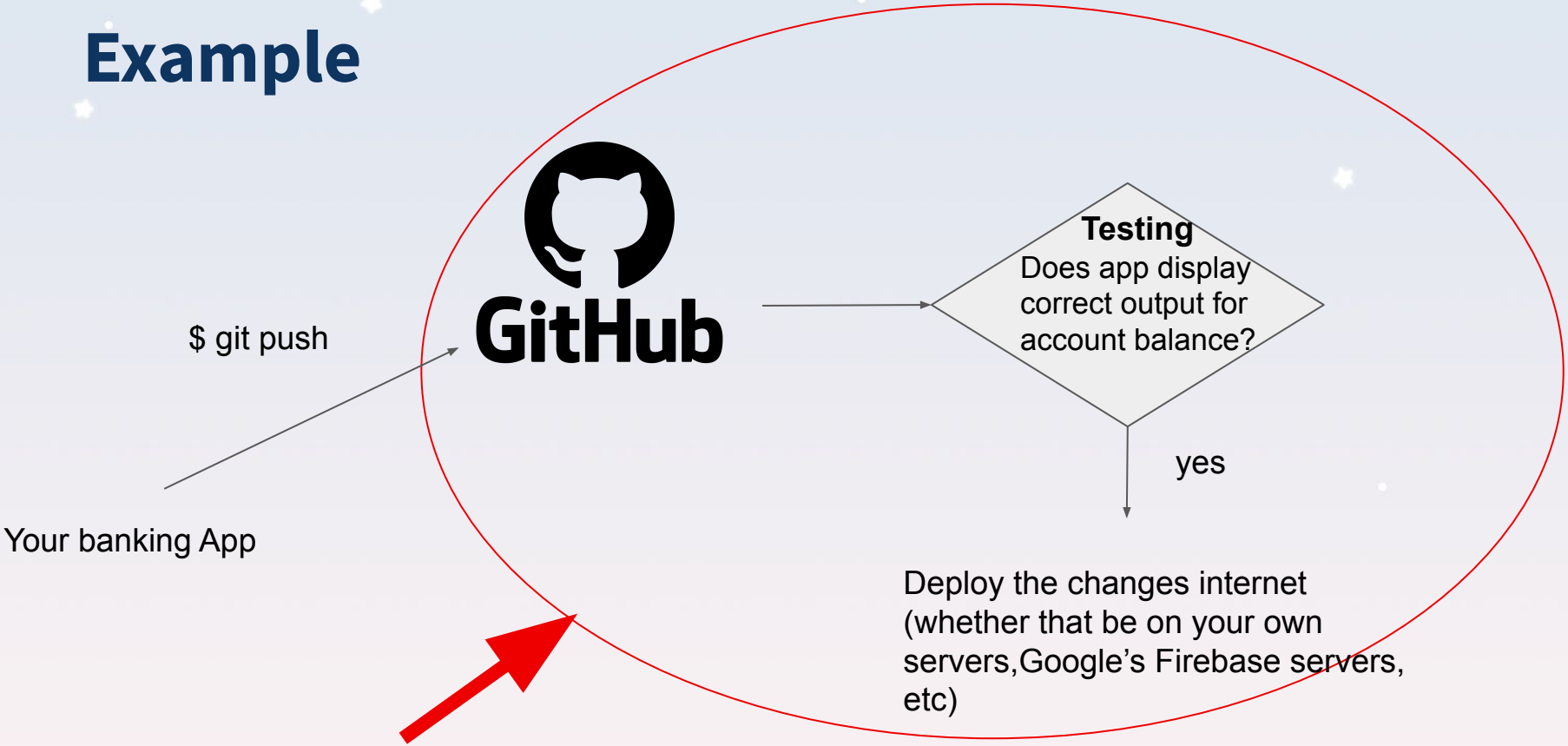
- **Continuous Integration**
- **Continuous Delivery**
- Microservices
- Infrastructure as Code
- Monitoring and Logging
- Communication and Collaboration

Note: Continuous Integration + Continuous Delivery referred to as “CI/CD”



- Pipeline: set of automated processes
  - Ex. Automatically testing your code, and then automatically deploying them

Example


$ git push
GitHub
Testing
Does app display correct output for account balance?
yes
Your banking App
Deploy the changes internet (whether that be on your own servers,Google's Firebase servers, etc)
How do we automate this? Aka how do we implement CI/CD for our app?

How do we automate this?

- We call this process of automating particular parts of software development “**continuous integration, continuous delivery, and continuous deployment (CI/CD for short)**”

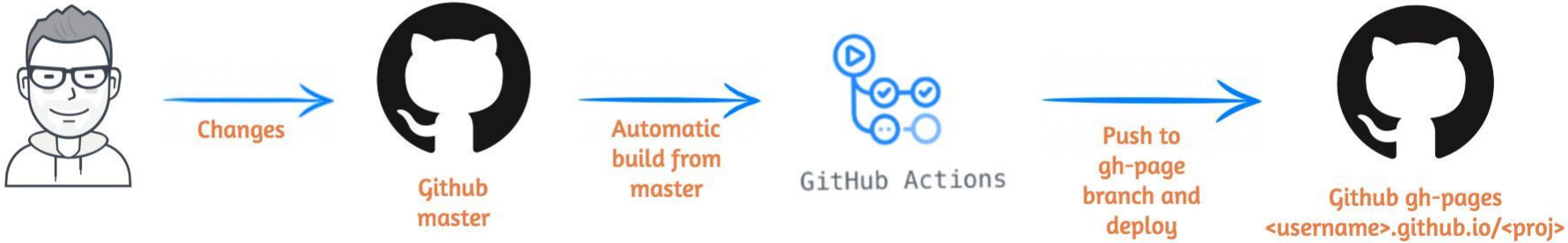
How do we automate this?

- Software/tools that allow us to implement CI/CD are:
  - **GitHub Actions**
  - Travis CI
  - Jenkins
  - etc.

GitHub Actions

- **Service created by GitHub** that we can use for CI/CD
- GitHub Actions makes it easy to **automate all your software workflows**
- You can configure a GitHub Actions workflow to be triggered when something happens in your repository
    - **Ex.** on a push to the repo, on a merge between two branches, on every pull request etc.

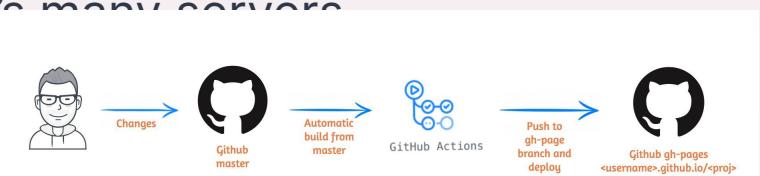
How Github Actions work



3. Specify what you want the script to do when it is triggered
4. When this trigger is set off, GitHub will run this script on their servers!

How Github Actions work (lower level)

1. We create a .github/workflows/<something>.yaml in the root level of our github repository
2. We specify when we want this script to run (ex. On a push to the **main** branch)
3. We can optionally define in some environment variables that will be available to use by the server running the script
4. We specify what the script should do
5. This script then gets run on one of GitHub's many servers



Example

```yaml
name: the name of this 'action' (whatever you want it to be)

on:
  push:
    branches: [main]
  workflow_dispatch: # allows this to be triggered manually

jobs:
  rebuild-ui:
    runs-on: ubuntu-latest
    steps:
      - name: the name of this 'step' (whatever you want it to be)
        env: # defining environment variables
          _HOST: ${{ secrets.HOST }}
          _USERNAME: ${{ secrets.USERNAME }}
          _PRIVATE_KEY: ${{ secrets.KEY }}
        run: |
          echo 'I should do something here'
          echo '...'
          echo 'success!'
```

myProject/.github/workflows/my-action.yaml

```
name: the name of this 'action' (whatever you want it to be)
```

```yaml
name: the name of this 'action' (whatever you want it to be)

on:
  push:
    branches: [main]
  workflow_dispatch: # allows this to be triggered manually
```

```yaml
name: the name of this 'action' (whatever you want it to be)

on:
  push:
    branches: [main]
  workflow_dispatch: # allows this to be triggered manually

jobs:
  rebuild-ui:
```

You can name this job whatever you want

```yaml
name: the name of this 'action' (whatever you want it to be)

on:
  push:
    branches: [main]
  workflow_dispatch: # allows this to be triggered manually

jobs:
  rebuild-ui:
    runs-on: ubuntu-latest
    steps:
```

You can name this job whatever you want

```yaml
name: the name of this 'action' (whatever you want it to be)

on:
  push:
    branches: [main]
  workflow_dispatch: # allows this to be triggered manually

jobs:
  rebuild-ui:
    runs-on: ubuntu-latest
    steps:
      - name: the name of this 'step' (whatever you want it to be)
        env: # defining environment variables
          _HOST: ${{ secrets.HOST }}
          _USERNAME: ${{ secrets.USERNAME }}
          _PRIVATE_KEY: ${{ secrets.KEY }}
```

Define these ‘secrets’ in your GitHub repos Settings -> Secrets

Secrets are basically **encrypted variables that you can define for your github actions repo** so you **don’t have to hardcode them in to your (possibly) publicly viewable script**

```yaml
name: the name of this 'action' (whatever you want it to be)

on:
  push:
    branches: [main]
  workflow_dispatch: # allows this to be triggered manually

jobs:
  rebuild-ui:
    runs-on: ubuntu-latest
    steps:
      - name: the name of this 'step' (whatever you want it to be)
        env: # defining environment variables
          _HOST: ${{ secrets.HOST }}
          _USERNAME: ${{ secrets.USERNAME }}
          _PRIVATE_KEY: ${{ secrets.KEY }}
```

Define these ‘secrets’ in your GitHub repos Settings -> Secrets
See here for details

Secrets are basically **encrypted variables that you can define for your github actions repo** so you **don’t have to hardcode them in to your (possibly) publicly viewable script**

```yaml
name: the name of this 'action' (whatever you want it to be)

on:
  push:
    branches: [main]
  workflow_dispatch: # allows this to be triggered manually

jobs:
  rebuild-ui:
    runs-on: ubuntu-latest
    steps:
      - name: the name of this 'step' (whatever you want it to be)
        env: # defining environment variables
          _HOST: ${{ secrets.HOST }}
          _USERNAME: ${{ secrets.USERNAME }}
          _PRIVATE_KEY: ${{ secrets.KEY }}
        run: |
          echo 'I should do something here'
          echo '...'
          echo 'success!'
```

Define these ‘secrets’ in your GitHub repos Settings -> Secrets

```
run: |
  echo 'I should do something here'
  echo '...'
  echo 'success!'
```

- This is *admittedly a pretty useless* example of a GitHub action (just writing some values to stdout)
- Later in the workshop we will see how we can ‘**build’ a React app** , **test it,** and **deploy it** to Firebase.

Testing

- One action that is often included in CI/CD is **automated testing**
- **Testing is important important**
  - Frequent testing of your codebase allows your program to be less prone to errors
  - This allows checking to make sure a new block of code **doesn't break** the previously written code
  - etc

Manual Testing

- This is done by a person (tester) **without** using any automated tools
- This is often very expensive and time consuming for a company
- Prone to human errors
- Any new application must be manually tested before its testing can be automated.

Automated Testing

- Done by a computer
- Extremely inexpensive and quick to run
  - Development of good tests suites can take time though!
- Might not be able to test certain things as well as a person though
  - Ex. If your website has a lot of animations and want to test to see if all of them look ok, you're likely better off with manual testing

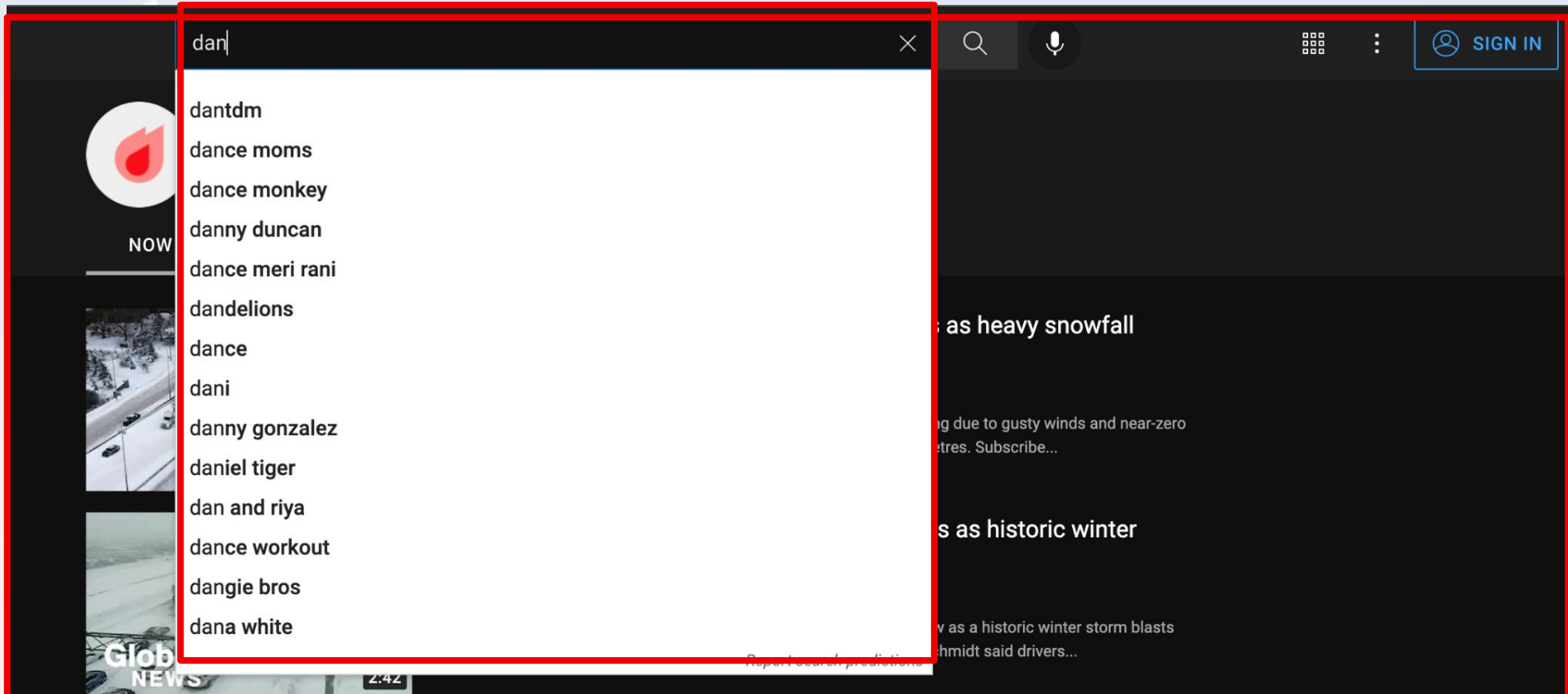
Automated Testing: Types of Tests

- **Unit Tests**
  - Testing small '**units**' of an application individually
  - For example, testing individual methods, functions, components or modules
- **Integration tests**
  - Testing to see if your modules/components/functions/etc work together
- Several other types of tests can be found here

Examples of each

- both doors likely work individually (would be verified by a unit test)
- but they clearly **do not work together** (would be verified by a integration test)

Unit vs Integration Example



Let’s see how tests look like in React!

Moving onto Firebase

Wait, what's Firebase?

- Firebase is a **platform** developed by Google for creating mobile and web applications.
- It abstracts away a lot of backend development for you
  - ex. you can get a website up and running with a single click without having to do any back-end work
  - simplifies setting up and connecting your app to a database (called Firestore)
  - Simplifies Authentication
  - And much more!

Firebase intro continued..

- We will just be focusing on the “web hosting” portion of Firebase for this workshop
- Please let us know if you’d like to see a dedicated workshop on Firebase!
- **Let’s create a new firebase project!**

Deploying React app on Firebase (~3 min)

1. **FORK** this repository (aka created your own copy of it) and **clone** this **forked** repo onto your computer.
2. **cd <name of this directory>**
3. **npm install firebase-tools -g**
4. **firebase login**
   a. Logs you in with your google account using the CLI
5. **firebase init**
   a. Will prompt you with several questions (I'll show you what to select in next slide)

Firebase-ify an existing React project

Answer the **firebase init** prompts as follows:

- Which Firebase CLI features do you want to set up for this folder?

  → **Configure and deploy Firebase Hosting sites**
- What project do you want to use? → **Use existing**
  - And then select the project you just created
- What to use as public directory? → **build**
- Configure as single page app? → **Yes**
- Set up automatic builds and deploys with GitHub? → **No**
- Overwrite index.html? → **No**

Firebase-ify an existing React project

```
? Which Firebase CLI features do you want to set up for this folder? Press Space to select features, then Enter to
 confirm your choices. Hosting: Configure and deploy Firebase Hosting sites

=== Project Setup

First, let's associate this project directory with a Firebase project.
You can create multiple project aliases by running firebase use --add,
but for now we'll just set up a default project.

? Please select an option: Use an existing project
? Select a default Firebase project for this directory: devops-workshop-d0a00 (devops-workshop)
i  Using project devops-workshop-d0a00 (devops-workshop)

=== Hosting Setup

Your public directory is the folder (relative to your project directory) that
will contain Hosting assets to be uploaded with firebase deploy. If you
have a build process for your assets, use your build's output directory.

? What do you want to use as your public directory? public
? Configure as a single-page app (rewrite all urls to /index.html)? Yes
? File public/index.html already exists. Overwrite? No
```

build

Firebase init result

- You should see that firebase created some new files and directories
- ***firebase json:***
  - where you can specify various hosting rules for your app (ex. where should firebase find all the files it will server users?)
- ***.firebaserc:***
  - - a file that contains info to identify your firebase project

Firebase init result

- To serve your project locally (through a web server running locally that listens to port 5000 by default)
- **npm install && npm run build** (if you haven't already)
- Followed by, **firebase serve**
- ***Note: only for windows users****, please remove the "CI=false &&" part from line 17 of package.json if you want to run it locally.* ***IMPORTANT:*** *add it back once you are ready to setup github actions for it.*
- ***Note:*** *if you get an 403 error when visiting localhost:5000,*

  Do this instead: ***firebase serve -o 0.0.0.0***

Let’s deploy to firebase

➔ Just execute the command (assuming you have ran **npm run build** beforehand):

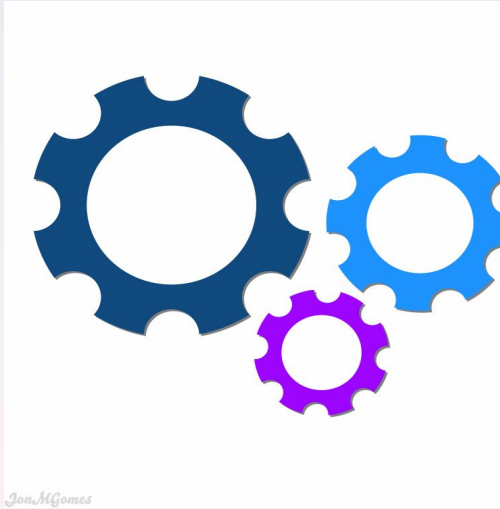
→ firebase deploy

- URL of your app will be outputted to you
- App is now live on the internet!
- ***Really easy, right?***

Let’s automate this with GitHub Actions!

Let’s head over to my computer!

For those revisiting the slides afterwards, the finished GitHub actions file is here



Let’s deploy to firebase (~1min)

- First thing **we need is a “Firebase CI Token”** to basically a**uthorize GitHub Actions** to be able to interact with our Firebase Project
- Run the following command in your terminal:
  - **firebase login:ci**

This will prompt you to sign in using your browser and if successful it will output a token in your terminal

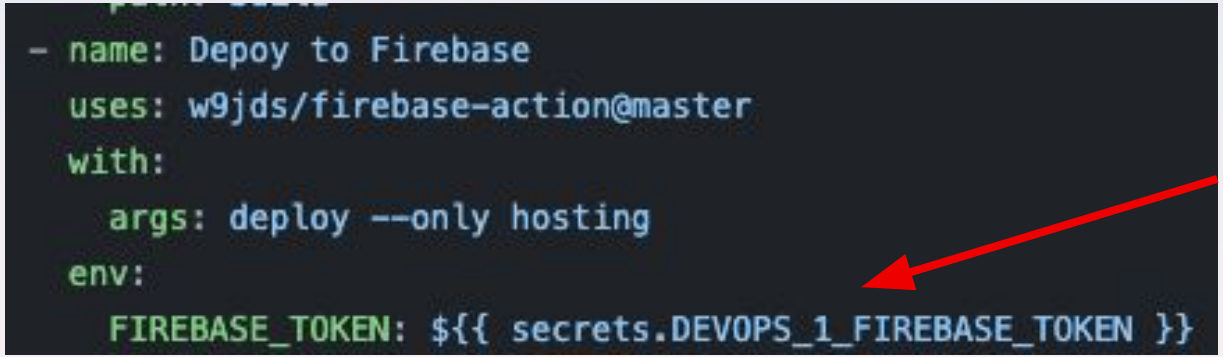
Deploying to firebase cntd. (~2min)

- Follow these steps to add this token as a ‘secret’ in your repository
  - Just think of a ‘secret’ as a safe, secure variable that your GitHub actions script can use
- To follow along with me (or to get the finished GitHub Action file working), **name this secret DEVOPS_1_FIREBASE_TOKEN**

(I’ll explain why later)

Deploying to firebase ctnd

- Here is an example of how you can access it in your GitHub Actions script

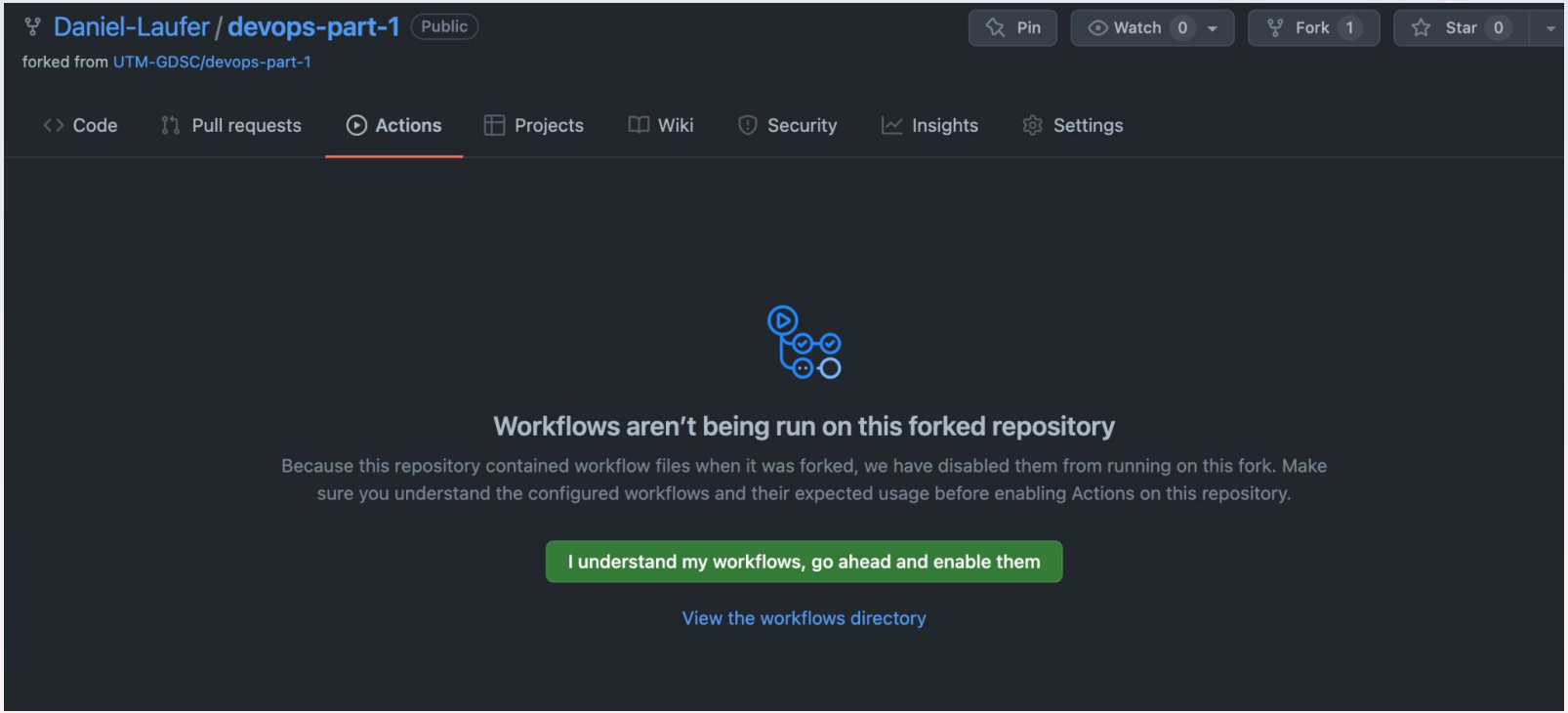


- NEVER REVEAL THIS TOKEN PUBLICALLY (ex. Don’t hardcode the actual value of the token in your github actions script)

Deploying to firebase ctnd

Let’s go over and explain what is happening in the **deploy.yaml** script found here (don’t worry, the finished script is already on your computer if you forked the workshop GitHub repository 😄)

GitHub Actions config

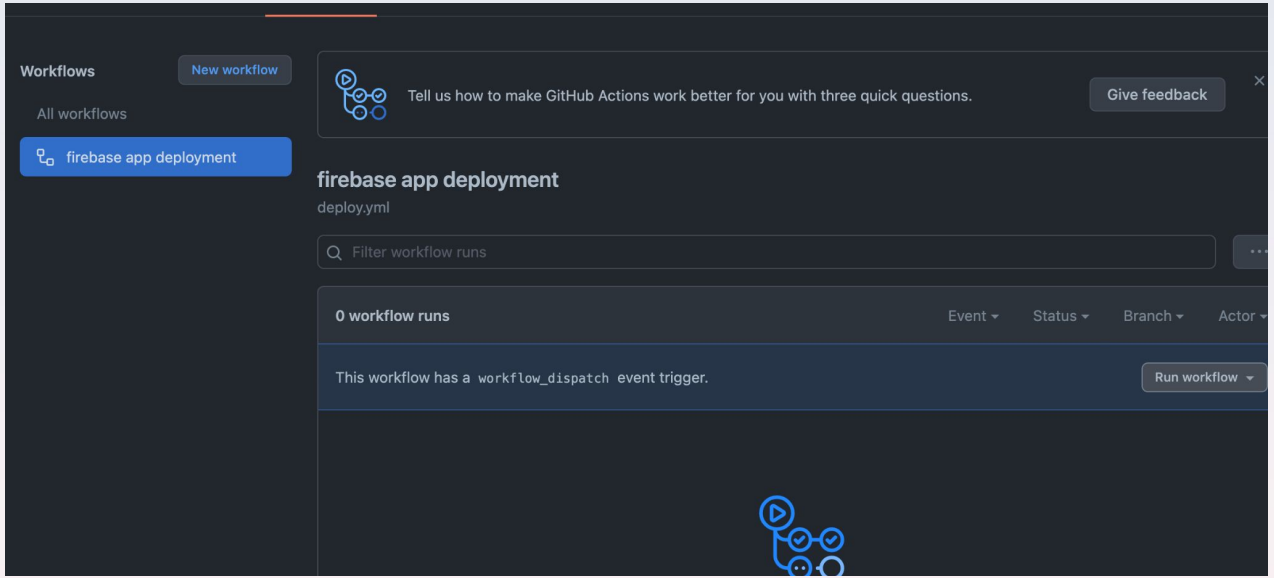


GitHub Actions config

- If you got everything done up to this point, try making a change to the React code and pushing it to the repo (careful! Your changes might break the automated tests I set up 😄 Try just adding <h1>hello</h1> somewhere (that shouldn't break anything :)))
- If you click the "Actions" tab in the repository you should see the Action running!

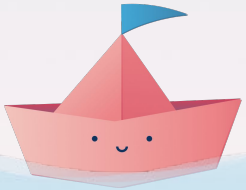
A final note

- You can run workflows manually by clicking “run workflow as seen below



Thank you!

Any questions?



We have a React App

Create a firebase project

“Firebase-ify” our React project

We can deploy it manually by typing in: firebase deploy

We created a CI/CD pipeline using GitHub actions:
(Writing a script that GitHub ran on its servers)
THIS RAN ON EACH PUSH TO THE MAIN BRANCH IN OUR GITHUB REPOSITORY

- Installed dependencies for the react app, Built the react app, ran the tests
  - If any of the tests failed, the GitHub action would stop running and tell us that something went wrong
- Deploy the firebase app to the internet