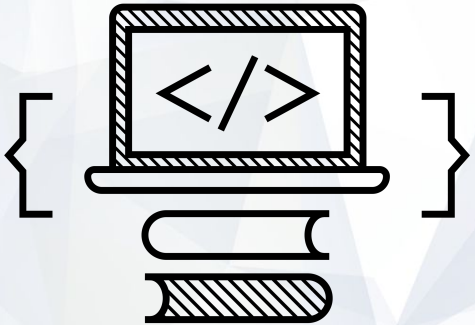
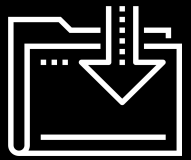


Deployment and CI/CD Automation

Skills Bootcamp in Front-End Web Development

Lesson 14.3



WELCOME

Learning Objectives

By the end of class, you will be able to:

- Create a CD pipeline between a GitHub Repo and Netlify.
- Optimize a CD pipeline between GitHub and Netlify by creating YAML scripts that trigger and run unit tests as well as deploy their code.
- Explain and discuss what YAML is and how it is utilized in a CI/CD pipeline.

What is YAML?

YAML is a human-friendly data-serialization standard for all programming languages.

https://yaml.org/

is a recursive acronym for

YAML **A**in’t **M**arkup **L**anguage

https://en.wikipedia.org/wiki/YAML

YAML...

- Is commonly used for configuration files and in applications where data are being stored or transmitted.
- Targets many of the same communications applications as Extensible Markup Language (XML) but has a minimal syntax that intentionally differs from SGML (standard generalized markup language).
- Uses both Python-style indentation to indicate nesting and a more compact format that uses [...] for lists and {...} for maps so that JSON files are valid YAML 1.2.

https://en.wikipedia.org/wiki/YAML

YAML

In this project, we are using YAML to configure our GitHub and Netlify CI/CD pipeline to:

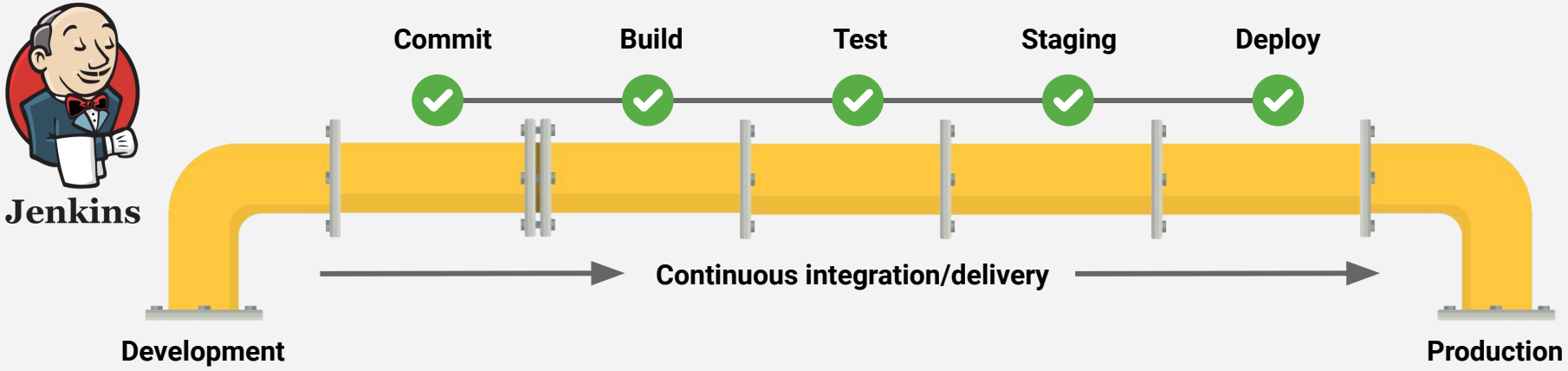
What is a pipeline?

In software development, a **pipeline** is the series of stages that a product goes through on its way to being finished.

It can be useful to think of pipelines as literal pipes—designers and developers push a product down the pipes as they work, refining it as they go.

What Is a Jenkins Pipeline?

A continuous delivery (CD) pipeline is an automated expression of your process for getting software from version control right through to your users and customers.



Every change to your software (committed in source control) goes through a complex process on its way to being released. This process involves building the software in a reliable and repeatable manner, as well as progressing the built software (called a “build”) through multiple stages of testing and deployment.

What is CI/CD?

Continuous integration/ continuous deployment (CI/CD) is the concept of automatically updating machines on your network whenever your configuration files change.

Advance Testing Concepts and CI/CD Automation

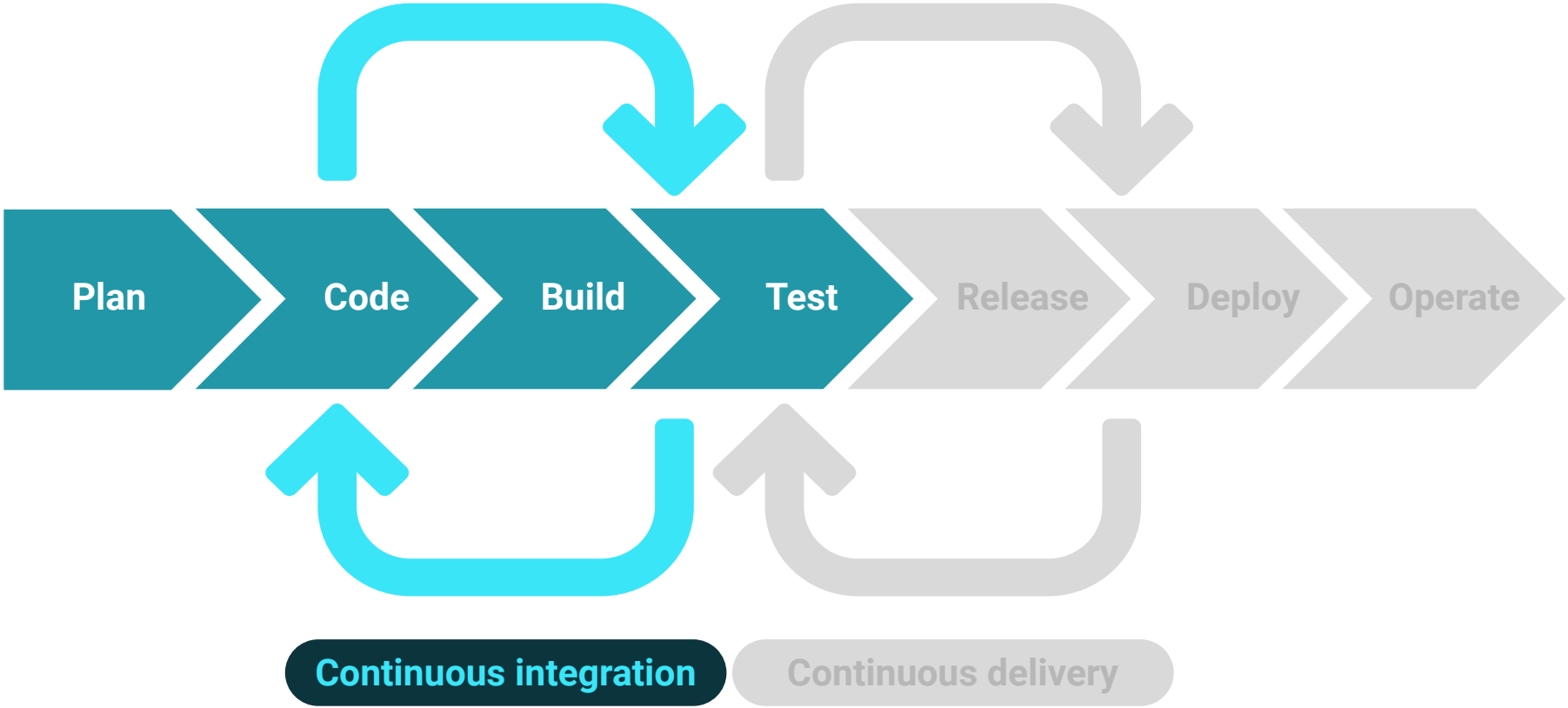
Whenever you change a machine's configuration file:

Continuous integration (CI) allows a team to share and test code efficiently.

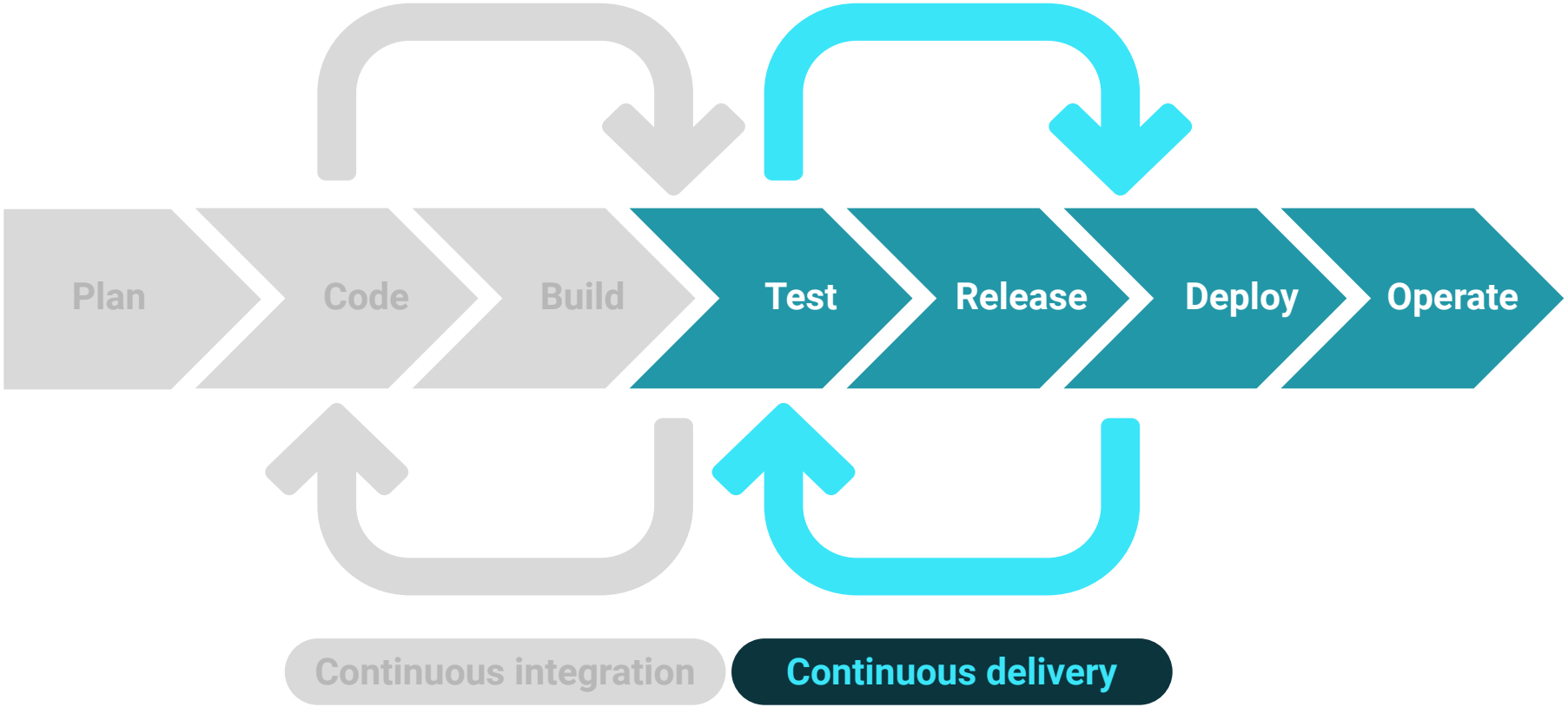
Continuous deployment (CD) is the process of automatically deploying changes to your live environment.

The primary advantage to CI/CD is that it allows you to manage your entire network simply by updating configuration files.

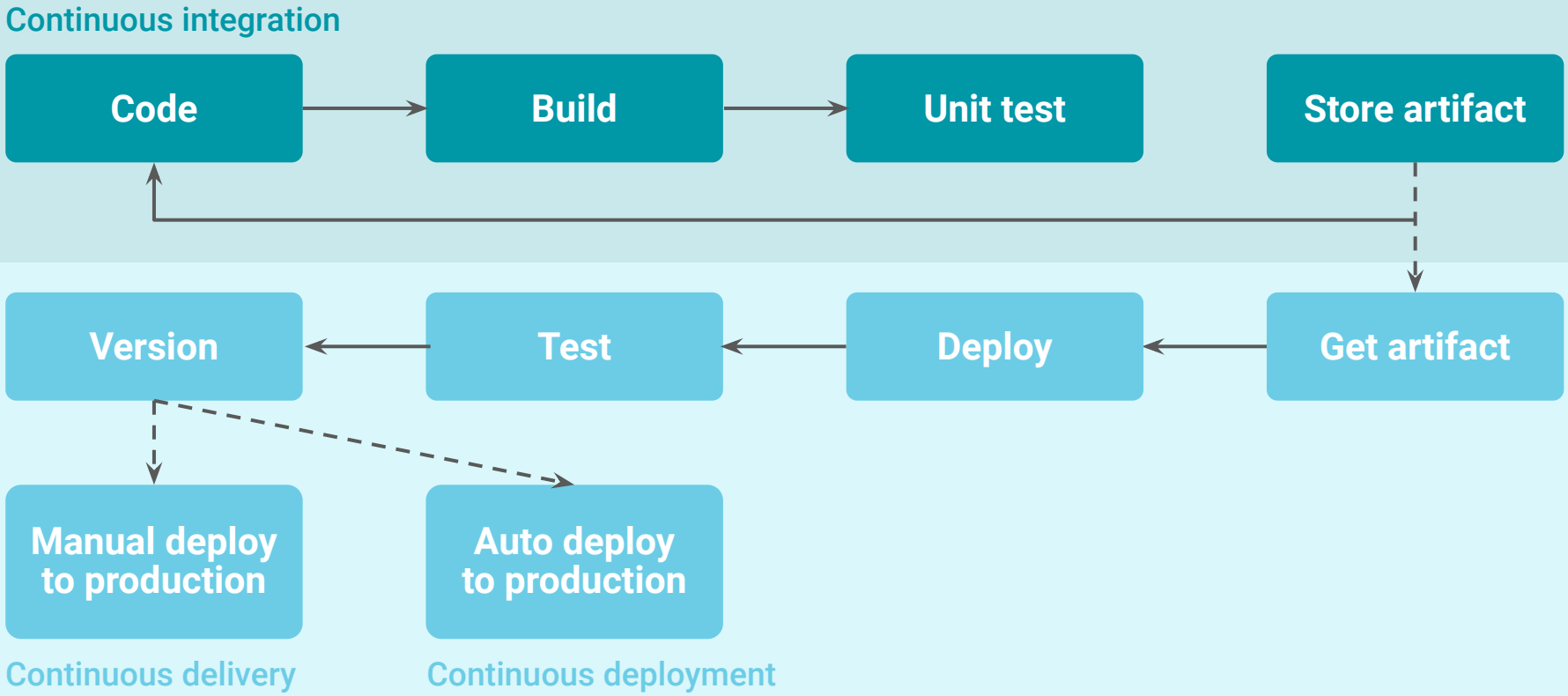
Continuous Integration



Continuous Delivery



CI/CD Pipeline Workflow



Time to Code

Game of Thrones Character Gallery: Deployment

Suggested Time:

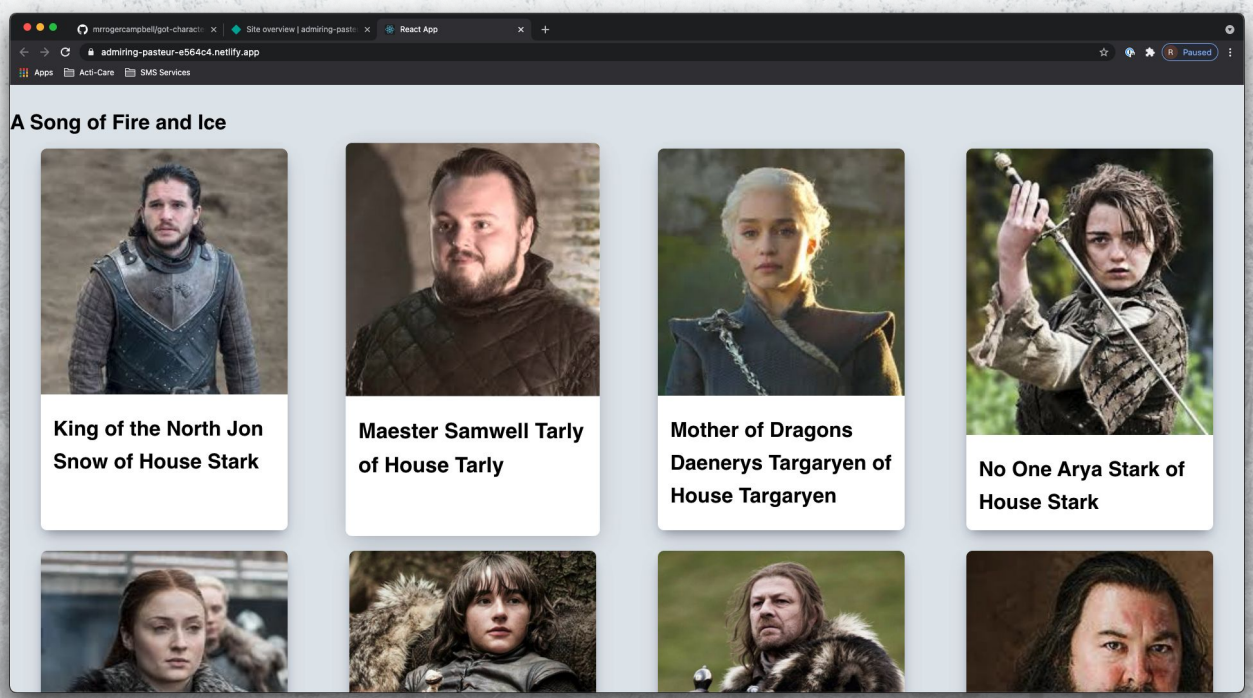
Game of Thrones Character Gallery: Deployment

Instructions:

- Create a Netlify account with GitHub credentials.
- Create a repo called `got-character-gallery` on `GitHub`.
- Create a continuous deployment pipeline between `Netlify` and the `got-character-gallery` repo housed on `GitHub`.

Game of Thrones Character Gallery: Deployment

Example: https://stupefied-joliot-0bb3db.netlify.app



Pre-Written Tests

Tests

Solution: App.test.js

```
// App.test.js

// importing the App Component
import App from '../App'
// importing the shallow render method from the enzyme package
import { shallow } from 'enzyme'

// Created a custom setup method, when invoked it returns a shallowly rendered instance of the App component
const setup = () => shallow(<App />)

// Created a custom setup method which accepts two arguments:
  // 1. wrapper - the wrapper which you want to search; ie the component
  // 2. val - the data-test value you would like to locate
const findByTestAttr = (wrapper, val) => wrapper.find(`[data-test='${val}']`)

// testing to see if
it('App Component Renders Without Error', () => {
  const wrapper = setup()
  const appComponent = findByTestAttr(wrapper, "component-app")

  expect(appComponent.length).toBe(1)
})
```

We are testing to see if the App component renders without errors.

Tests

Solution: CharacterGallery.test.js

```
// CharacterGallery.test.js
import CharacterGallery from '../components/CharacterGallery'
import { shallow } from 'enzyme'

const setup = () => shallow(<CharacterGallery />)

const findByTestAttr = (wrapper, val) => wrapper.find(`[data-test='${val}']`)

it('CharacterGallery Component Renders Without Error', () => {
    const wrapper = setup()

    const charComponent = findByTestAttr(wrapper, "component-char-gallery")

    expect(charComponent.length).toBe(1)
})
```

We are testing to see if the `CharacterGallery` component renders without errors.

In `Character.test.js` we are performing more in-depth testing.

Activity: Develop Character Gallery

In this activity, you will reinforce and build upon the testing and React skills that you have learned so far by designing and developing test based on feature requirements inside the activity's README.

Suggested Time:

Time’s Up! Let’s Review.

Review: Develop Character Gallery

Solution: App.js

```
// App.js
import ./App.css';
import React from 'react';
import CharacterGallery from ./components/CharacterGallery'

function App() {

  return (
    <div data-test="component-app">
      <CharacterGallery />
    </div>
  );
}

export default App;
```

Here we are just importing and rendering the `CharacterGallery` within the `App` component.

Review: Develop Character Gallery

Solution: CharacterGallery.js

```
// CharacterGallery.js
import React from 'react';
// Be sure to copy the json file from the activity directory into your project
import characterData from '../data/characterData.json'
Import Character from './Character'

const CharacterGallery = () => {
    const listOfChars = characterData.map((char, i) => <Character {...char} key={i} />)
    return (
        <div data-test="component-char-gallery">
            {listOfChars}
        </div>
    );
};

export default CharacterGallery;
```

Review: Develop Character Gallery

In this component, we are:

01

Importing all the character data from `characterData.json` and storing them in a variable called `characterData`.

02

Iterating over the `characterData` variable with a `map method` and returning a new array that contains an instance of the `Character` component for each character dataset in the `characterData` array.

- We are storing this new array within the `listOfChars` variable.

03

Rendering the `listOfChars` variable within the `CharacterGallery` component.

Review: Develop Character Gallery

Solution: Character.js

```
// Character.js
import React from 'react';

const Character = ({ name, imgUrl, birth, death, race, realm, spouse }) => {
    return (
        <div data-test='component-character'>
            <h1 data-test='char-name'>{name}</h1>

            <img data-test='char-img' src={imgUrl} alt={name} />

            <ul data-test='char-list'>
                <li data-test='char-birth'>
                    Date of Birth: {birth}
                </li>
                <li data-test='char-death'>
                    Date of Death: {death}
                </li>
                <li data-test='char-race'>
                    Race: {race}
                </li>
                <li data-test='char-realm'>
                    Realm: {realm}
                </li>
                <li data-test='char-spouse'>
                    Spouse: {spouse}
                </li>
            </ul>
        </div>
    );
};

export default Character;
```

Review: Develop Character Gallery

Here, we are:

01

Destructuring the props being passed to the Character component

- For more on destructuring props in React, see this article.

02

Rendering each destructed prop into its corresponding element.

Activity: Repo Setup and Deployment

In this activity, you will reinforce and build upon the development skills that you have learned so far by creating a `remote repo` on GitHub and connecting it to your `local repo` and deploying to Netlify

Suggested Time:

Time’s Up! Let’s Review.

Time to Code

Configure GitHub and Netlify

Suggested Time:

Configure GitHub and Netlify

If we used Netlify to build and deploy, we would link our GitHub repo directly.

We would also use their build preview feature instead of creating a “separate” dev website.

Using Netlify is cleaner, but they limit their free build minutes to 300/month.

GitHub Actions has 3,000 for private repos and is unlimited for public repos.

Time to Code

Set Up Workflow Actions and YAML

Suggested Time:

Solution: Set Up Workflow Actions and YAML

01

In the **root of your project**, create a `.github` directory.

- `mkdir .github`

02

Inside of `.github`, create a directory called `workflows`.

- `mkdir ./.github/workflows`

03

Inside of `workflows/push.yml`, copy the code from:
[06-We_SetupWorkflowActionsAndYaml]

Time to Code

Push It!

Suggested Time:

Today's Challenge:

Updating Your React Portfolio

Suggested Time:

The End