

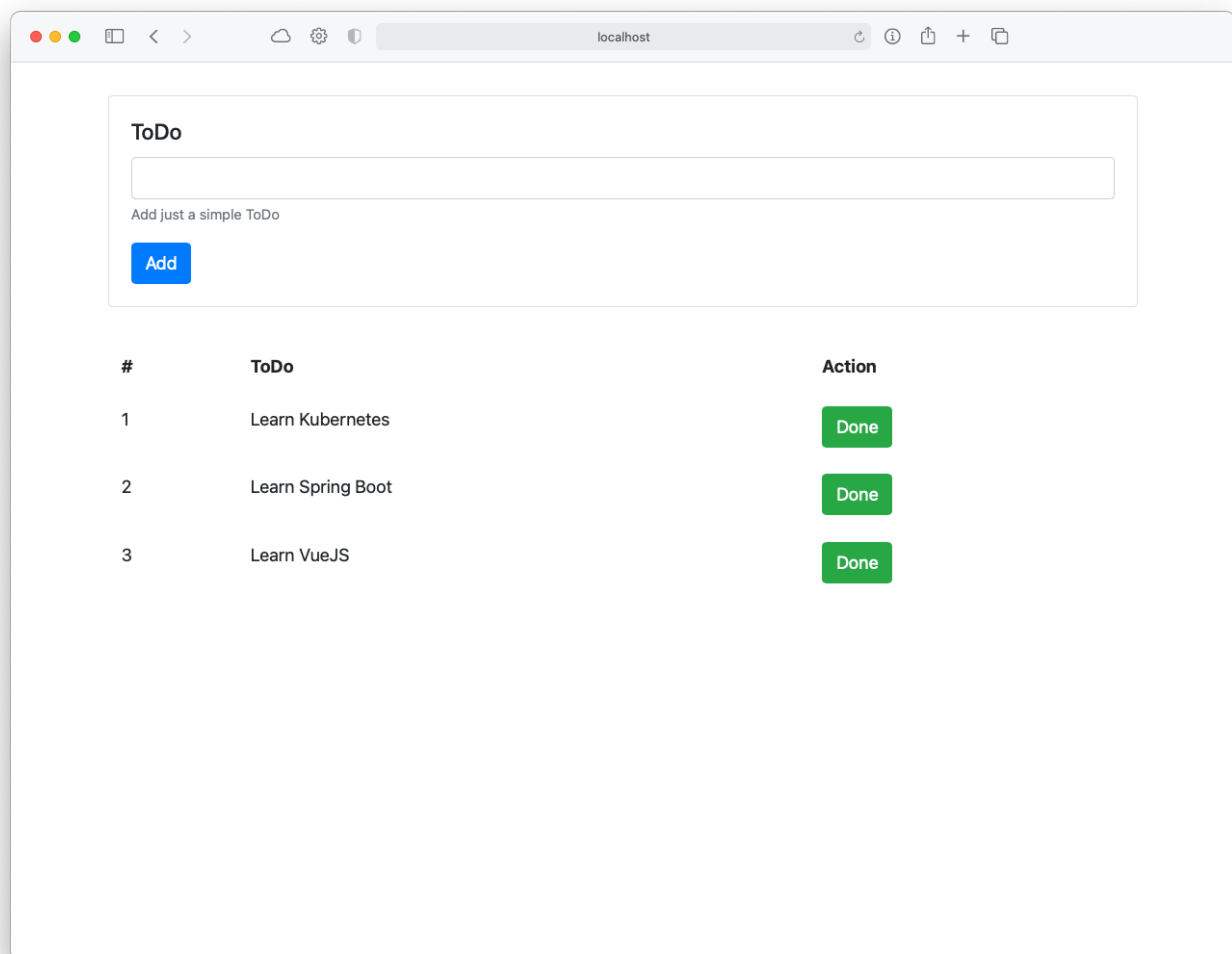
Simple ToDo App with VueJS and Spring Boot

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 Install 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/Z6nronJ3oU0FUFPcJEUQouq2+l"  
    crossorigin="anonymous">  
  
  </head>  
  <body>  
    <div id="app"></div>  
  </body>  
</html>
```

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

src/utls/event-bus.js

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

4. Create the `todo.service.js` file with the following content:

src/utils/todo.service.js

```
import Axios from 'axios';

Axios.defaults.baseURL = process.env.ROOT_API;

const RESOURCE_NAME = '/todos';

export default {
  getAll() {
    return Axios.get(RESOURCE_NAME);
  },
  get(id) {
    return Axios.get(`${RESOURCE_NAME}/${id}`);
  },
  create(data) {
    return Axios.post(RESOURCE_NAME, data);
  },
  update(id, data) {
    return Axios.put(`${RESOURCE_NAME}/${id}`, data);
  },
  delete(id) {
    return Axios.delete(`${RESOURCE_NAME}/${id}`);
  }
};
```

This script has all the external calls with **Axios**.

5. 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"'
})
```

config/dev.env.js

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



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

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

```

<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>

```

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

```

<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 { EventBus } from "../utils/event-bus";
import TodoService from "../utils/todo.service"

export default {
  name: "ToDoList",
  data: function() {
    return {
      todos: []
    }
  },
  mounted() {
    EventBus.$on("new-todo", todo => { this.newToDo(todo)});
  },
  created() {
    this.getTodos();
  },
  methods: {
    getTodos: function() {
      TodoService.getAll()
        .then(response => this.todos = response.data);
    },
  },

```

```

    newToDo: function(todo){
      TodoService.create(todo)
        .then(response => this.todos.push(response.data));
    },

    doneToDo: function(id, event){
      TodoService.delete(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>

```

8. Replace the `App.vue` with the following content:

src/App.vue

```
<template>
  <div id="app" class="container" style="margin-top: 30px;">
    <to-do/>
    <to-do-list/>
  </div>
</template>

<script>
import Todo from './components/Todo'
import TodoList from './components/TodoList.vue'

export default {
  name: 'App',
  components: {
    Todo, TodoList
  }
}

</script>

<style>

</style>
```

9. Test you app with:

```
npm run dev
```

10. Create a **Dockerfile** with the 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;"]
```

11. [OPTIONAL] Build your 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*

The screenshot shows the Spring Initializr web application in a browser. The interface is divided into several sections:

- Project:** Radio buttons for **Maven Project** (selected) and **Gradle Project**.
- Language:** Radio buttons for **Java** (selected), **Kotlin**, and **Groovy**.
- Spring Boot:** Radio buttons for versions: 2.5.0 (SNAPSHOT), 2.5.0 (M2), 2.4.4 (SNAPSHOT), **2.4.3** (selected), and 2.3.10 (SNAPSHOT). There is also a 2.3.9 option.
- Project Metadata:** Form fields for Group (com.example), Artifact (todo), Name (todo), Description (Demo project for Spring Boot), and Package name (com.example.todo).
- Dependencies:** A list of dependencies with checkboxes: **Spring Web** (WEB, checked), **Spring Data JPA** (SQL, checked), **Lombok** (DEVELOPER TOOLS, checked), and **H2 Database** (SQL, checked). There is an "ADD ..." button with a plus icon.

At the bottom, there are three buttons: **GENERATE** (with a download icon), **EXPLORE** (with a keyboard shortcut CTRL + SPACE), and **SHARE...**. Social media icons for GitHub and Twitter are visible in the bottom left corner.

2. Uncompress the ZIP and import it into your favorite IDE.
3. 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;
import org.springframework.stereotype.Repository;

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

```
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));
    }
}
```

```
package com.example.todo;

import lombok.RequiredArgsConstructor;
import org.springframework.web.bind.annotation.*;

import javax.servlet.http.HttpServletRequest;
import java.util.HashMap;
import java.util.Map;

@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;
    }
}
```

4. 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');
```

5. 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
```

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

```
<!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/Z6nronJ3oU0FUFPcjEUQouq2+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>
```

```
<!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/Z6nronJ3oU0FUfPcJEUQouq2+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>
```

7. Test you app with:

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

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

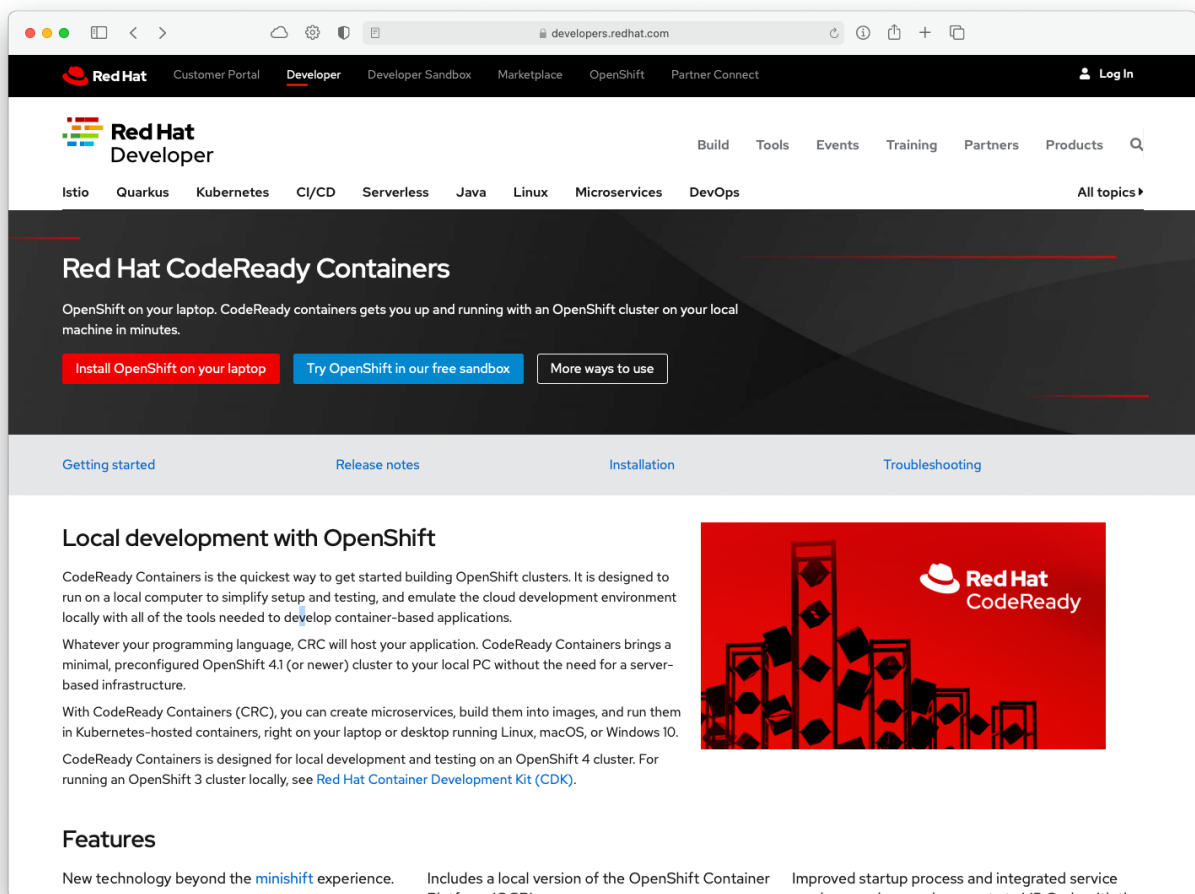
```
./mvnw spring-boot:build-image -Dspring-boot.build-image.imageName=<your-id>/todo:v1
docker run -it -p 8081:8081 --rm --name todo <your-id>/todo:v1
docker push <your-id>/todo:v1
docker tag <image-id> <your-id>/todo:latest
docker push <your-id>/todo:latest
```



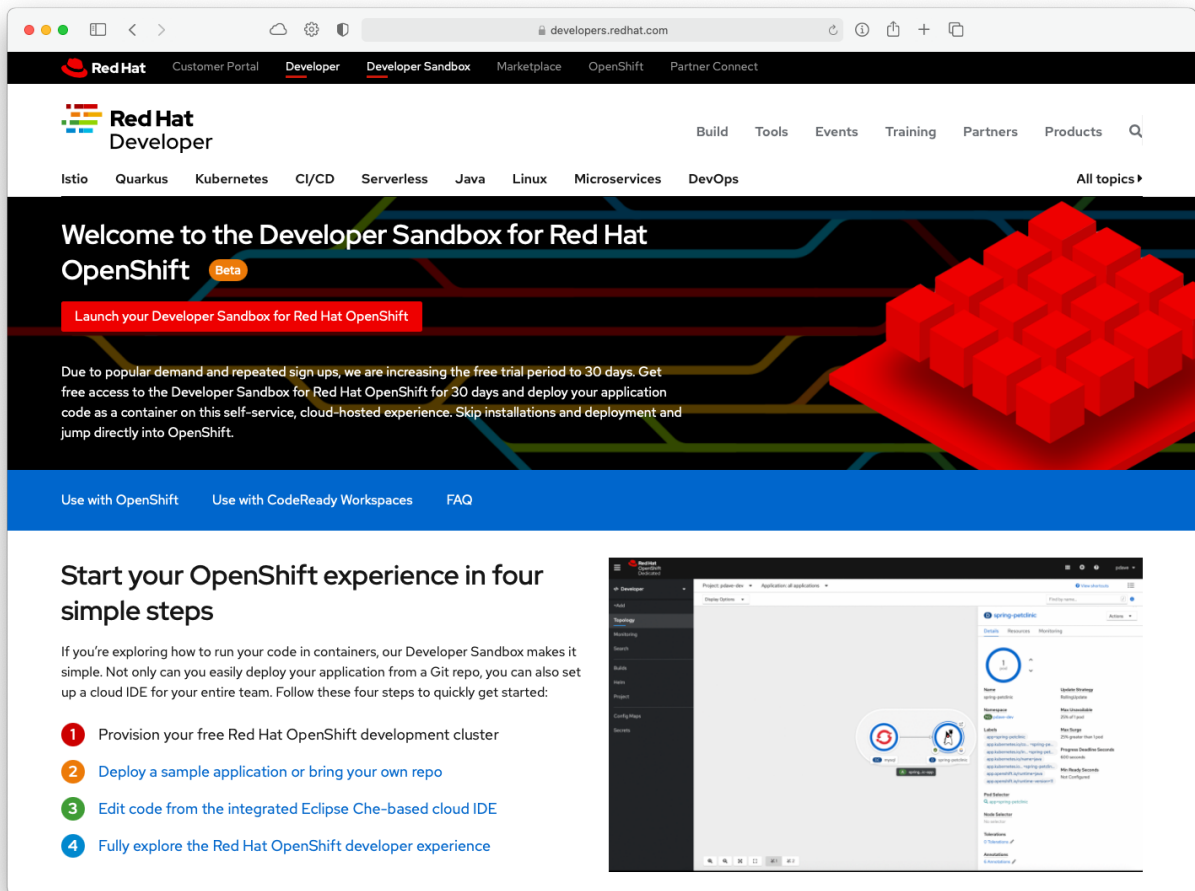
This step is **OPTIONAL**, you **DON'T** need it for deploying in OpenShift.

Deploy the ToDo App in OpenShift

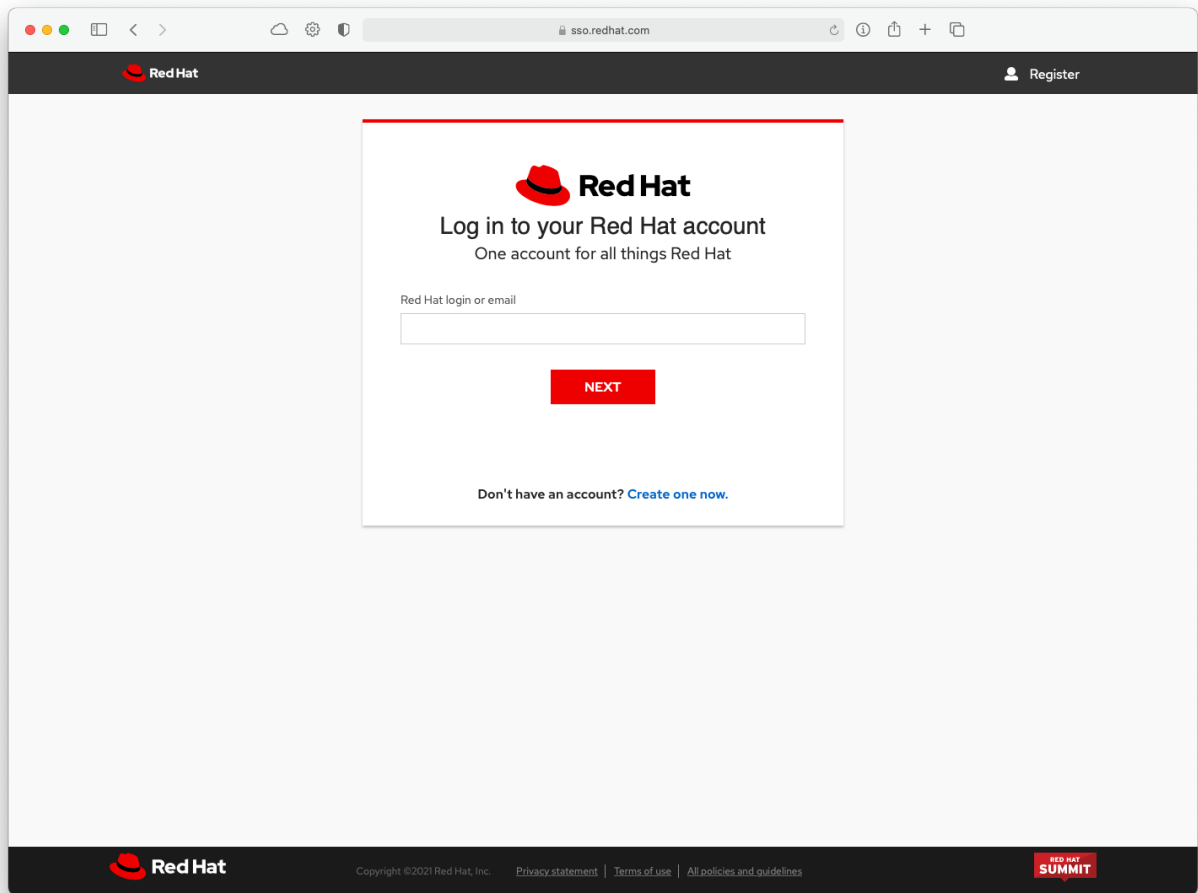
1. Open you account en RedHat con CodeReady Containers: <https://ibm.biz/BdfjCM>



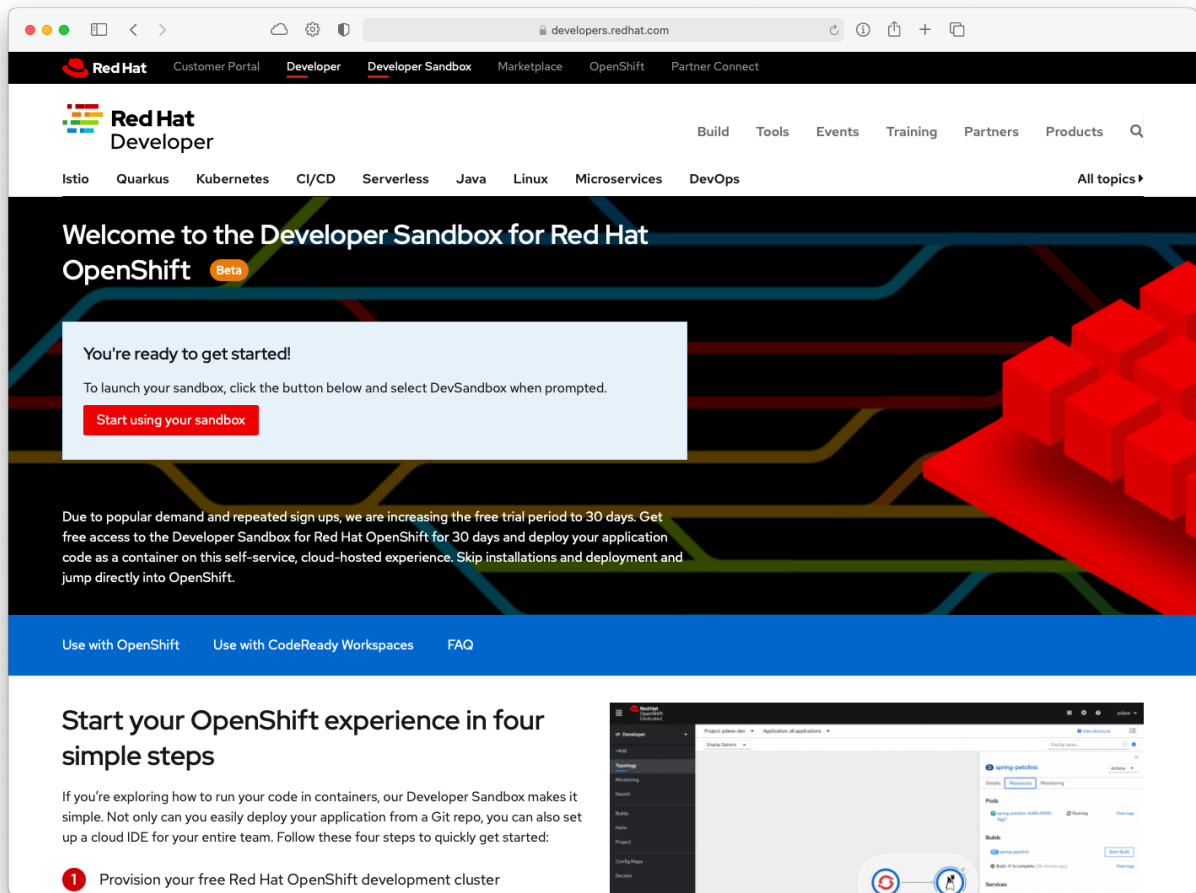
2. Go to the Developer Sandbox and Launch it.



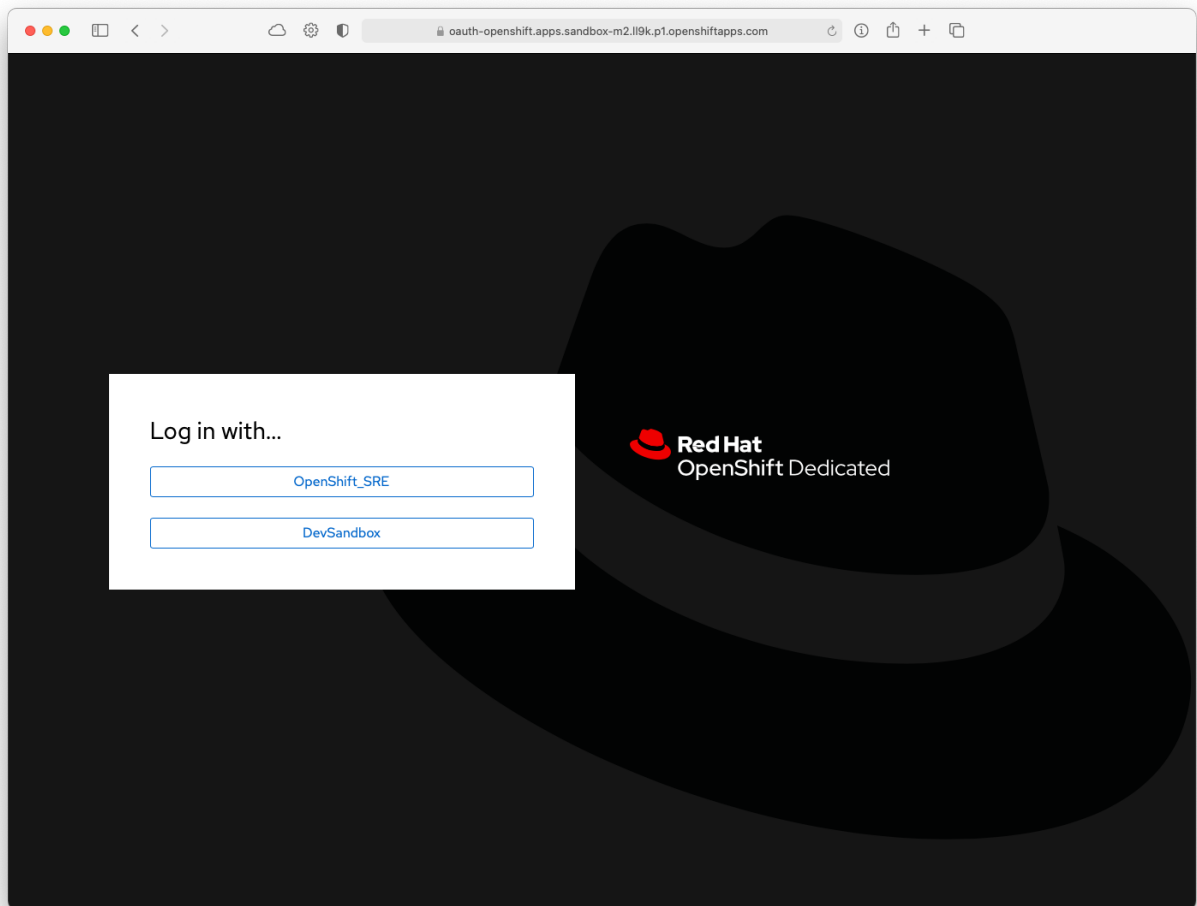
3. Create an account or Login into RedHat Portal



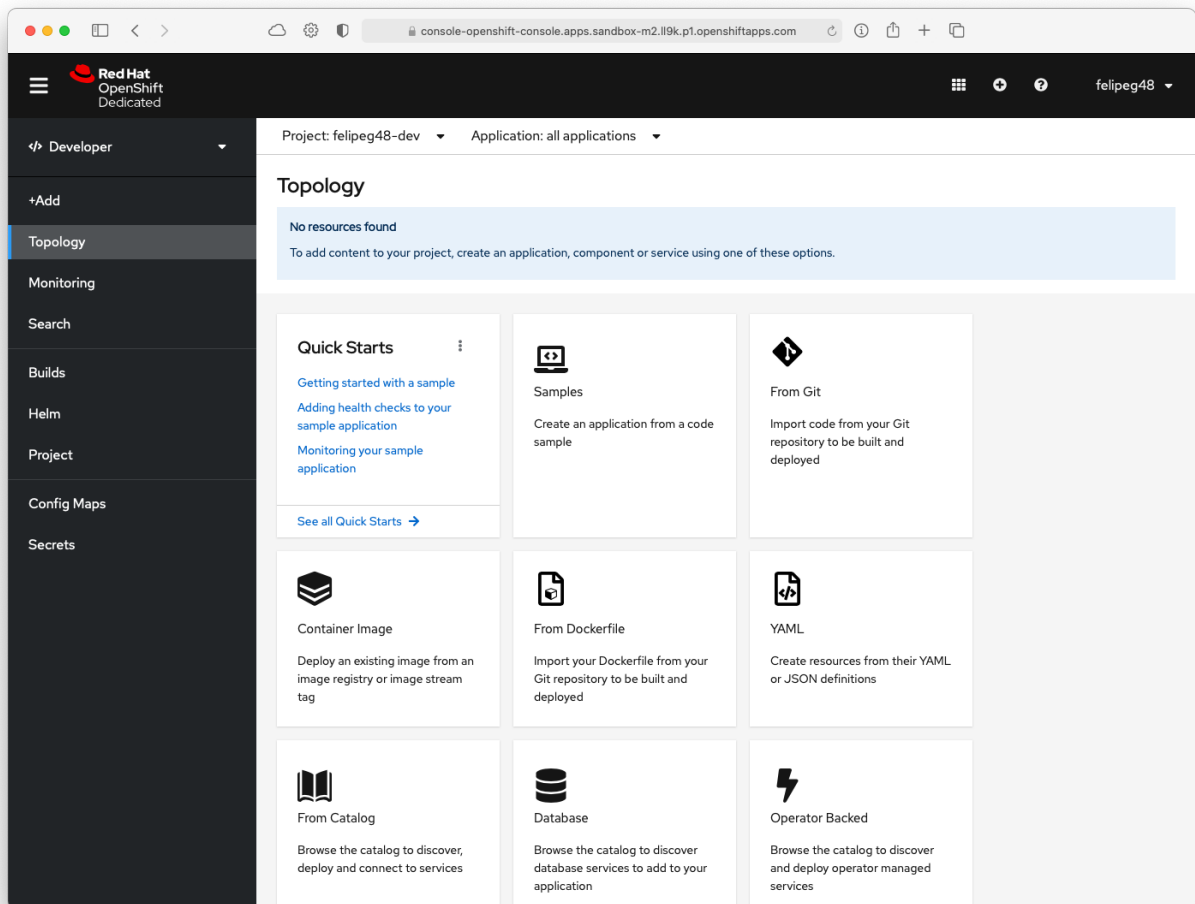
Once you login, click the **Start using your sandbox** button.

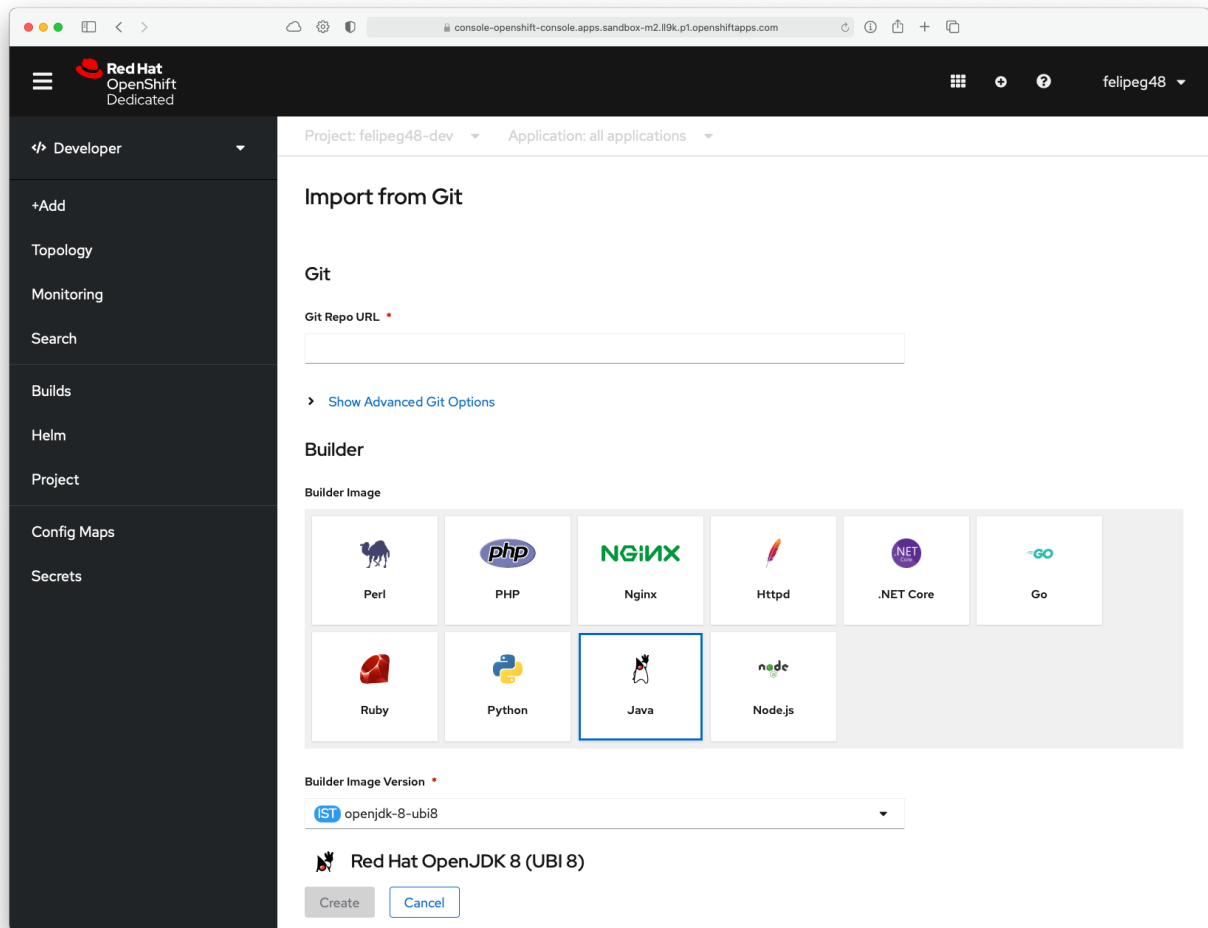


4. Select DevSandBox



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





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

Git

Git Repo URL *

https://github.com/felipeg48/vuejs-spring-boot.git



Validated

▼ Hide Advanced Git Options

Git Reference

Optional branch, tag, or commit.

Context Dir

/todo

Optional subdirectory for the application source code, used as a context directory for build.

Source Secret

Select Secret Name



Secret with credentials for pulling your source code.











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

Builder

Builder Image

⚠ Unable to detect the builder image.

Select the most appropriate one from the list to continue.

 Perl	 PHP	 Nginx	 Httpd	 .NET Core	 Go
 Ruby	 Python	 Java	 Node.js		

Builder Image Version *

IST openjdk-8-ubi8



Red Hat OpenJDK 8 (UBI 8)

BUILDER JAVA OPENJDK

Build and run Java applications using Maven and OpenJDK 8.

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

General

Application Name

A unique name given to the application grouping to label your resources.

Name *

A unique name given to the component that will be used to name associated resources.

Resources

Select the resource type to generate

☒ **Deployment**

apps/Deployment

A Deployment enables declarative updates for Pods and ReplicaSets.

☐ **Deployment Config**

apps.openshift.io/DeploymentConfig

A Deployment Config defines the template for a pod and manages deploying new images or configuration changes.

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

Advanced Options

☒ Create a route to the application
Exposes your application at a public URL

Routing

Hostname

Public hostname for the route. If not specified, a hostname is generated.

Path

Path that the router watches to route traffic to the service.

Target Port

Create "8081"

Security

☐ Secure Route

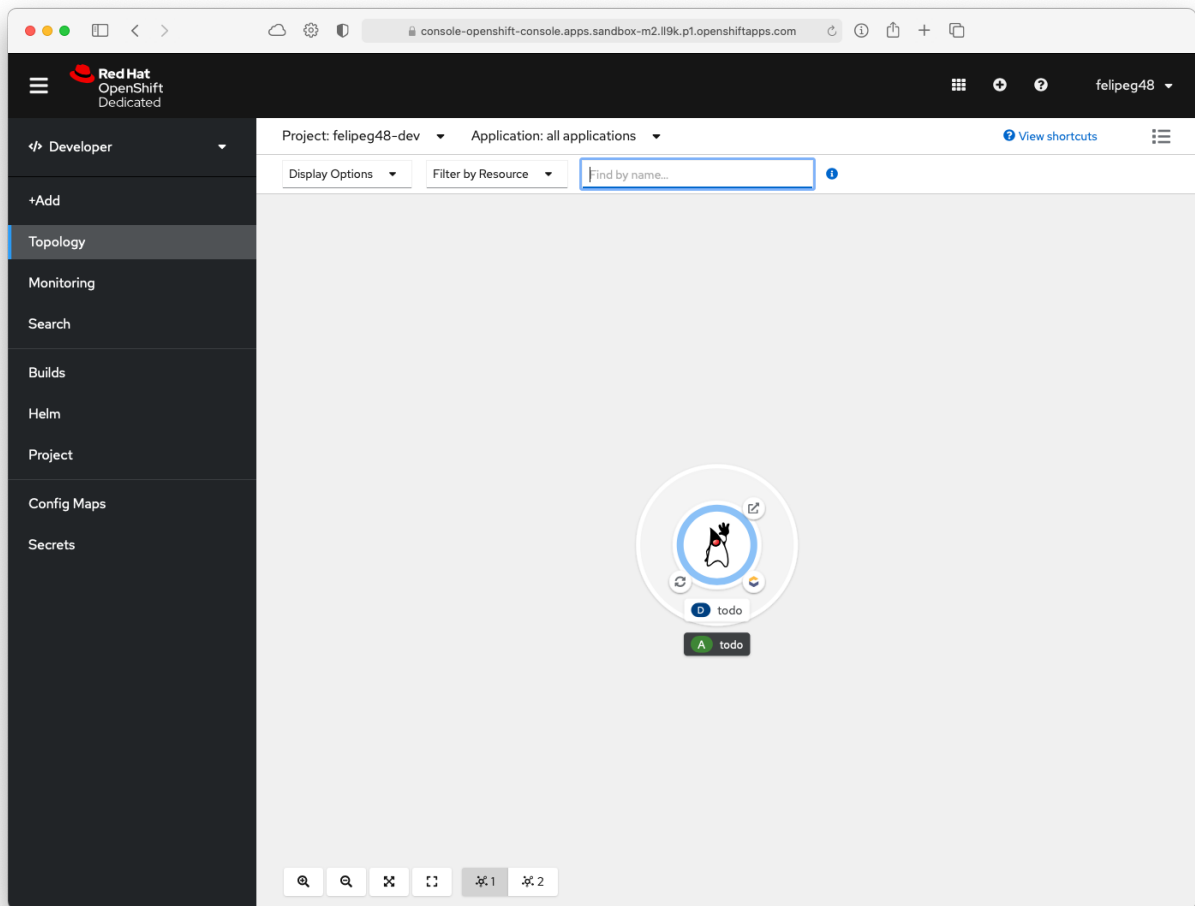
Routes can be secured using several TLS termination types for serving certificates.

Click on the names to access advanced options for [Health Checks](#), [Build Configuration](#), [Deployment](#), [Scaling](#), [Resource Limits](#) and [Labels](#).



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

Red Hat OpenShift Dedicated

Project: felipe48-dev Application: all applications View shortcuts

Display Options Filter by Resource Find by name...

Developer +Add Topology Monitoring Search Builds Helm Project Config Maps Secrets

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

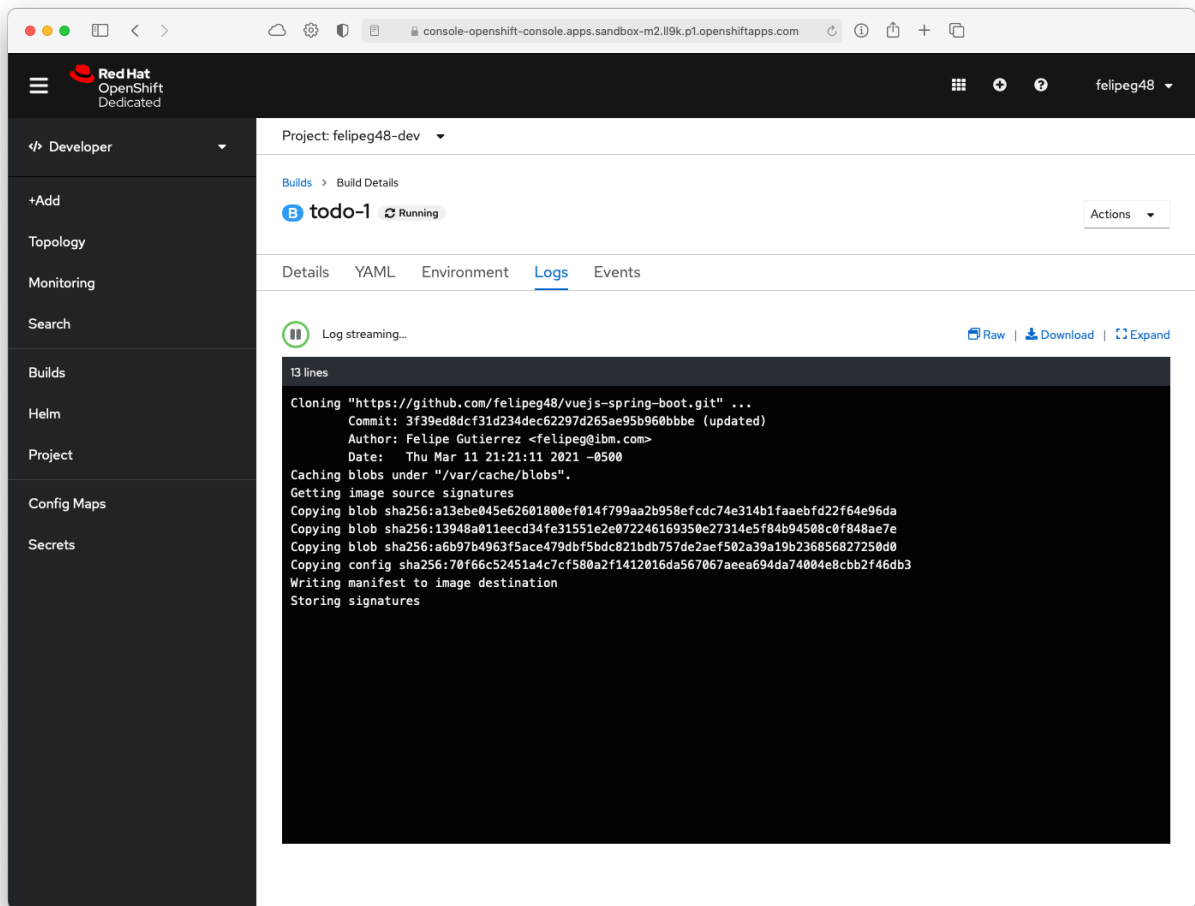
Build #1 is running (less than a minute ago) View logs

Services

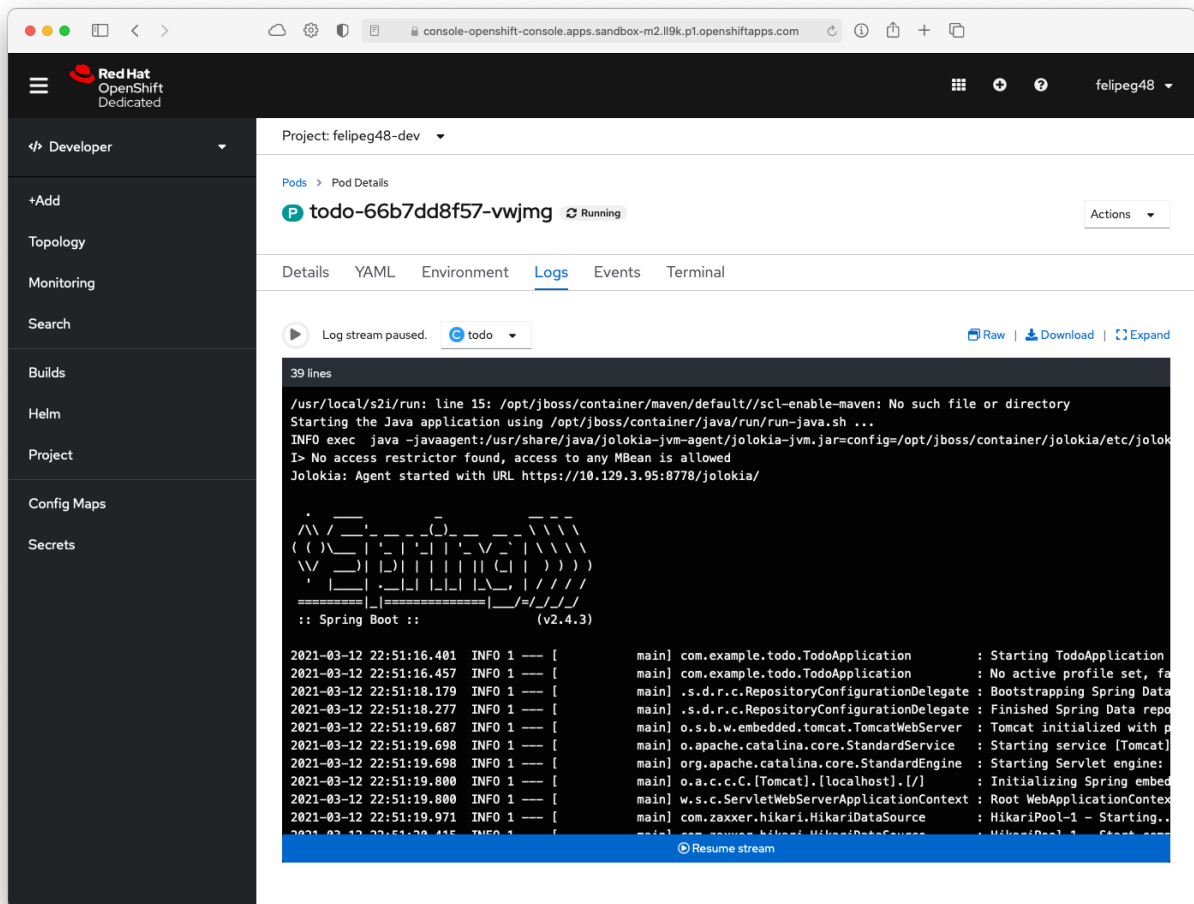
todo

Service port: 8080-tcp → Pod Port: 8080
Service port: 8443-tcp → Pod Port: 8443
Service port: 8778-tcp → Pod Port: 8778

Routes

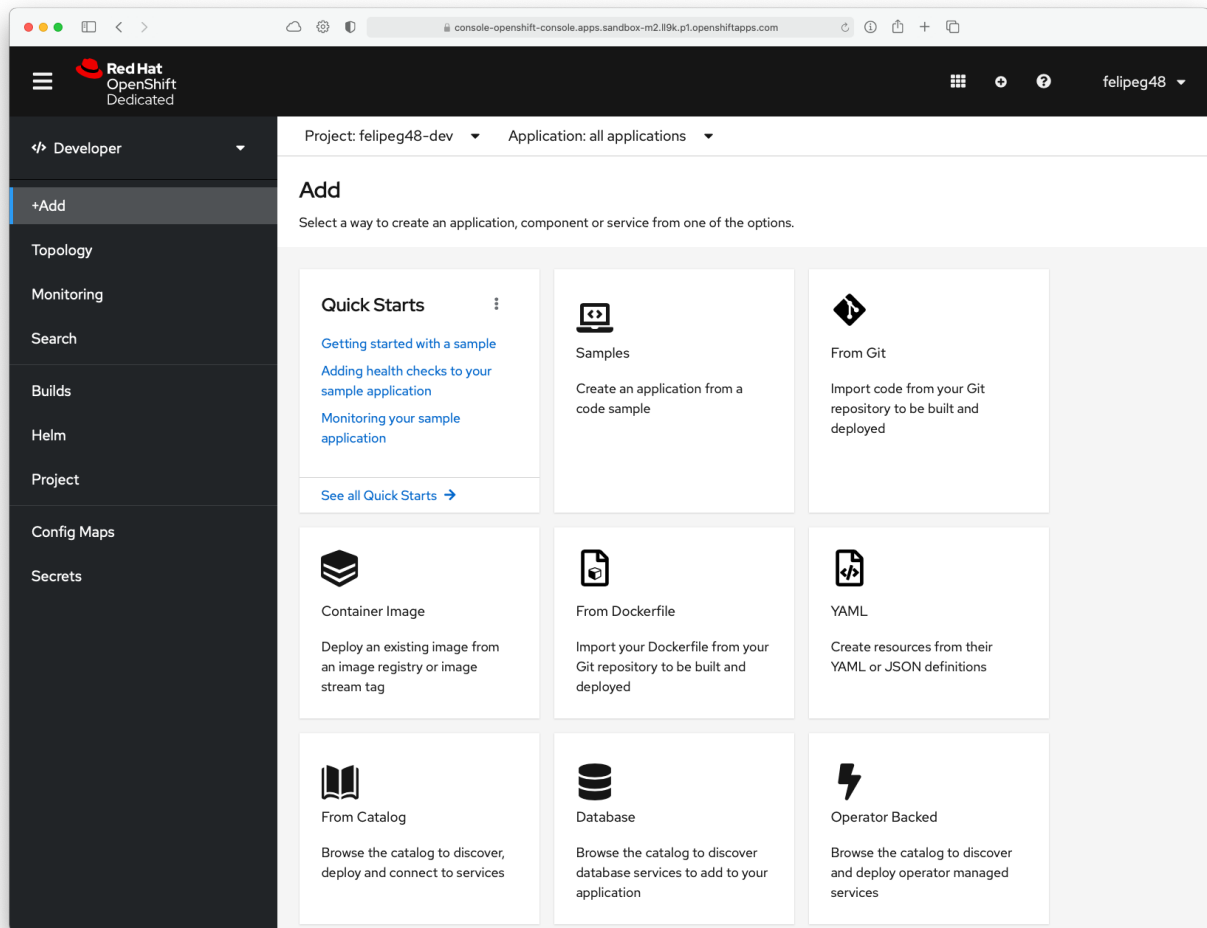


You can also see the logs of the Pods running.



Deploy the ToDo UI App in OpenShift

1. Open your 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.

Git

Git Repo URL *

https://github.com/felipeg48/vuejs-spring-boot.git



Validated

▼ Hide Advanced Git Options

Git Reference

Optional branch, tag, or commit.

Context Dir

/todo-ui

Optional subdirectory for the application source code, used as a context directory for build.

Source Secret

Select Secret Name



Secret with credentials for pulling your source code.

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

General

Application

todo



Select an application for your grouping or no application group to not use an application grouping.

Name *

todo-ui

A unique name given to the component that will be used to name associated resources.

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

Dockerfile

Dockerfile Path

Dockerfile

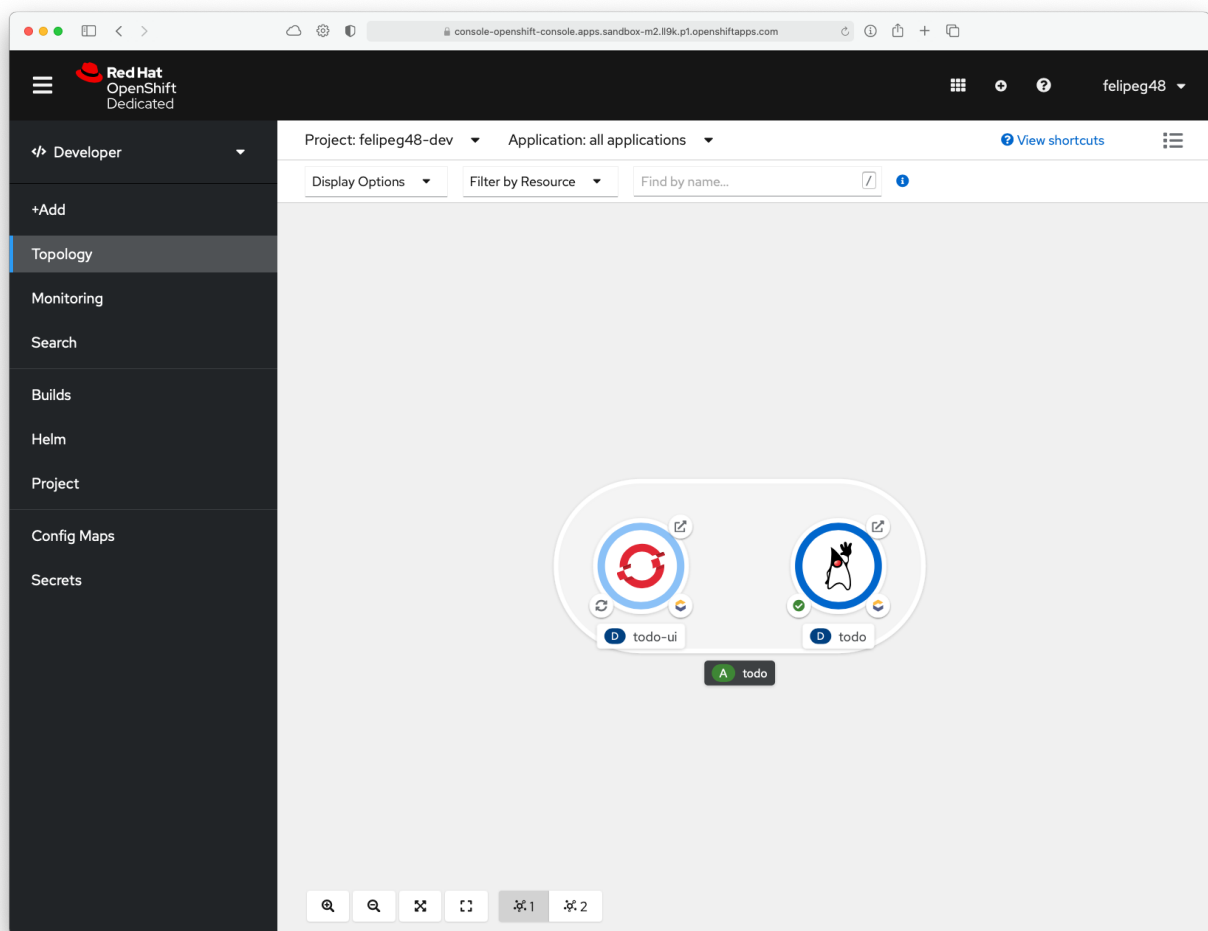
Allows the builds to use a different path to locate your Dockerfile, relative to the Context Dir field.

Container Port

8080


Port number the container exposes.

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





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

Builds

 todo-ui

Start Build

 Build #2 is running (less than a minute ago) [View logs](#)

 Build #1 failed (a minute ago) [View logs](#)

Failed pulling builder image.

```
Pulling image nginx:stable-alpine ...
Warning: Pull failed, retrying in 5s ...
Warning: Pull failed, retrying in 5s ...
Warning: Pull failed, retrying in 5s ...
error: build error: failed to pull image: After retrying
2...g and upgrading: https://www.docker.com/increase-rate
```

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

Advanced Options

☒ Create a route to the application
Exposes your application at a public URL

Routing

Hostname

Public hostname for the route. If not specified, a hostname is generated.

Path

Path that the router watches to route traffic to the service.

Target Port

Target port for traffic.

Security

☐ Secure Route

Routes can be secured using several TLS termination types for serving certificates.

Click on the names to access advanced options for [Health Checks](#), [Build Configuration](#), [Deployment](#), [Scaling](#), [Resource Limits](#) and [Labels](#).

4. Make sure the App is working.

Challenge

- If they apps are not running as it suppose 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?