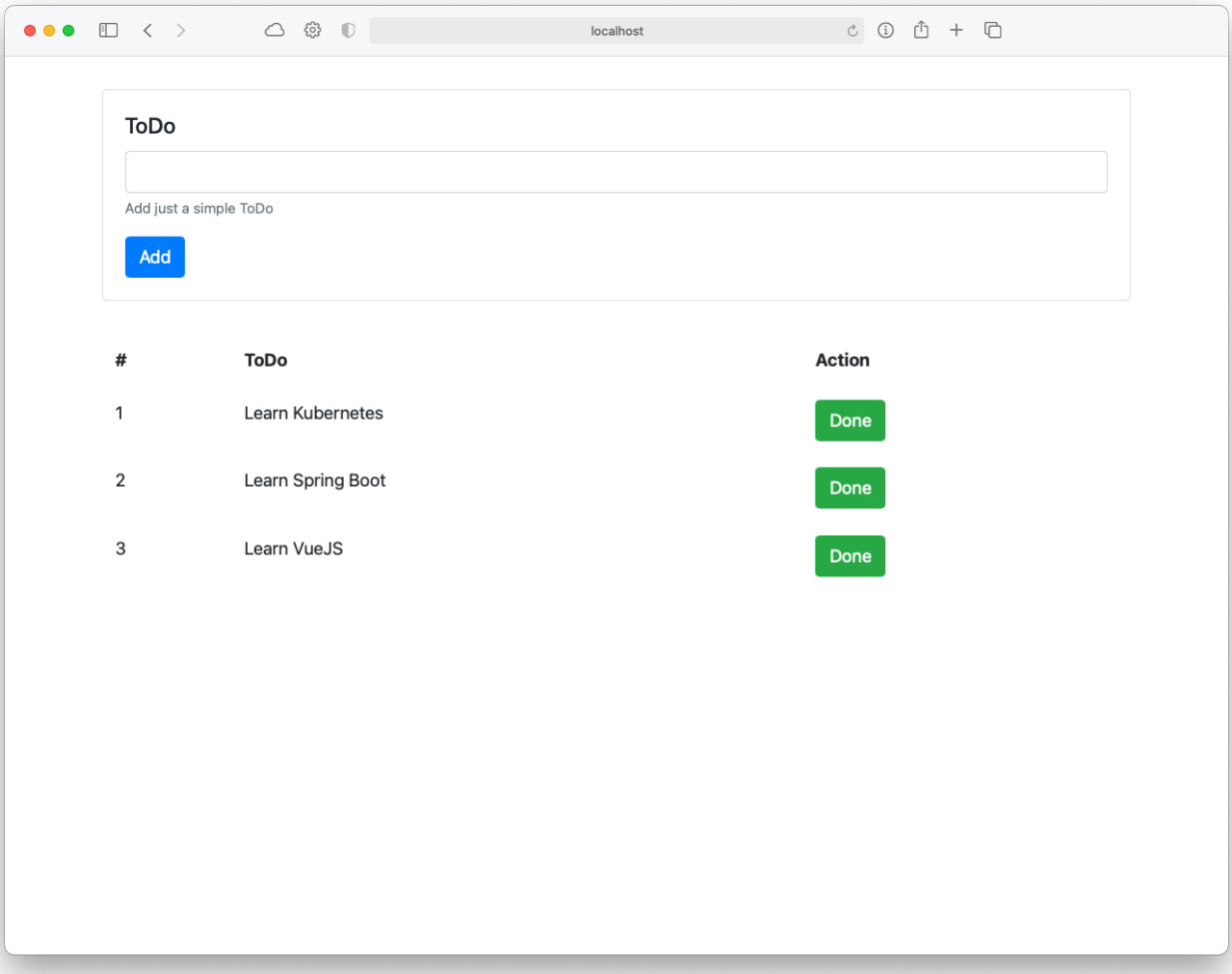
Simple ToDo App with VueJS and Spring Boot

Requirements

- Java 8
- NodeJS / NPM
- VueJS CLI
- Docker / Docker Desktop
- CodeReady Containers
- IDE: VS Code | JetBrains | Eclipse | Spring Tools

VueJS UI

We are going to create a UI for the ToDo Service



1. Create the Project and Insatll Axios

```
vue init webpack todo-ui
cd todo-ui
npm install axios
```

2. Add the Bootstrap CSS to the `index.html` file.

   index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <title>ToDo UI</title>


    <link rel="stylesheet" href=
"https://cdn.jsdelivr.net/npm/bootstrap@4.6.0/dist/css/bootstrap.min.css"
    integrity="sha384-
B0vP5xmATw1+K9KRQjQERJvTumQW0nPEzvF6L/Z6nronJ3oUOFUFpCjEUQouq2+l"
    crossorigin="anonymous">


  </head>
  <body>
    <div id="app"></div>
  </body>
</html>
```

3. Create the `ToDo.vue` file with the following content:

src/components/ToDo.vue

```
<template>
  <div class="row">
    <div class="col-12">
      <div class="card">
      <div class="card-body">
        <h5 class="card-title">ToDo</h5>
        <form @submit="checkForm">
            <div class="form-group">
                <input type="text" class="form-control" @keyup.enter.prevent=
"checkForm" v-model="todo">
                <small class="form-text text-muted">Add just a simple ToDo</small>
            </div>
            <button class="btn btn-primary" type="submit">Add</button>
        </form>
      </div>
  </div>
</div>

  </div>
</template>

<script>

import { EventBus } from "../utils/event-bus.js";

export default {
  name: 'ToDo',
  data () {
    return {
      todo: ''
    }
  },
  methods: {
    checkForm: function(e){
      EventBus.$emit("new-todo", { description: this.todo });
    }
  }
}
</script>


<style>

</style>
```

4. Create the `ToDoList.vue` file with the following content:

src/components/ToDo.vue

```
<template>
  <div class="row todo-list">
    <div class="col-12">
      <table class="table table-borderless table-hover">
        <thead>
          <tr>
            <th scope="col">#</th>
            <th scope="col">ToDo</th>
            <th scope="col">Action</th>
          </tr>
        </thead>
        <tbody>

              <tr v-for="(todo, index) in todos" :key="index">
                  <td>{{ index + 1}}</td>
                  <td>{{ todo.description }}</td>
                  <td>
                    <button class="btn btn-success" @click="doneToDo(todo.id,
$event)">Done</button>
                  </td>
              </tr>

        </tbody>
      </table>
    </div>
  </div>
</template>

<script>

import axios from "axios"
import { EventBus } from "../utils/event-bus.js";


export default {
  name: "ToDoList",
  data: function() {
    return {
      todos: []
    }
  },
  mounted() {
    EventBus.$on("new-todo", todo => { this.newToDo(todo)});
  },
  created() {
    this.getToDos();
  },
  methods: {
    getToDos: function() {
        axios.get(process.env.ROOT_API)
          .then(response => this.todos = response.data);
```

```
    },

    newToDo: function(todo){
      axios.post(process.env.ROOT_API, todo)
          .then(response => this.todos.push(response.data));
    },

    doneToDo: function(id, event){
       axios.delete(process.env.ROOT_API + "/" + id)
          .then(response => console.log(response.data));
      event.preventDefault();
      this.todos = this.remove(id);
    },

    remove: function(value){
      return this.todos.filter(function(ele){
            return ele.id != value;
        });
    }

  }
}
</script>

<style>

.todo-list {
  margin-top: 30px;
}

</style>
```

5. Create the `event-bus.js` file with the following content:

   src/utils/event-bus.js

```
import Vue from 'vue';
export const EventBus = new Vue();
```

6. Add the `ROOT_API` variable to the `dev.env.js` and `prod.env.js`.

config/dev.env.js

```
'use strict'
const merge = require('webpack-merge')
const prodEnv = require('./prod.env')

module.exports = merge(prodEnv, {
  NODE_ENV: '"development"',
  ROOT_API: '"http://localhost:8081/todos"'
})
```

config/dev.env.js

```
'use strict'
module.exports = {
  NODE_ENV: '"production"',
  ROOT_API: '"http://todo:8081/todos"'
}
```

As recommendation, the service call should be internal and a different port, in this case it will be `http://todos:8081/todos`

7. Test you app with:

```
npm run dev
```

8. Create a `Dockerfile` witht eh following content

```
# build stage
FROM node:lts-alpine as build-stage
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

# production stage
FROM bitnami/nginx as production-stage
COPY --from=build-stage /app/dist /app
EXPOSE 8080
CMD ["nginx", "-g", "daemon off;"]
```

9. **[OPTIONAL]** Build you image, test it and push it to your Registry.

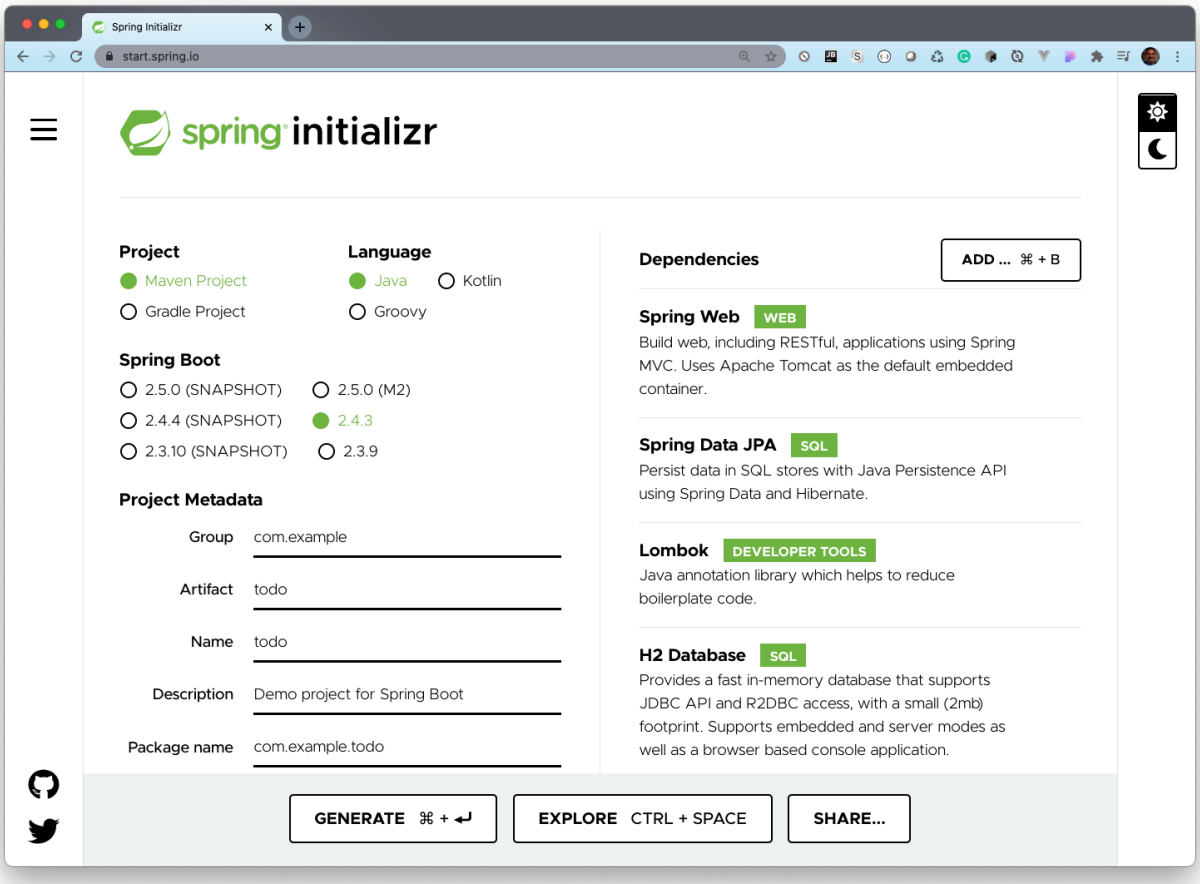
```
docker build -t <your-id>>/todo-ui:v1 .
docker run -it -p 8080:80 --rm --name todo-ui <your-id>/todo-ui:v1
docker push <your-id>>/todo-ui:v1
docker tag <image-id> <your-id>>/todo-ui:latest
docker push <your-id>>/todo-ui:latest
```

This step is **OPTIONAL**, you **DON'T** need it for deploying in OpenShift.

Spring Boot

1. Go to https://start.spring.io/

   Add the following Dependencies: *Web*, *Data JPA*, *H2*, *Lombok*, *MySQL Driver*



2. Add the following classes/interface:

src/main/java/com/example/todo/ToDo.java

```
package com.example.todo;


import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.hibernate.annotations.GenericGenerator;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

@NoArgsConstructor
@AllArgsConstructor
@Data
@Entity
@Table(name = "todo")
public class ToDo {

    @Id
    @GeneratedValue(generator = "uuid")
    @GenericGenerator(name = "uuid", strategy = "org.hibernate.id.UUIDGenerator")
    private String id;
    private String description;

}
```

src/main/java/com/example/todo/ToDoRepository.java

```
package com.example.todo;

import org.springframework.data.repository.CrudRepository;

public interface ToDoRepository extends CrudRepository<ToDo,String> {
}
```

src/main/java/com/example/todo/ToDoNotFoundException.java

```
package com.example.todo;

public class ToDoNotFoundException extends RuntimeException{

    public ToDoNotFoundException(){
        super("ToDo provided was not found");
    }

    public ToDoNotFoundException(String id){
        super(String.format("Todo with id: %s was not found",id));
    }
}
```

src/main/java/com/example/todo/ToDoController.java

```
package com.example.todo;

import lombok.RequiredArgsConstructor;
import org.springframework.web.HttpRequestMethodNotSupportedException;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.mvc.support.DefaultHandlerExceptionResolver;

import javax.servlet.http.HttpServletRequest;
import java.util.HashMap;
import java.util.Map;
import java.util.stream.Collectors;
import java.util.stream.StreamSupport;

@CrossOrigin("*")
@RequiredArgsConstructor
@RequestMapping("/todos")
@RestController
public class ToDoController {

    private final ToDoRepository toDoRepository;

    @GetMapping
    public Iterable<ToDo> getAll(){
        return this.toDoRepository.findAll();
    }

    @GetMapping("/{id}")
    public ToDo getById(@PathVariable String id){
        return this.toDoRepository.findById(id).orElseThrow(() ->new
ToDoNotFoundException(id));
    }

    @PostMapping
```

```
    public ToDo newToDo(@RequestBody ToDo toDo){
        return this.toDoRepository.save(toDo);
    }

    @DeleteMapping("/{id}")
    public ToDo deleteById(@PathVariable String id){
        ToDo toDo = this.toDoRepository.findById(id).orElseThrow(() ->new
ToDoNotFoundException(id));
        this.toDoRepository.deleteById(toDo.getId());
        return toDo;
    }

    @ExceptionHandler({ToDoNotFoundException.class})
    public Map<String, String> handleError(HttpServletRequest req, Exception ex){
        Map<String,String> result = new HashMap<>();
        result.put("message",ex.getMessage());
        return result;
    }

}
```

3. Add the `data.sql` for initial data.

   src/main/resources/data.sql

```
insert into TODO (id,description)
values ('1f31285e-2a4d-4d2c-ba09-9718fc1f3e4c', 'Learn Kubernetes');
insert into TODO (id,description)
values ('aa6d0450-fc8e-4d57-a47c-a7317dac60fc', 'Learn Spring Boot');
insert into TODO (id,description)
values ('bd18b3b8-a803-4d5f-8b97-d1a2298fb563', 'Learn VueJS');
```

4. Add the following content to the `application.properties`

src/main/resources/application.properties

```
# Spring
spring.application.name=todo

# MVC
server.error.whitelabel.enabled=false

# Server
server.port=${PORT:8081}

# H2
spring.h2.console.enabled=true

# Info
info.app.name=${spring.application.name}
info.app.developer.name=Felipe Gutierrez
info.app.developer.email=felipeg@email.com

# Data
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

5. Add the `index.html` and `404.html pages`

src/main/resources/static/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>ToDo Service</title>
    <link rel="stylesheet" href=
"https://cdn.jsdelivr.net/npm/bootstrap@4.6.0/dist/css/bootstrap.min.css"
integrity="sha384-B0vP5xmATw1+K9KRQjQERJvTumQW0nPEzvF6L/Z6nronJ3oUOFUFpCjEUQouq2+l"
crossorigin="anonymous">
    <style>
        body {
            font-family: Avenir SansSerif Verdana;
        }
    </style>
</head>
<body>
<div class="container">
    <div class=""row>
        <div class="col-12">
            <div class="card" style="top: 10px;">
                <div class="card-header">
                    <h1>Welcome</h1>
                </div>
                <div class="card-body">
                    <h5 class="card-title">ToDo Service</h5>
                    <p class="card-text">Everything about ToDos.</p>
                    <a class="btn btn-primary" href="/todos">API</a>
                </div>
            </div>
        </div>
    </div>
</div>
</body>
</html>
```

src/main/resources/static/error/404.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>ToDo Service</title>
    <link rel="stylesheet" href=
"https://cdn.jsdelivr.net/npm/bootstrap@4.6.0/dist/css/bootstrap.min.css"
integrity="sha384-B0vP5xmATw1+K9KRQjQERJvTumQW0nPEzvF6L/Z6nronJ3oUOFUFpCjEUQouq2+l"
crossorigin="anonymous">
    <style>
        body {
            font-family: Avenir SansSerif Verdana;
        }
    </style>
</head>
<body>
<div class="container">
    <div class=""row>
        <div class="col-12">
            <div class="card" style="top: 10px;">
                <div class="card-body">
                    <div class="alert alert-danger" role="alert">
                        There was an Error!
                    </div>
                    <h5 class="card-title">ToDo Service</h5>
                    <p class="card-text">Everything about ToDos.</p>
                    <a class="btn btn-primary" href="/todos">API</a>
                </div>
            </div>
        </div>
    </div>
</div>
</body>
</html>
```

6. Test you app with:

```
./mvnw spring-boot:run
```

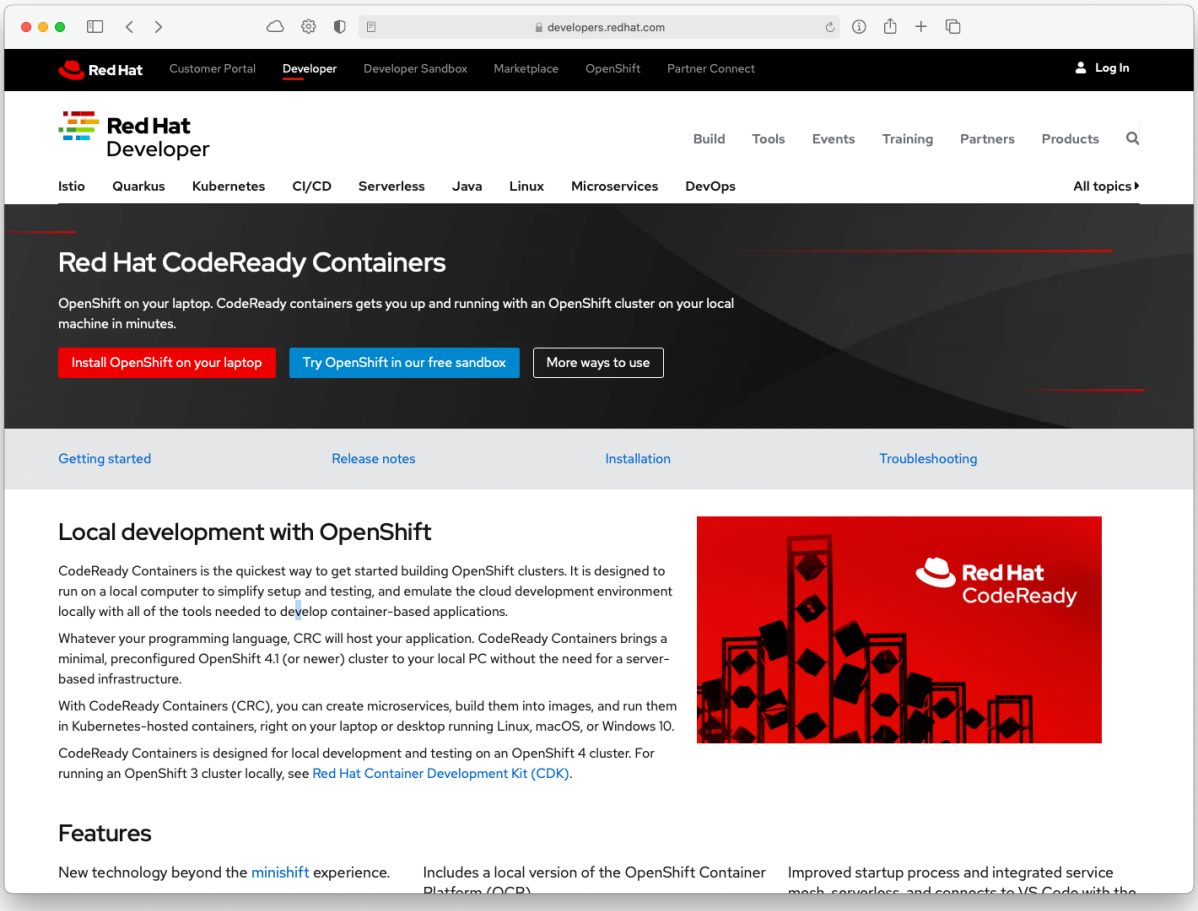
7. **[OPTIONAL]** Create a Docker image, test and push it to the registry.

```
./mvnw spring-boot:build-image -Dspring-boot.build-image.imageName=<your
-id>/todo:v1
docker run -it -p 8081:8081 --rm --name todo <your-id>/todo:v1
docker push <your-id>>/todo:v1
docker tag <image-id> <your-id>>/todo:latest
docker push <your-id>>/todo:latest
```

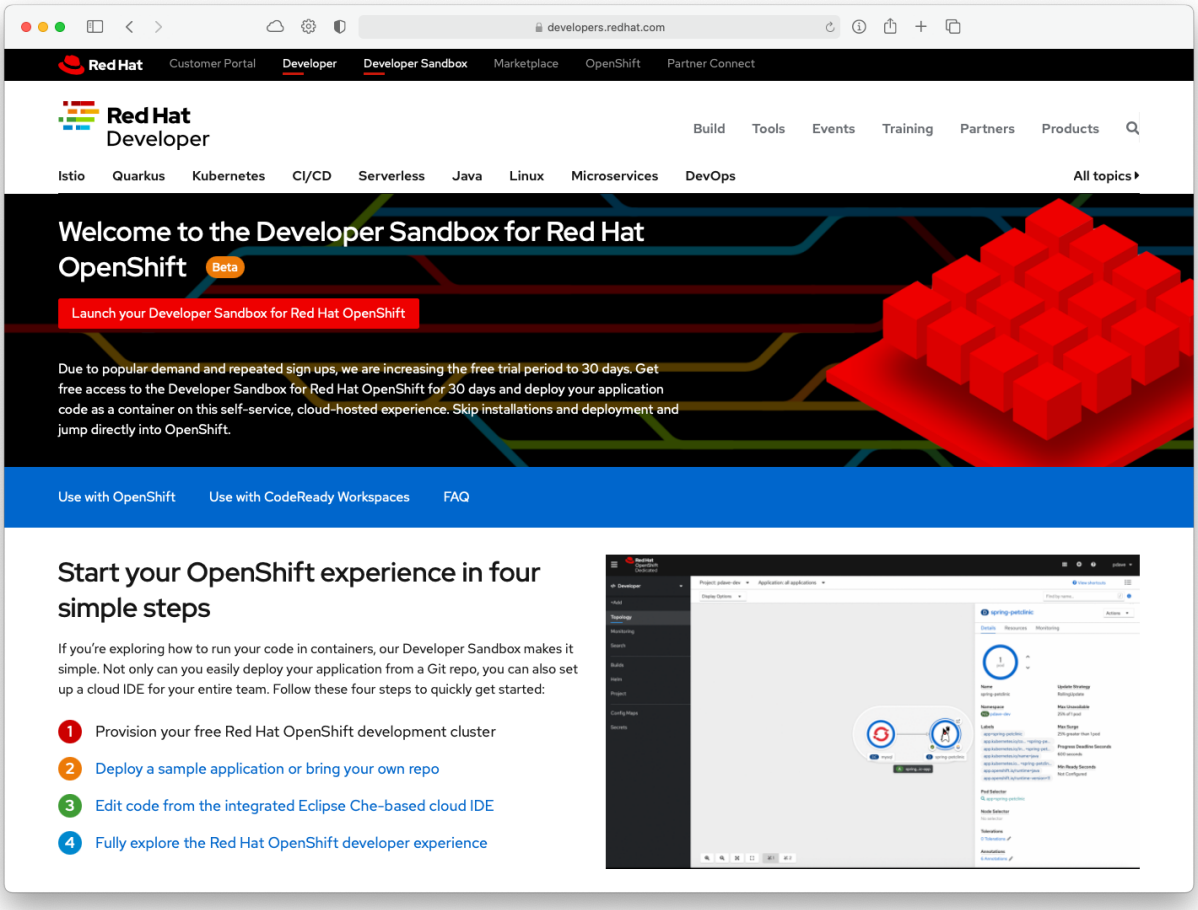
This step is **OPTIONAL**, you **DON'T** need it for deploying in OpenShift.

Deploy the ToDo App in OpenShift

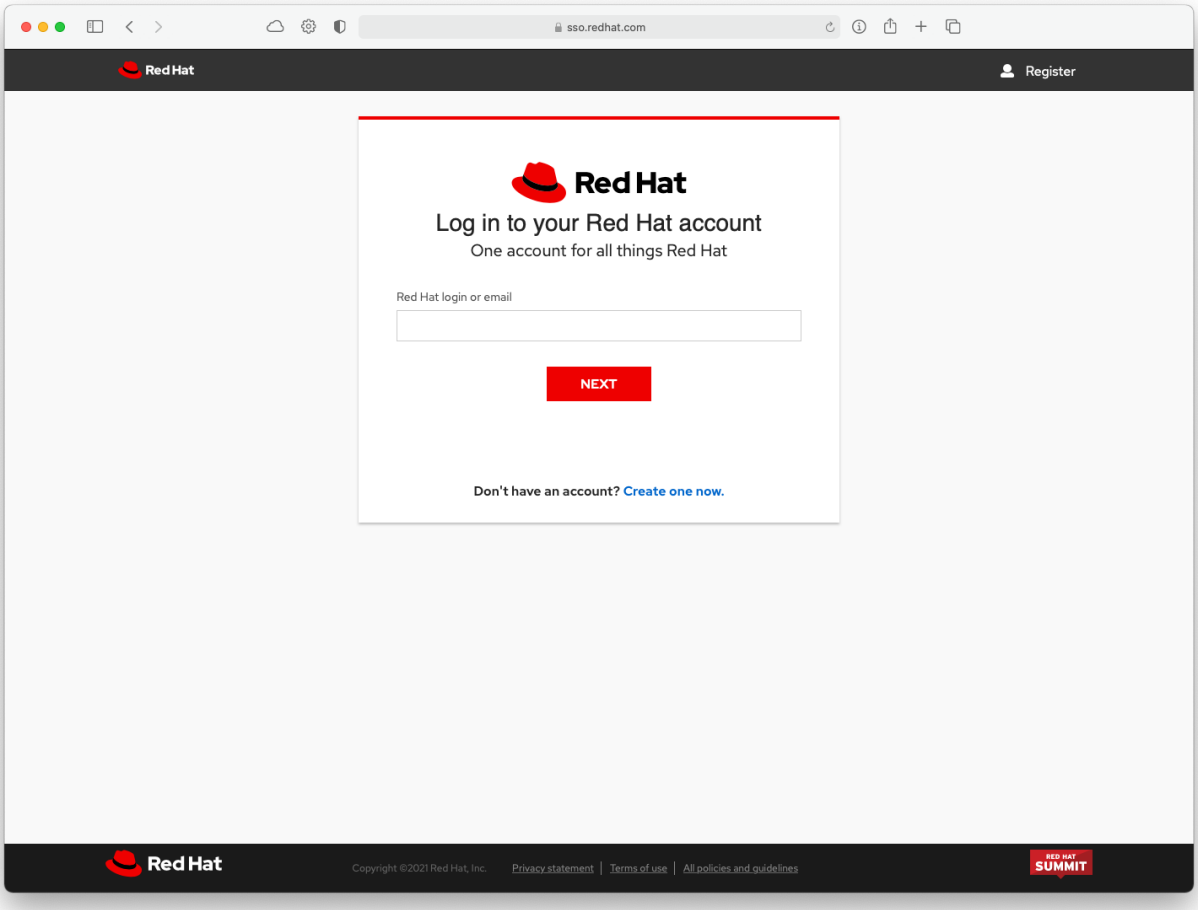
1. Open you account en RedHat con CodeReady Containers: https://ibm.biz/BdfjCM



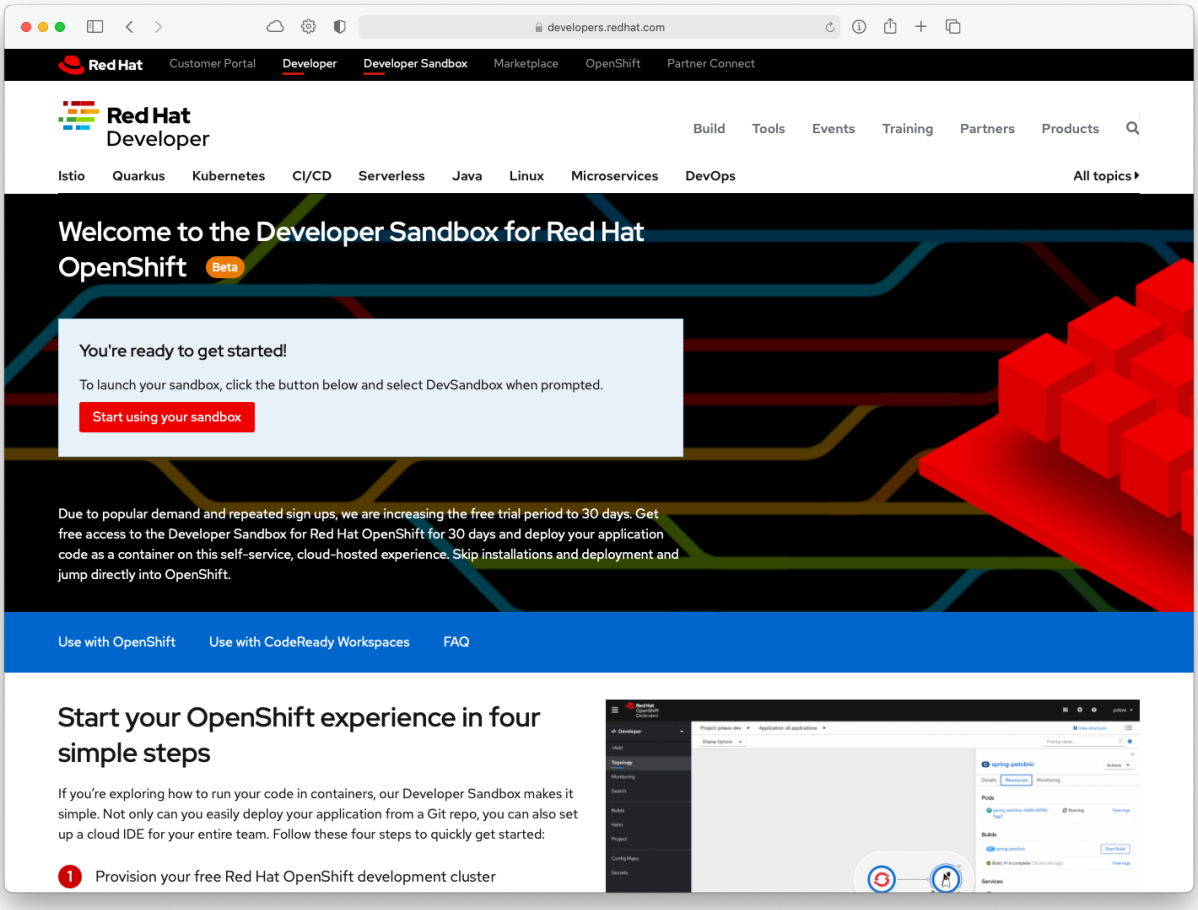
2. Go to the Developer Sandbox and Launch it.



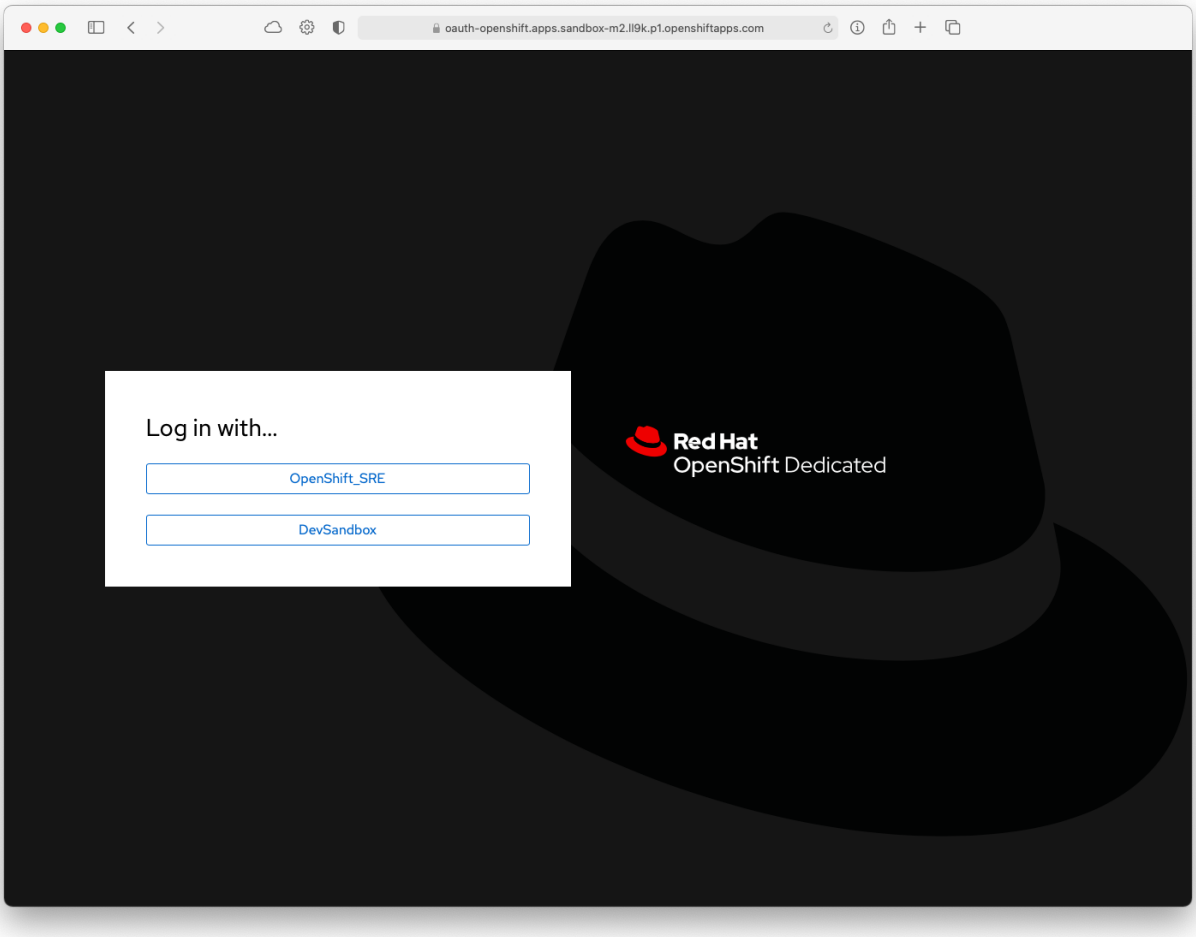
3. Create an account or Login into RedHat Portal



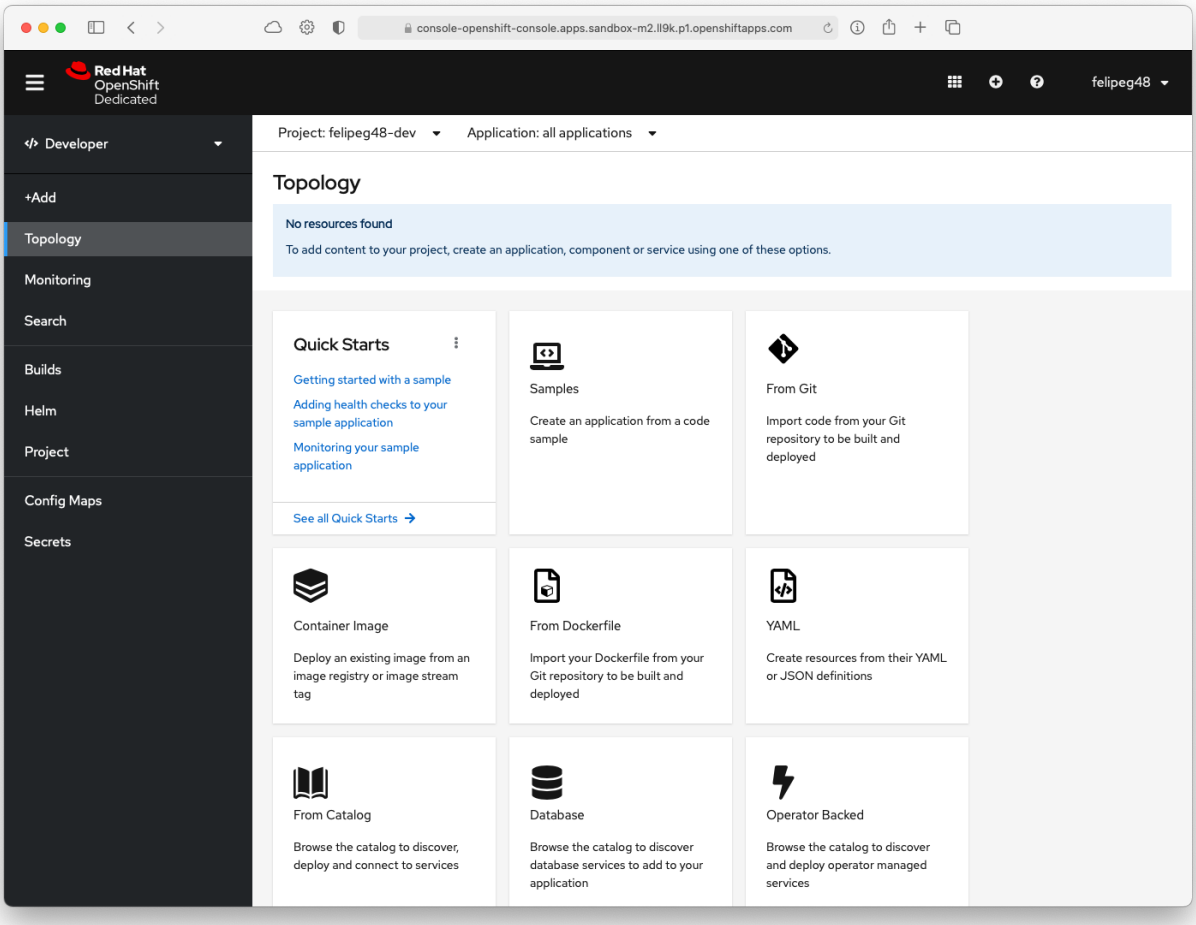
Once you login, click the `Start using your sandbox` button.

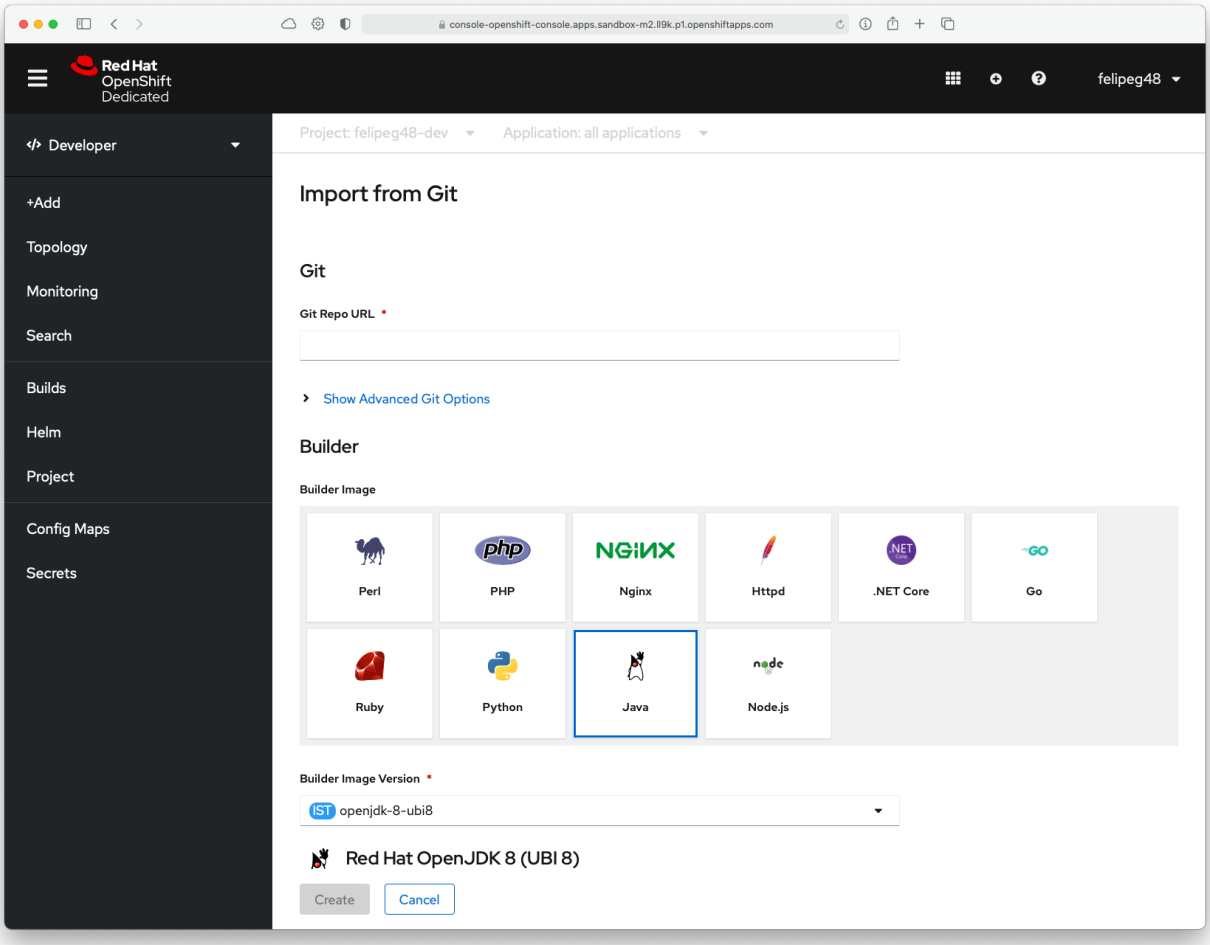


4. Select DevSandBox

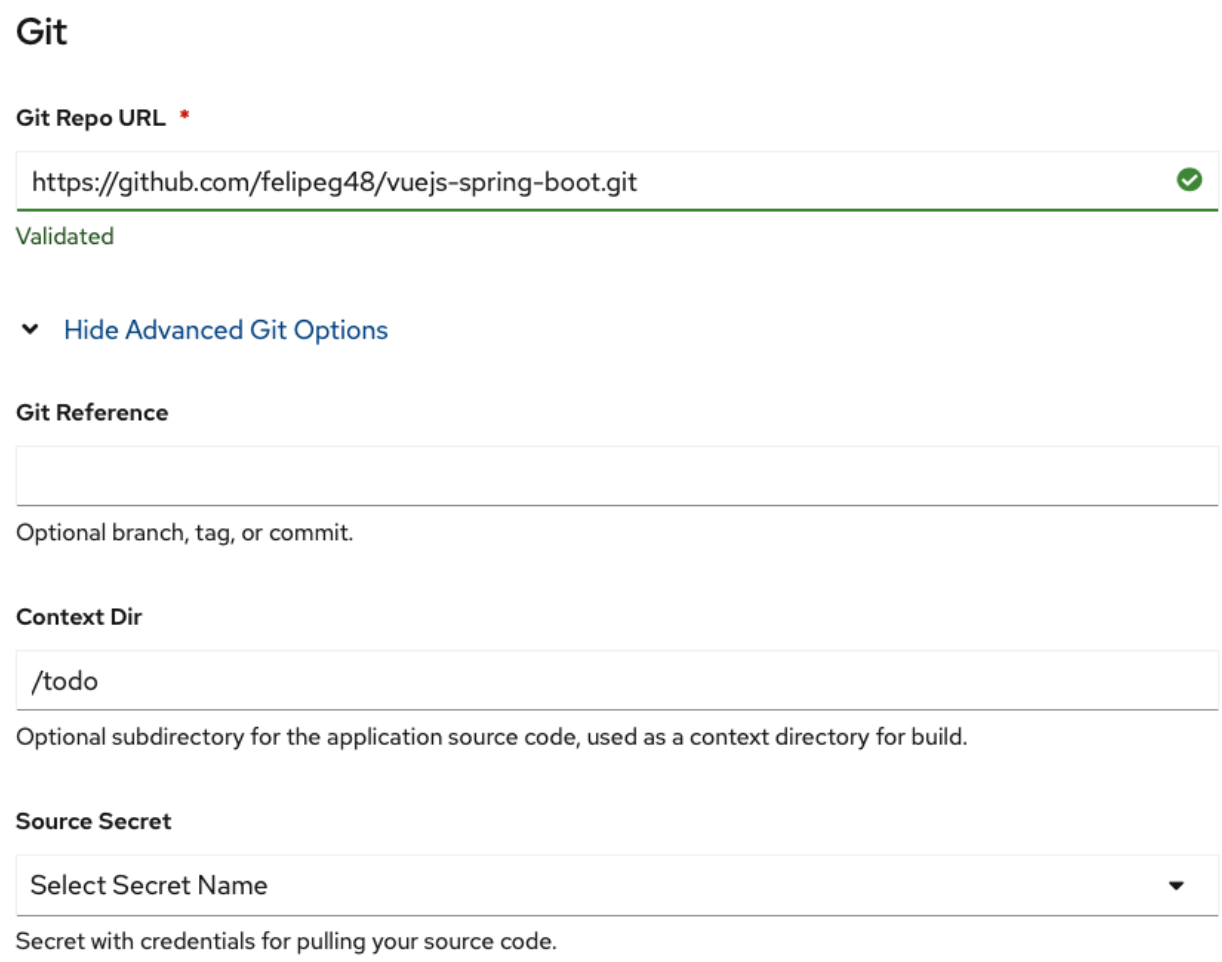


5. It will appear the **Topology** Dashboard. Select from `Git` and paste the `todo` git project.

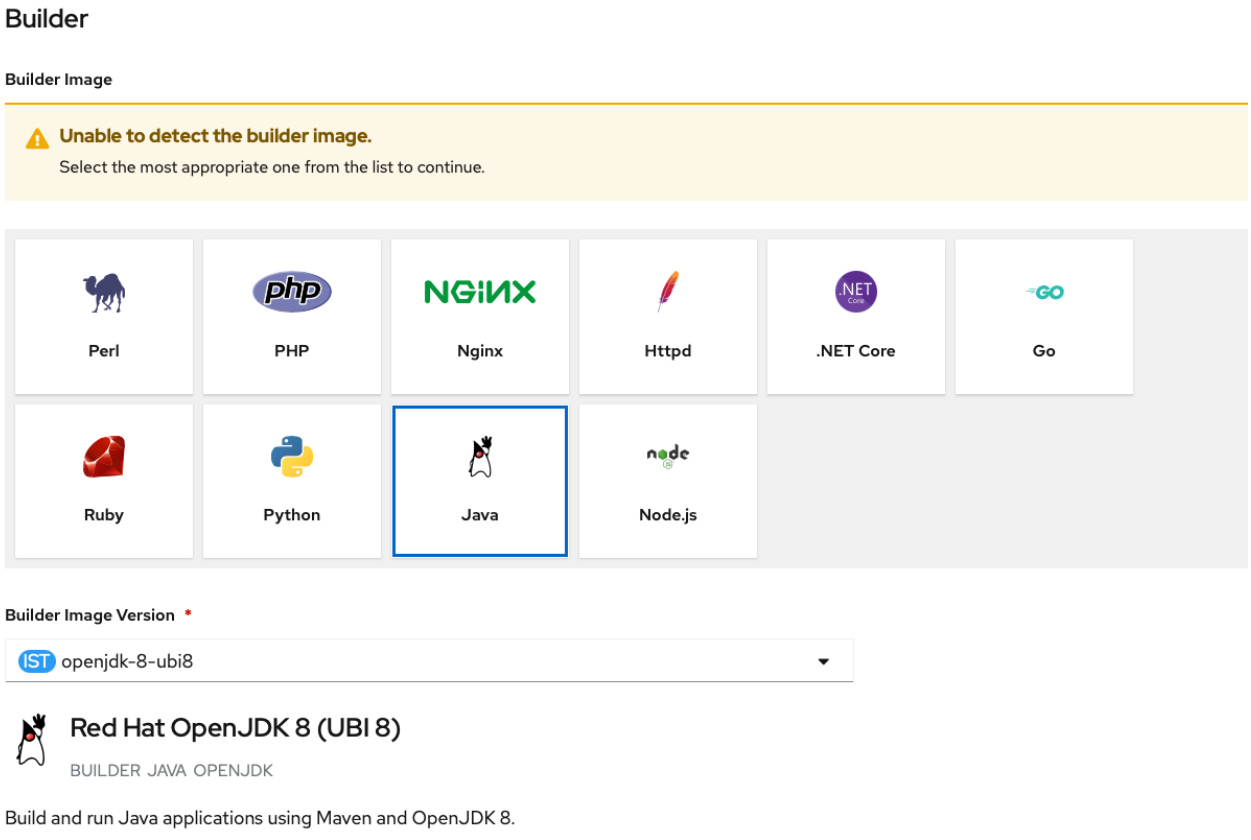

console-openshift-console.apps.sandbox-m2.ll9k.p1.openshiftapps.com
Red Hat OpenShift Dedicated
felipeg48
Developer
Project: felipeg48-dev
Application: all applications
+Add
Topology
Monitoring
Search
Builds
Helm
Project
Config Maps
Secrets
Topology
No resources found
To add content to your project, create an application, component or service using one of these options.
Quick Starts
Getting started with a sample
Adding health checks to your sample application
Monitoring your sample application
See all Quick Starts
Samples
Create an application from a code sample
From Git
Import code from your Git repository to be built and deployed
Container Image
Deploy an existing image from an image registry or image stream tag
From Dockerfile
Import your Dockerfile from your Git repository to be built and deployed
YAML
Create resources from their YAML or JSON definitions
From Catalog
Browse the catalog to discover, deploy and connect to services
Database
Browse the catalog to discover database services to add to your application
Operator Backed
Browse the catalog to discover and deploy operator managed services



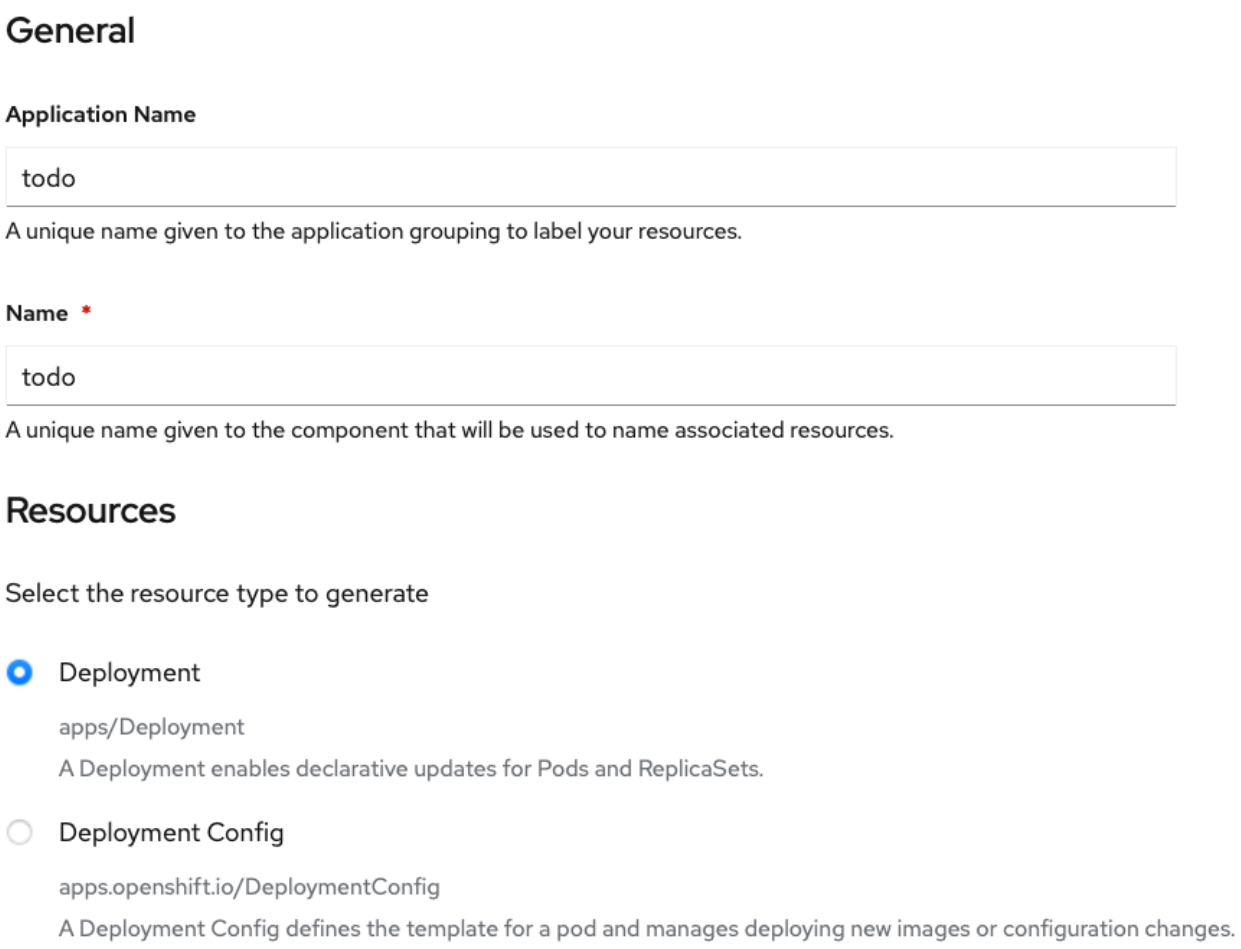
Add the `/todo` in the **Context Dir** in the **Advanced Git Options** section.



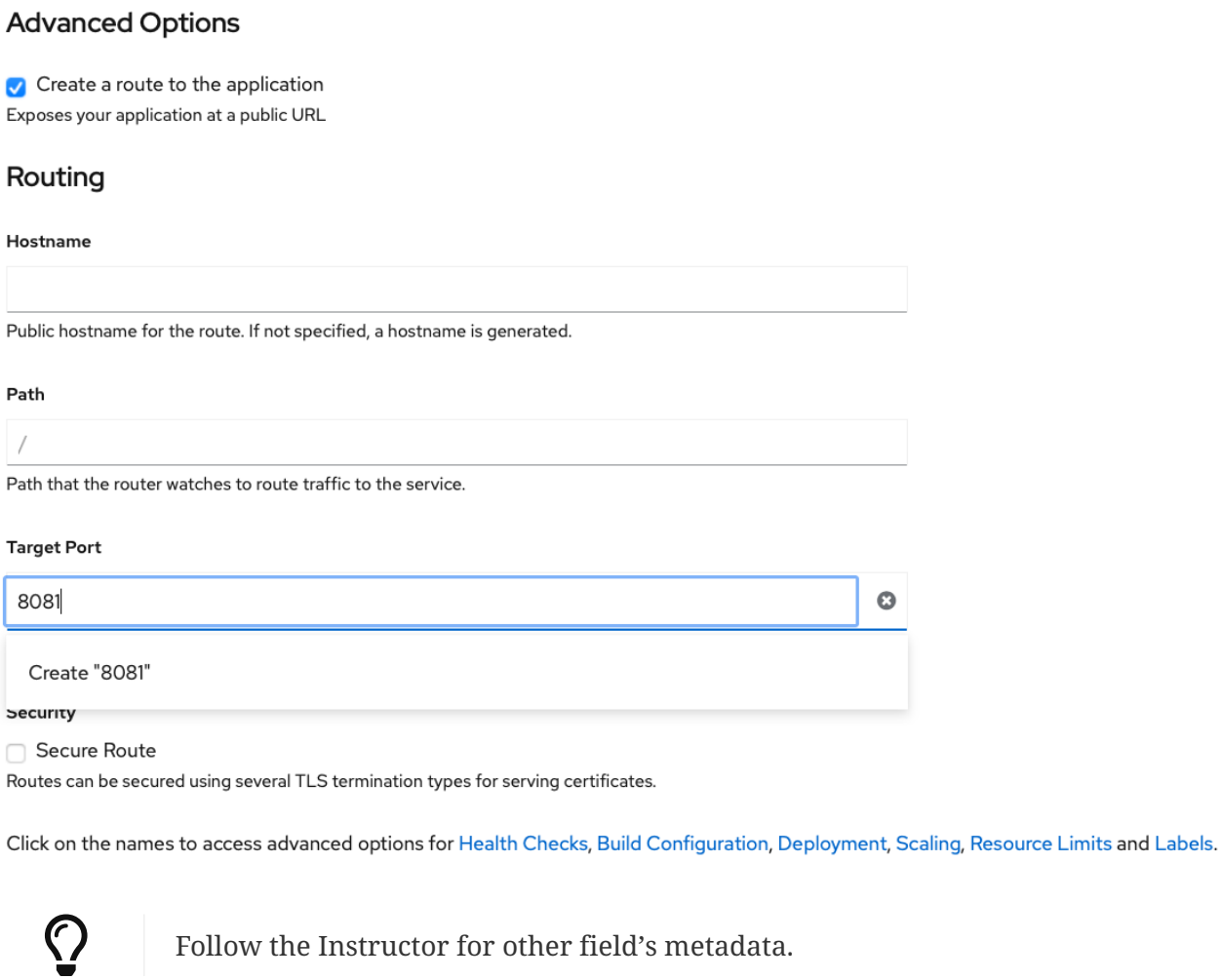
Select `Java` icon in the **Builder** section and the `openjdk-8-ubi8` in the **Builder Image Version**.



In the **General** section add the value `todo`.

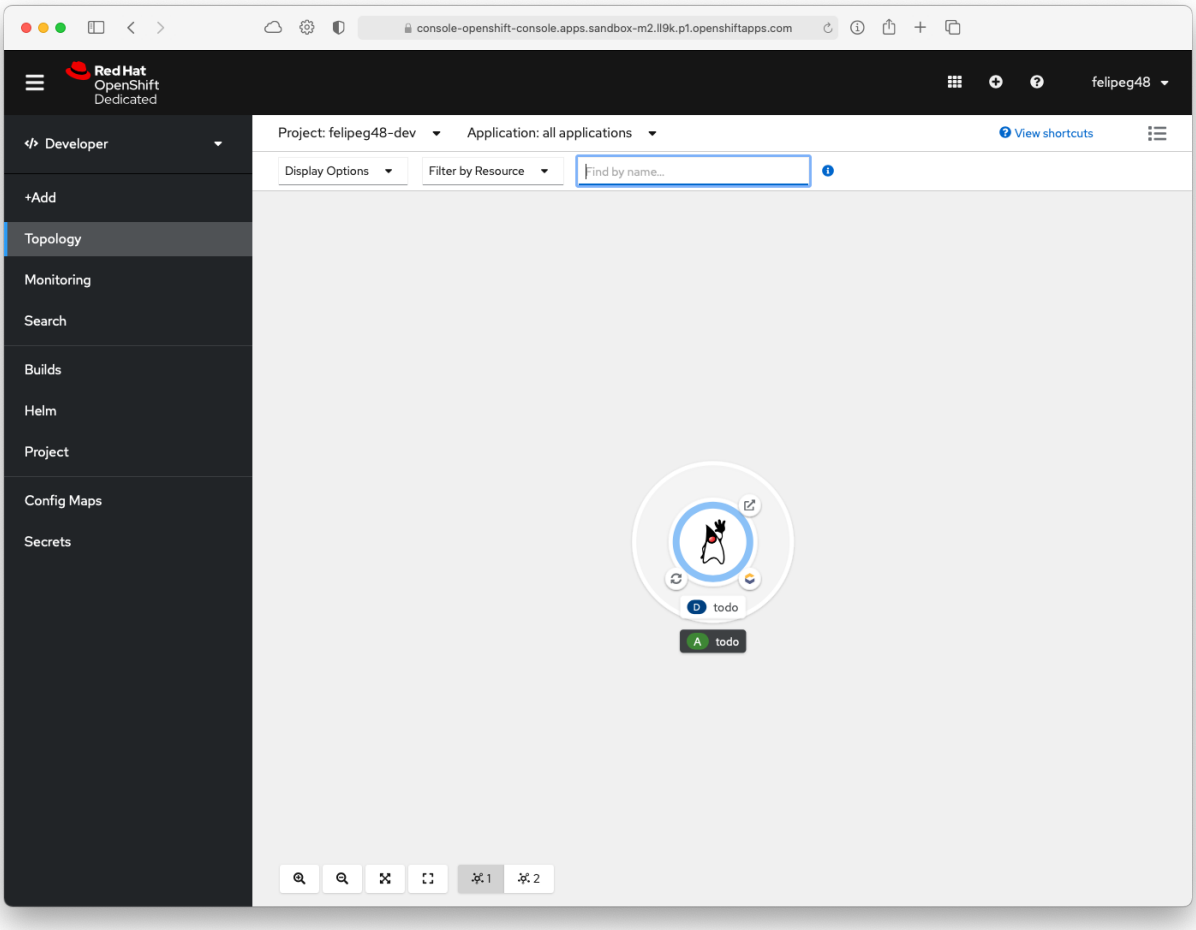


In the **Advanced Options** section click the `Routes` and create/add the port `8081`

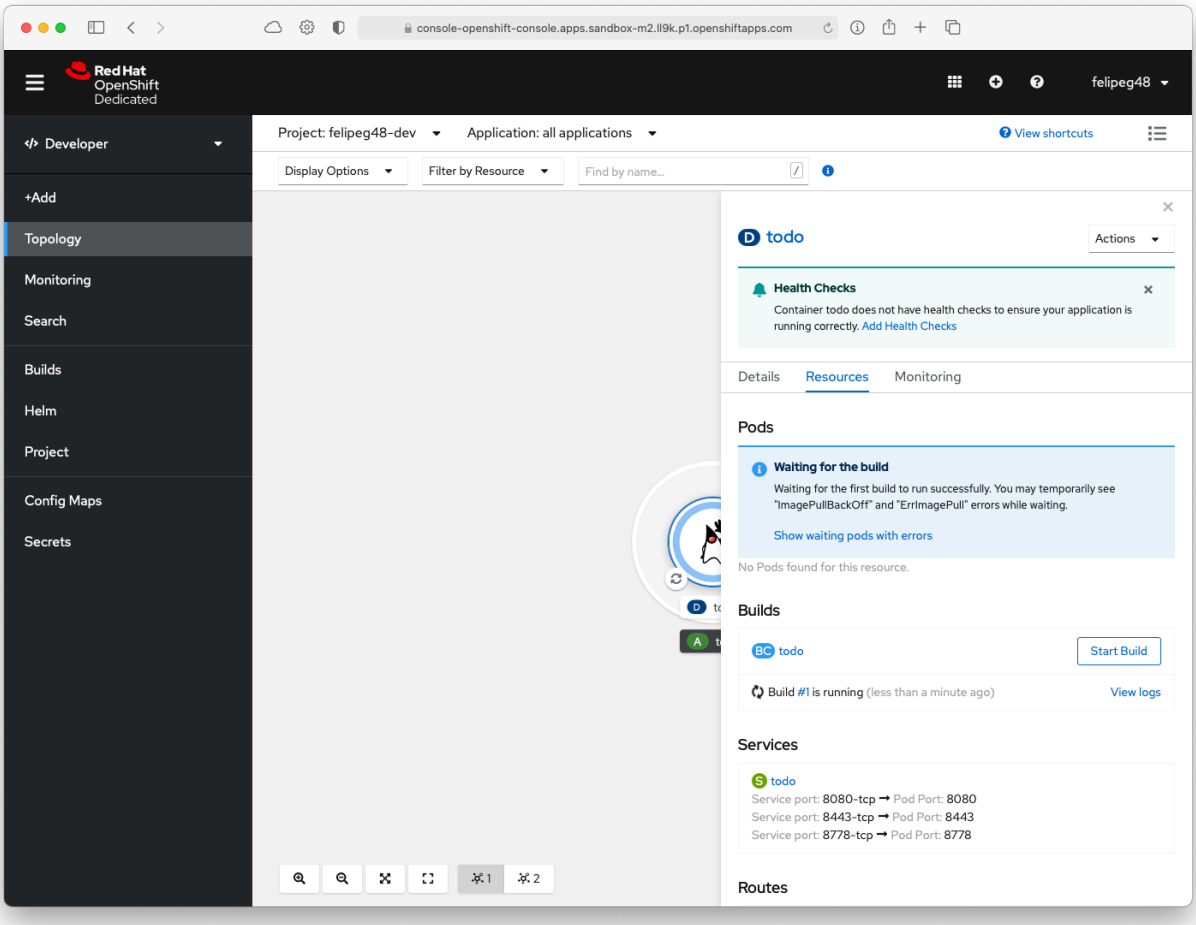


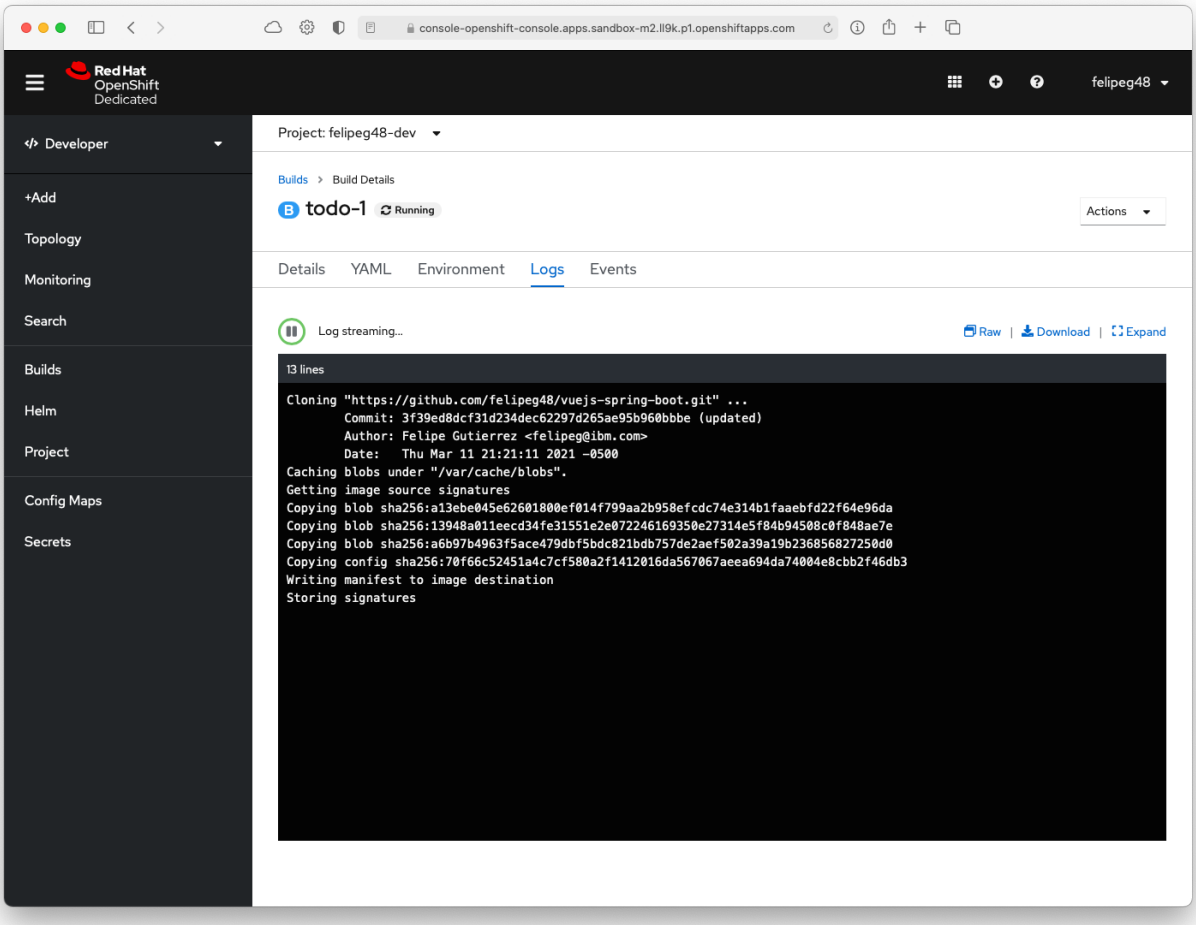
Follow the Instructor for other field's metadata.

6. You will present the **Workload** page with your app in the middle. Review the options.

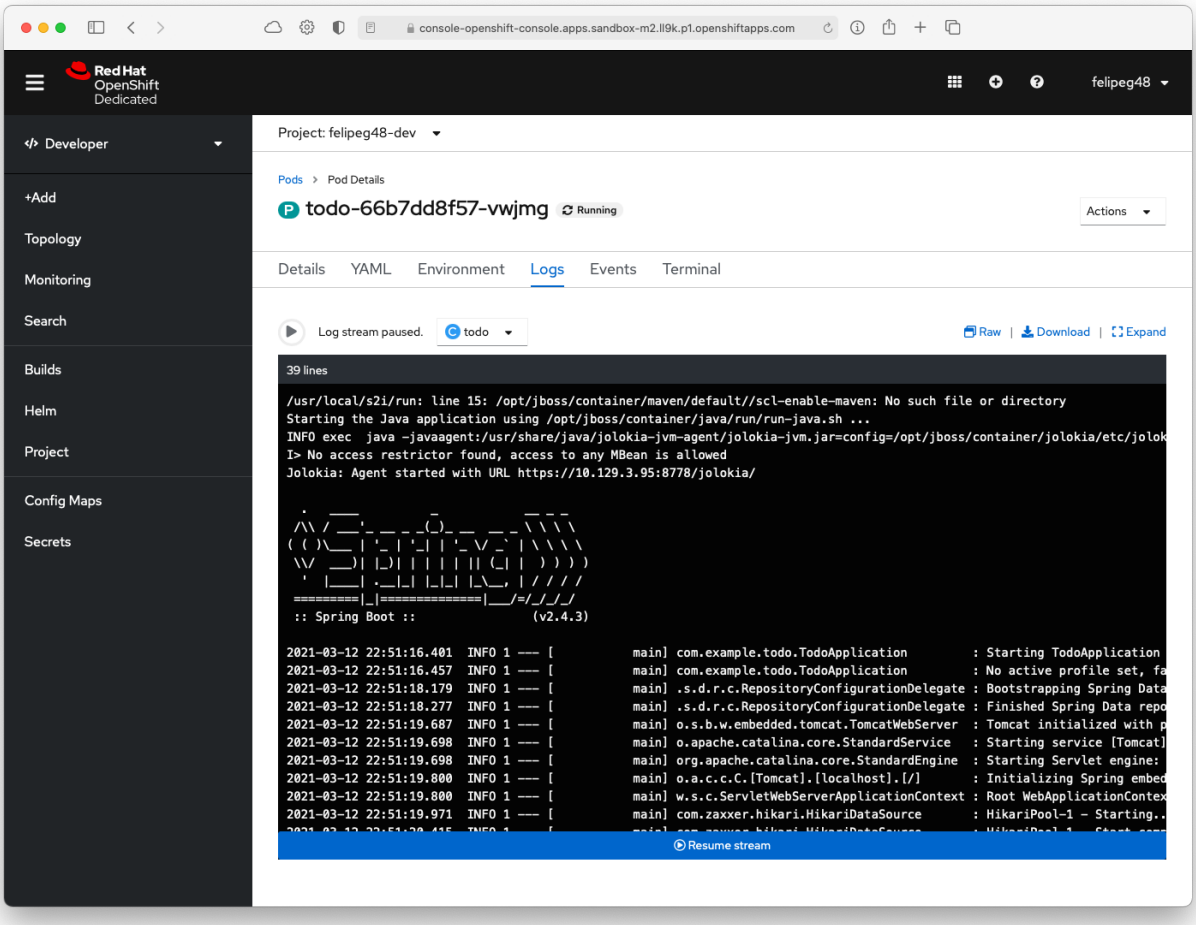


You can click in the `View logs` link in the **Builds** section


console-openshift-console.apps.sandbox-m2.ll9k.p1.openshiftapps.com
Red Hat OpenShift Dedicated
felipeg48
Developer
+Add
Topology
Monitoring
Search
Builds
Helm
Project
Config Maps
Secrets
Project: felipeg48-dev
Application: all applications
View shortcuts
Display Options
Filter by Resource
Find by name...
todo
Actions
Health Checks
Container todo does not have health checks to ensure your application is running correctly. Add Health Checks
Details
Resources
Monitoring
Pods
Waiting for the build
Waiting for the first build to run successfully. You may temporarily see "ImagePullBackOff" and "ErrImagePull" errors while waiting.
Show waiting pods with errors
No Pods found for this resource.
Builds
todo
Start Build
Build #1 is running (less than a minute ago)
View logs
Services
todo
Service port: 8080-tcp → Pod Port: 8080
Service port: 8443-tcp → Pod Port: 8443
Service port: 8778-tcp → Pod Port: 8778
Routes

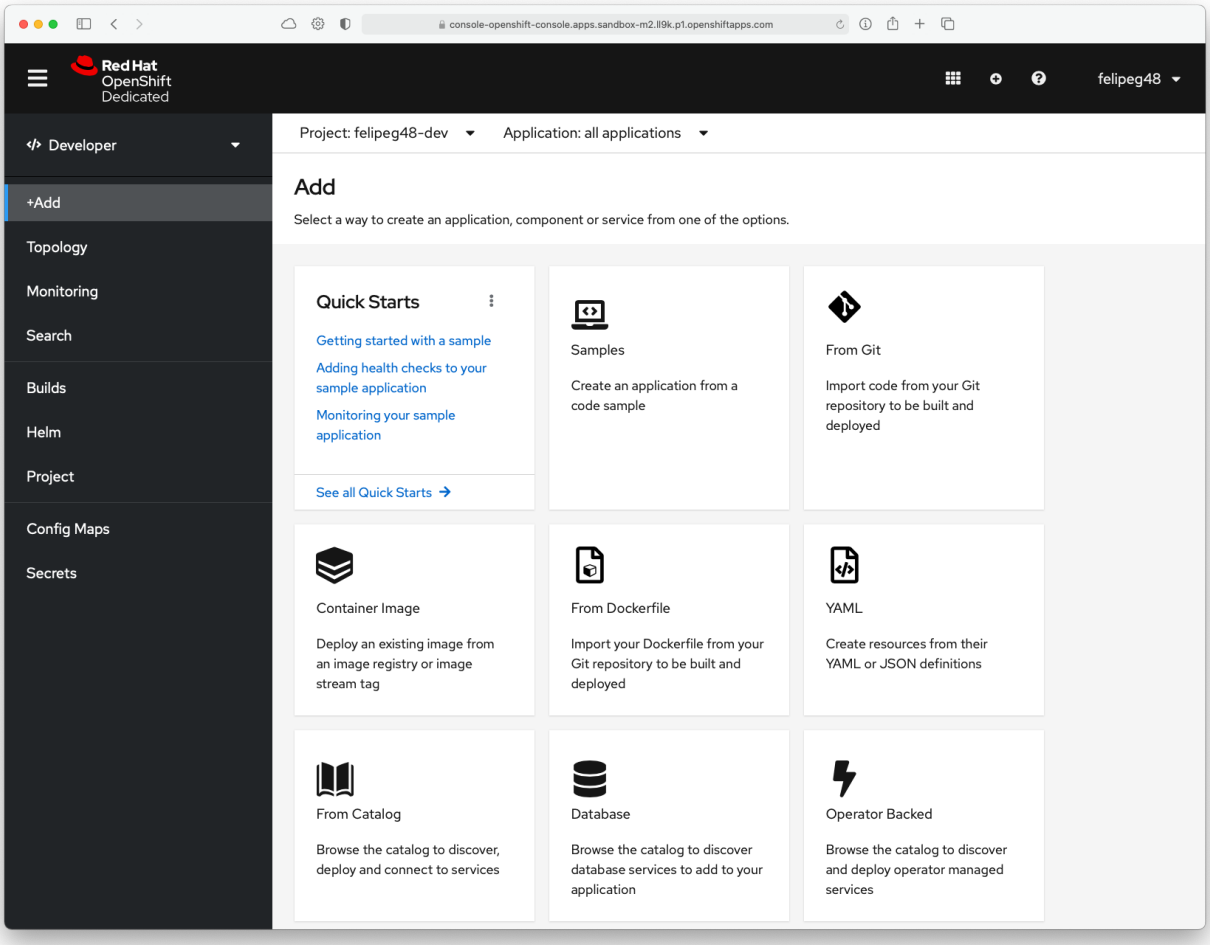


You can also see the logs of the Pods runing.

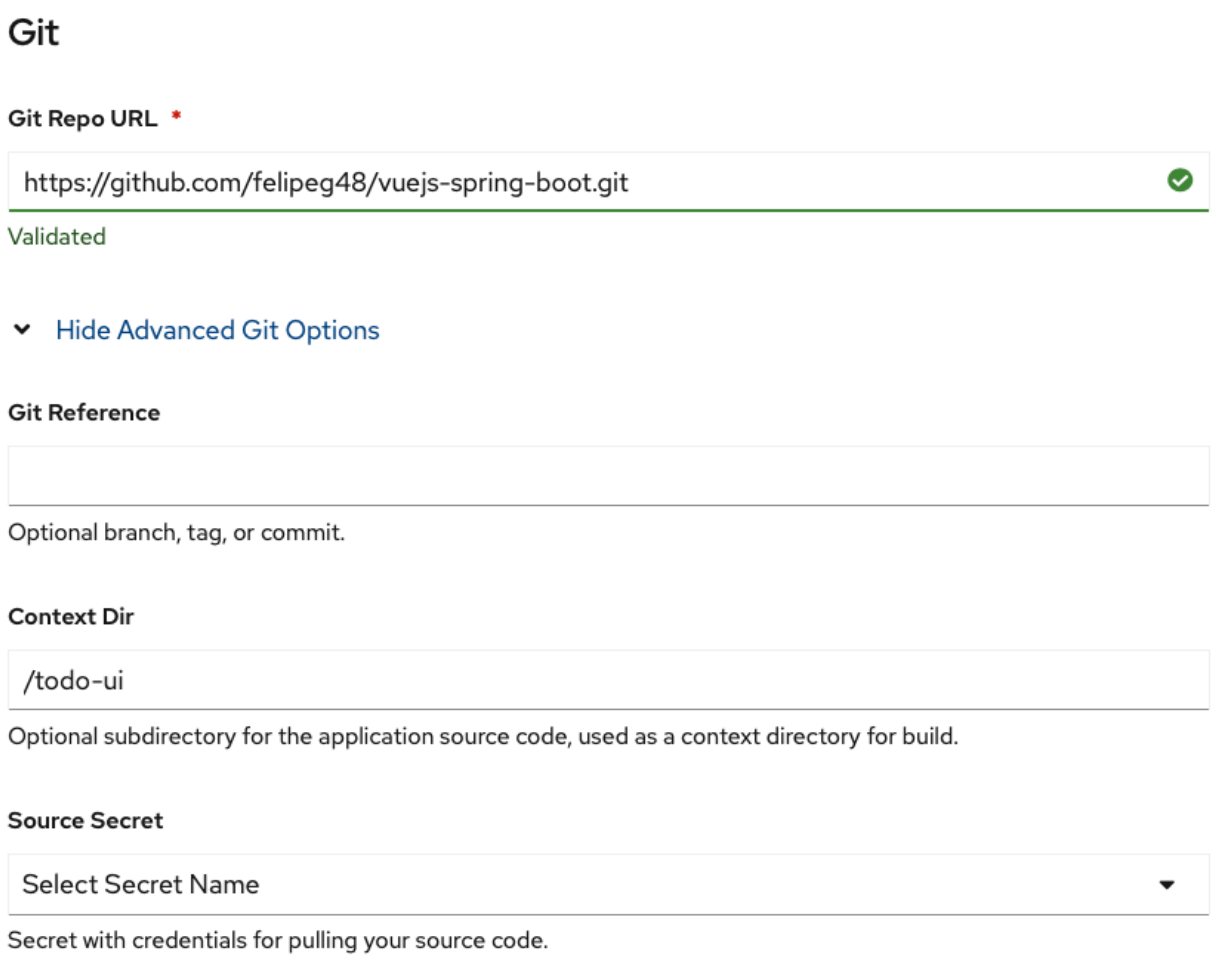


Deploy the ToDo UI App in OpenShift

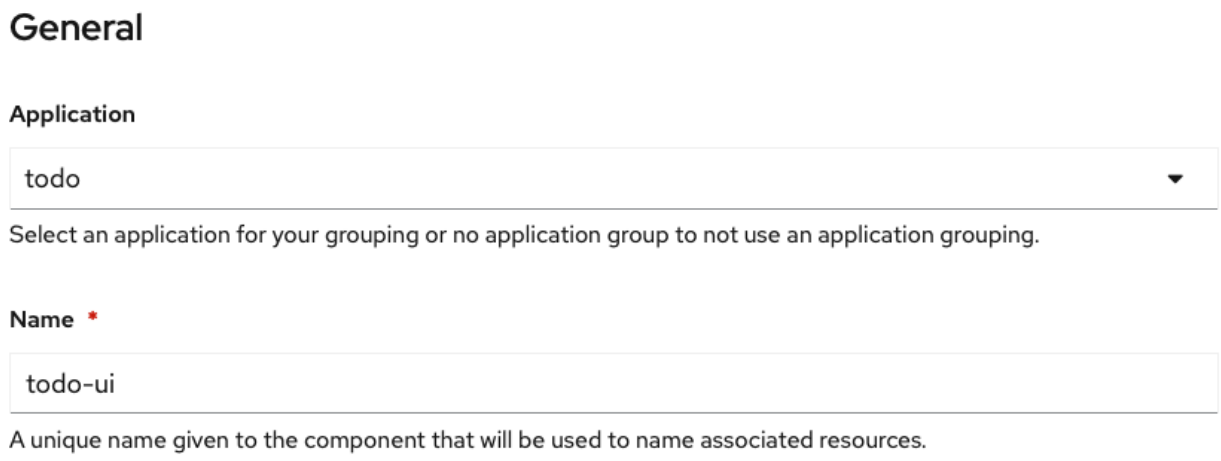
1. Open you OpenShift Cluster page.
2. Click the +**Add** button.
3. Select the **Dockerfile** square.



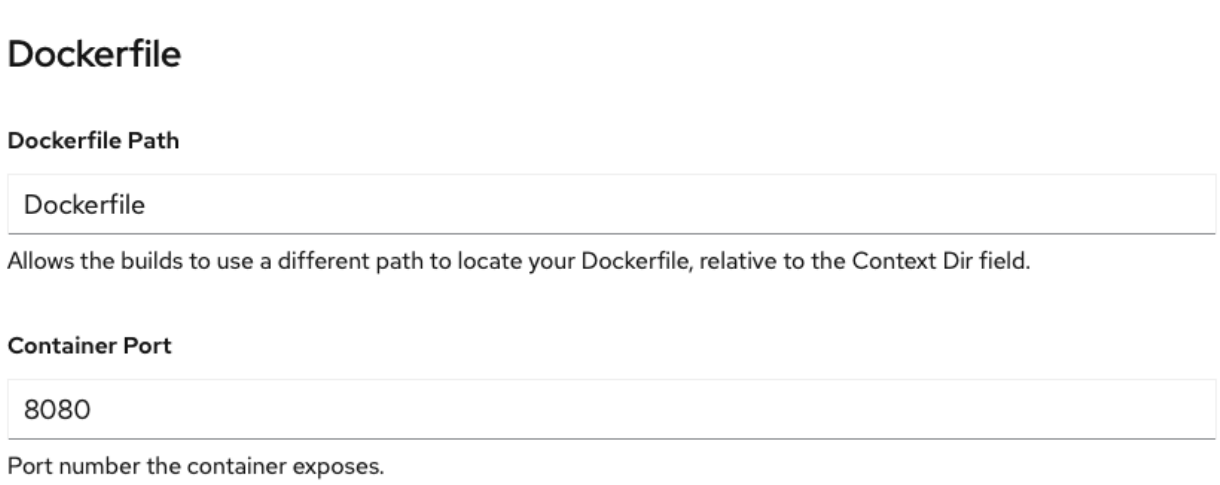
Add the `/todo-ui` in the **Context Dir** in the **Advanced Git Options** section.



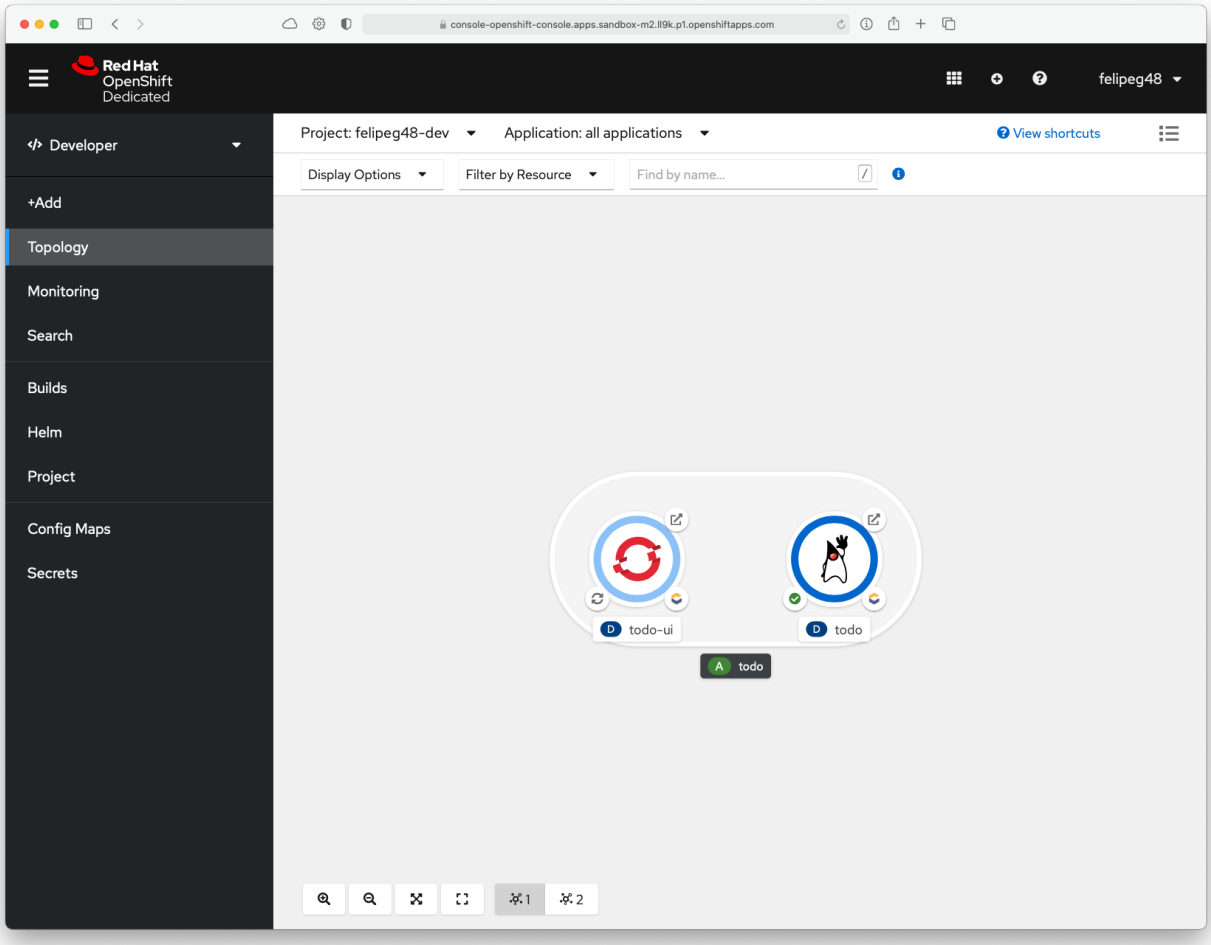
Add `todo` in the **Application** and `todo-ui` in the **Name** fields in the **General** section.



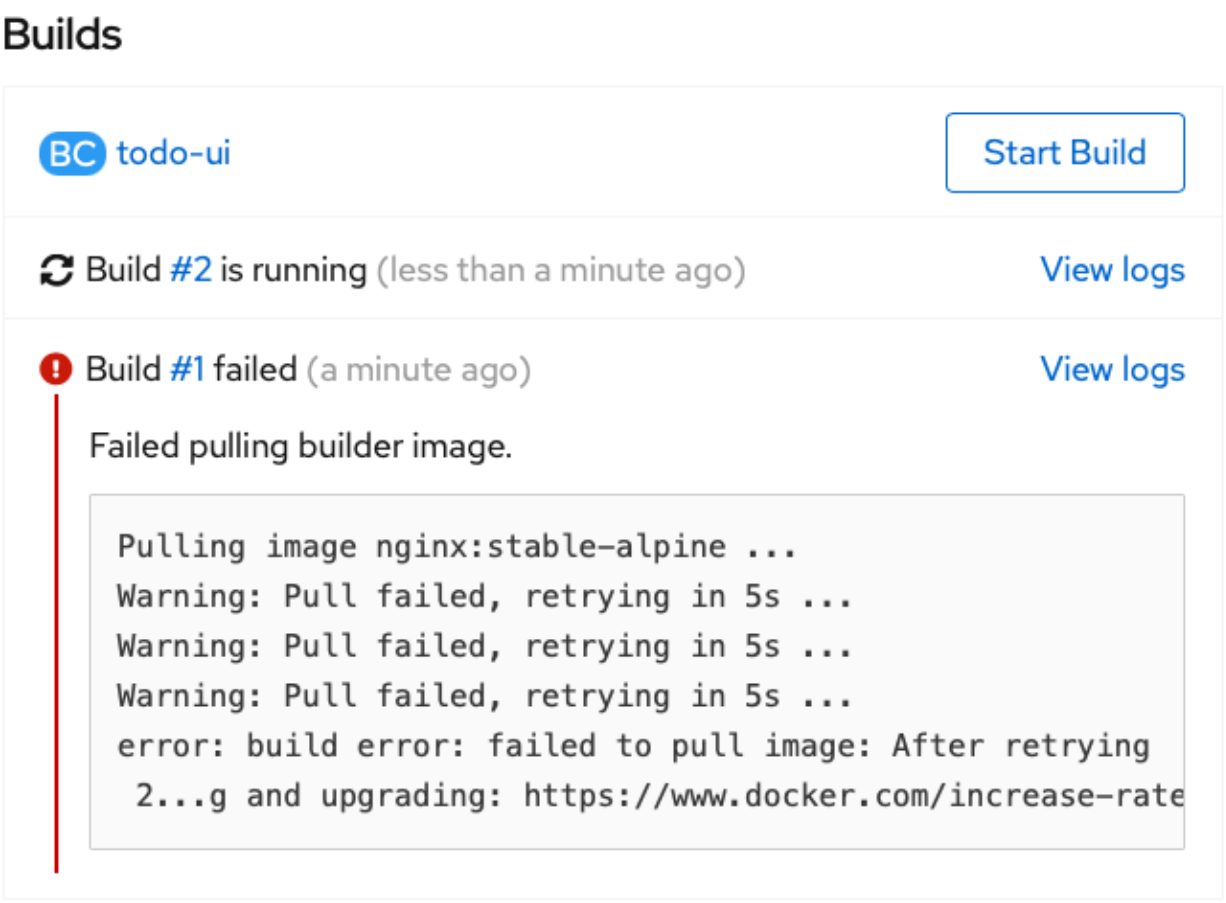
In the **Docker** section just make sure the **Dockerfile Path** is `Dockerfile` and the **Container Port** is `8080`.



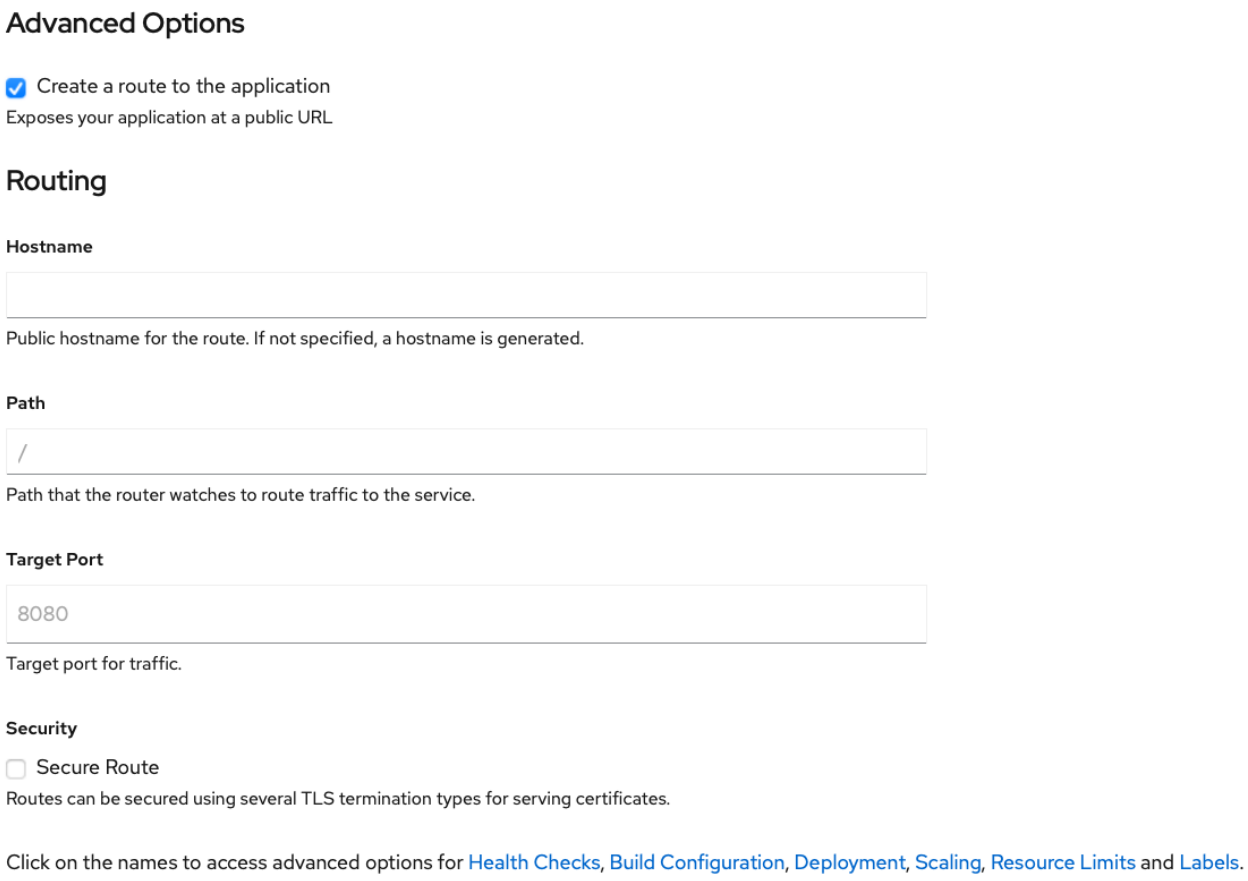
You should get the **Topology** Dashboard with the two apps.



Take a look at the logs. If fails, you can alway re-do the `Build`



Finally in the **Advaced Options** make sure the **Target Port** is set to `8080`



4. Make sure the App is working.

Challenge

- If they apps are not running as it supose to, fix them!
- The Spring Boot app is still using `H2` Database, how can we use `MySQL` what part of the code you need to change? What part of the Steps you need to re-do?