Worksheet

Aim

The aim of this session is to create a node backend service and deploy it to fly.io. During each step you'll learn the relevant cli commands to download code, run servers, update git repositories and even deploy services.

Getting Started

This will require having a laptop that is prepared for developement. You'll need to make sure you have all of various tooling installed before you get going. If you have any problems at all, just ask for help.

References

Git

- How to save your ass when you have problems with git - https://ohshitgit.com/
- Simple but in depth explanation of how git works - https://xosh.org/explain-git-in-simple-words/
- Pro Git book, highly recommended if you want to understand advance features - https://git-scm.com/book/en/v2
- MIT "The Missing Semester" on Git (Everything your uni course doesn't teach that you need to learn yourself) - https://missing.csail.mit.edu/2020/version-control/

Docs

General Docs

- Best docs for all things web - https://devdocs.io/
- Mozilla Reference - https://developer.mozilla.org/en-US/docs/Web
- Front End Web Tutorials - https://developer.mozilla.org/en-US/docs/Learn
- Awesome terminal environment - https://medium.com/@satriajanaka09/setup-zsh-oh-my-zsh-powerlevel10k-on-ubuntu-20-04-c4a4052508fd

Free Stuff

- All the free GitHub Student stuff - https://education.github.com/pack#offers

Tools used by this worksheet

- Node LTS docs - https://nodejs.org/dist/latest-v20.x/docs/api/
- node package manager - https://docs.npmjs.com/
- Fly.io free app hosting - https://fly.io/
- ExpressJS, simple production grade node server - https://expressjs.com/
- PugJS, easy to use html templates for node - https://pugjs.org/api/getting-started.html
- Terminal shortcuts cheatsheet (Command Control + Command Recall is especially useful) - https://kapeli.com/cheat_sheets/Bash_Shortcuts.docset/Contents/Resources/Documents/i
- Nodemon, automatically restart your server when you edit your code - https://github.com/remy/nodemon
- Github Actions - https://docs.github.com/en/actions/learn-github-actions

Requirements

Windows Specific

- Install WSL
- Configure VSCode for WSL
- Run all commands in Ubuntu WSL!

Everything Else

- GitHub Account
- Fly.io Account
  - Add a card to fly.io it's free to use for hobbyist but requires a card for billing
  - See pricing information https://fly.io/docs/about/pricing/ - be careful scaling up resources!
- VSCode Editor
  - VSCode Plugins

- HomeBrew
- Docker
- gh-cli
- flyctl
- nvm
- Node LTS - `nvm install --lts && nvm use --lts`

Part One

Get you git environment set up so you can download repositories and view them in your editor.

Download this project

Setup your environment locally by logging into GitHub and downloading this project.

- `gh --help` , `gh -h` , `man gh` - see help information, all services should have `-h` , `--help` or a manpage
- `gh auth login` - Login to github on the command line, select the following options
  - `Github.com`
  - `HTTPS`
  - `Authenticate Git` - `Y` (Terminal User Interfaces will use Y for Yes and N for No when asking questions)
  - `Login with a web browser`
  - Copy code, press enter, paste code in browser, enter 2FA
- `git config --global user.email you@youremail.com` - Configure your git identity for your author information
- `git config --global user.name "Your Name"`
- `cd ~/code` - or wherever else you keep your code
- `gh repo clone penson122/git-tutorial`

Set up your editor

Opening a workspace in VSCode

- Windows Users follow Configure VSCode for WSL to ensure wsl is properly configured and you vscode can open projects the subsystem.

Terminal instructions if you're brave

- `cd git-tutorial` - enter the git-tutorial folder
- `code .` - open vscode at this directory

Part Two

Install project dependencies and run this example project.

Install Dependencies

- `npm install` - install project dependencies

Run It Locally

- `npm run start` - start the server
  - `CTRL-C` - kill the server
- `npm run watch` - start the server in watch mode (auto reloads after code change)
  - exit watch mode before running more commands

Part Three

Create your own repo

- `gh repo create`

    - `Create a new repository on GitHub from scratch`
    - Enter a name - (press enter to submit)
    - Enter a description
    - Public
    - README File? `Y`
    - .gitignore? `Y`
    - Scroll down to `Node` - (Use arrow keys)
    - License? `Y`
    - `MIT License`
    - Continue? `Y`
    - Create the new locally? N
- `cd ~/code` - or wherever you keep your projects
- `gh repo clone me/my-repo` - where `me` is your username and `my-repo` is the name you created before
- `cd my-repo` - The project is downloaded to the folder `my-repo`, enter that folder
- `code .` Or follow Part One instructions for your project
- `gh repo view --web` - open your repo in your browser

Create a basic server

Initialise a node project and create a hello world node project

- `npm init -y` - create node package manager config file
- `npm install --save express` - Install Express Framework
- `npm install --save-dev nodemon` - Install Nodemon server watcher
    - This is a developer dependency. It's not used by the production build of your app. Any package that isn't required by your code directly should be installed with `--save-dev` as a good rule of thumb.
- Open your editor and create `index.js`
- Enter the following:

```
const express = require('express');
const path = require('path');


const app = express();
```

```
const port = process.env.PORT || '8080';

app.get('/', (req, res) => {
  res.send('Hello, World!');
});

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`);
});
```

Commit your changes

- `git add .` - stage files from your workspace
- `git status` - see what's going to be committed
- `git commit -m "descriptive messaage"` - create a commit with a message
- `git push` - push changes to remote
- `gh repo view --web` - Click on 2 Commits to see the timeline of changes to your project
- `git log --oneline` - see that same history locally
  - Type `q` to exit `less` mode
- Remember the help and manual pages to see what else you can do

Test It Locally

- `node index.js` - Start the server locally
  - `CTL-C` to exit the server
- Go to `localhost:8080` in your web browser
- Modify `index.js` `Hello, World!` text
- Refresh your browser, why didn't it change?
- Update the following to `package.json`

```
    "scripts": {
        "start": "node index.js",
```

```
        "watch": "nodemon -e js,html,css -w ."
    },
```

- `npm run watch` - Run your server in watch mode
- Open your browser, edit your code, go back to your browser, it changed!

Commit your changes

- `git status`
- `git commit -am "message"` - `-a` automatically stages any modified files (doesn't add files not known to git)
- `git push`

Add Linter

A linter is a code analysis tool which can inform you about errors, style or quality issues and potential bugs. They can be used to enforce code style and autocmatically fix some issues.

- `npm init @eslint/config`
  - Install eslint if it asks
  - `To check syntax, find problems, and enforce code style`
  - `CommonJS`
  - `None of these`
  - `Typescript?` `No`
  - `Node`
  - `Use a popular style guide`
  - `Airbnb`
  - `JavaScript`
  - `Install Now?` `Yes`
  - `npm`

vscode config interlude

- Configure VSCode fix on save

running it manually

- Update `package.json`

```
  "scripts": {
    "start": "node index.js",
    "watch": "nodemon -e js,html,css -w .",
    "lint": "eslint . --fix"
  },
```

- Update `.eslintrc`

```
module.exports = {
  env: {
    browser: true,
    commonjs: true,
    es2021: true,
  },
  extends: 'airbnb-base',
  overrides: [
    {
      env: {
        node: true,
      },
      files: [
        '.eslintrc.{js,cjs}',
      ],
      parserOptions: {
        sourceType: 'script',
      },
    },
```

```
  ],
  parserOptions: {
    ecmaVersion: 'latest',
  },
  rules: {
  },
};
```

- `npm run lint` - Run linter and fix simple errors automatically
  - Try deleting some semicolons and then running this command (or if you have configured fix on save, simply saving the file).
- Commit your changes

Part Four

Update the service so it can be easily tested and write your first test.

Prepare for testing

Install Test Dependencies

- `npm install --save-dev supertest jest`

Improve Server for testability

The `index.js` has both the routes and server listener coupled in the same file. This makes testing more diffficult, separating them allows us to test just the routes without having to restart the server in all the tests.

- `mkdir routes`
- create `routes/main.js`

```
const { Router } = require('express');

const router = Router();

router.get('/', (req, res) => {
  res.send('Hello, World!');
});

module.exports = router;
```

- update index.js

```
const express = require('express');

const router = require('./routes/main')

const app = express();
const port = process.env.PORT || '8080';

app.use('/', router);

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`);
});
```

Add test script

- Update `package.json`

```
  "scripts": {
    "start": "node index.js",
    "watch": "nodemon -e js,html,css -w .",
```

```
    "lint": "eslint . --fix",
    "test": "jest --verbose --coverage",
    "test-watch": "jest --watchAll --verbose --coverage"
  },
```

Add a test

- create `routes/main.test.js`

```
const request = require("supertest");
const express = require("express");
const router = require("./main");

const app = new express();
app.use('/', router);

// Creates a test suite for the root path
describe("Test the root path", () => {
  // Creates a test that expects a 200 status code
  test("It should response the GET method", async () => {
    // Sets the number of assertions expected to be run
    expect.assertions(3);
    // Sends a GET request to the root path
    const response = await request(app).get("/");
    // Expects a 200 status code
    expect(response.statusCode).toBe(200);
    // Expects the response text to be "Hello, World!"
    expect(response.text).toBe("Hello, World!");
    // Expects the response header content-type to be "text/html;
    ↪  charset=utf-8"
```

```
    expect(response.headers["content-type"]).toBe("text/html;
     ↪  charset=utf-8");
  });
});
```

- `npm run test` - see the results of the tests
- `npm run test-watch` - Run the tests in watch mode
  - Try modifying `main.js` or `main.test.js` to see what happens when the expectations fail
- Commit your changes

Part Five

Configure Fly and deploy your first project

FlyCtl Init

- `fly auth login`
- `fly launch`
  - Generate a name or choose something unique
  - Ihr

Deploy

- `fly deploy`
  - Make sure you have docker installed
  - Follow the link at the end to see your server
  - `... fly.dev` is the url
- Go to https://fly.io/dashboard to see your active servers
  - You can only have 3 free VMs!

Part Five

Configure CI/CD and automatically deploy to your cloud environment.

- Don’t commit your api tokens to any repository
- This is very dangerous and can be very costly
- This code is on the public internet where anyone can read it
- Can copy your api tokens and use them for themselves and spend your money or worse host criminal services with your key
- https://blog.gitguardian.com/secrets-credentials-api-git/

Add actions file

- `Create .github/workflows/action.yaml`
- For more details on how this works - https://fly.io/docs/app-guides/continuous-deployment-with-github-actions/

```
name: Fly Deploy
on:
  push:
    branches:
      - main
jobs:
  deploy:
    name: Deploy app
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Use Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '18.x'
```

```
      - run: npm ci
      - run: npm test
      - uses: superfly/flyctl-actions/setup-flyctl@master
      - run: flyctl deploy --remote-only
        env:
          FLY_API_TOKEN: ${{ secrets.FLY_API_TOKEN }}
```

Configure repository secret

- `fly tokens create deploy -x 999999h` and copy the output
- Open Actions Secrets for your repository
  - `gh repo view --web`
  - Settings > Secrets and Variables > Actions
  - New Repository Secret
  - Name: `FLY_API_TOKEN`
  - Secret: Paste the outout of `fly tokens ...` command

Automatically deploy from push

- Commit your changes
- `git push`

Congratulations!

You now have a node web app with hot-reload, linting, automated tests, deployed to the cloud with a full CI/CD pipeline.

Happy Hacking!