

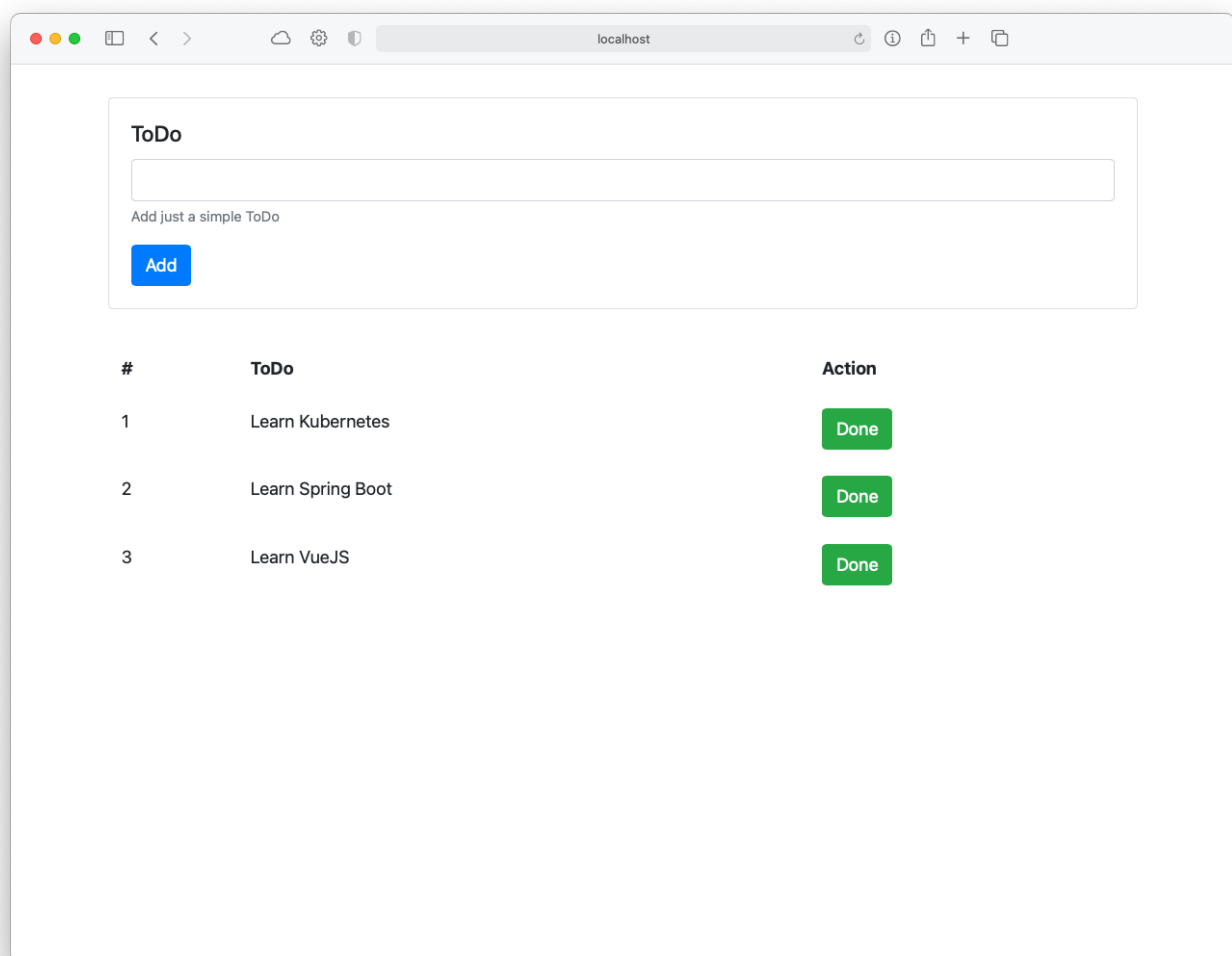
Simple ToDo App with VueJS and Spring Boot

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 `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>

```

4. 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 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/todos"'
}
```

7. Test you app with:

```
npm run dev
```

8. 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 nginx:stable-alpine as production-stage
COPY --from=build-stage /app/dist /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

9. [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:** ☒ Maven Project, ☐ Gradle Project
- Language:** ☒ Java, ☐ Kotlin, ☐ Groovy
- Spring Boot:** ☐ 2.5.0 (SNAPSHOT), ☐ 2.5.0 (M2), ☐ 2.4.4 (SNAPSHOT), ☒ 2.4.3, ☐ 2.3.10 (SNAPSHOT), ☐ 2.3.9
- Project Metadata:**
 - Group: com.example
 - Artifact: todo
 - Name: todo
 - Description: Demo project for Spring Boot
 - Package name: com.example.todo
- Dependencies:** A list of dependencies with checkboxes and descriptions:
 - Spring Web** (WEB): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
 - Spring Data JPA** (SQL): Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
 - Lombok** (DEVELOPER TOOLS): Java annotation library which helps to reduce boilerplate code.
 - H2 Database** (SQL): Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

At the bottom, there are three buttons: **GENERATE** (with a download icon), **EXPLORE** (with a keyboard shortcut CTRL + SPACE), and **SHARE...**

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

```
<!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+1"
    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:

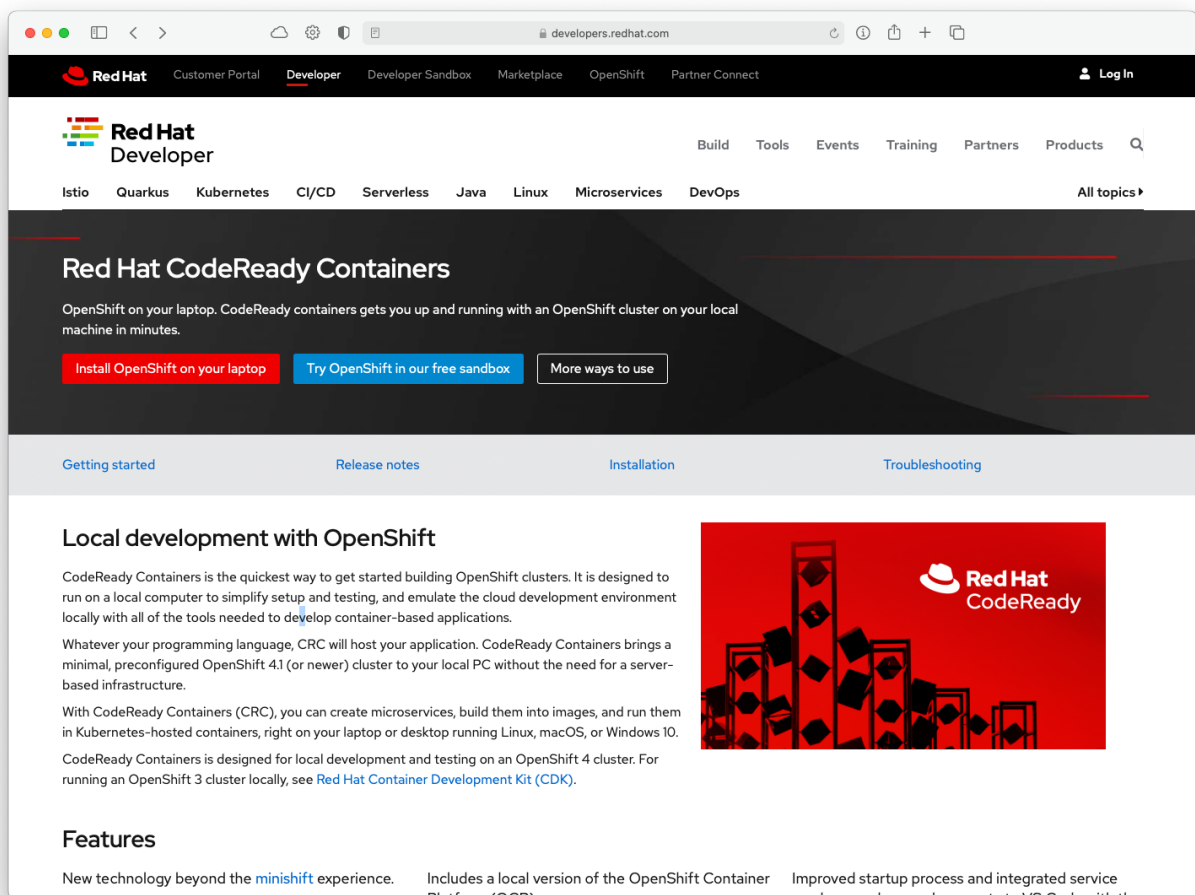
```
./mvnw spring-boot:run
```

7. 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
```

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.

The screenshot shows the Red Hat Developer website. The header includes the Red Hat logo and navigation links for Customer Portal, Developer, Developer Sandbox, Marketplace, OpenShift, and Partner Connect. Below the header, there's a secondary navigation bar with links for Istio, Quarkus, Kubernetes, CI/CD, Serverless, Java, Linux, Microservices, DevOps, and All topics. The main content area features a large banner for the 'Welcome to the Developer Sandbox for Red Hat OpenShift' with a 'Beta' badge. A prominent red button says 'Launch your Developer Sandbox for Red Hat OpenShift'. Below this, a paragraph explains that due to popular demand, the free trial period is increased to 30 days. A blue bar contains links for 'Use with OpenShift', 'Use with CodeReady Workspaces', and 'FAQ'. The section 'Start your OpenShift experience in four simple steps' lists four steps: 1. Provision your free Red Hat OpenShift development cluster, 2. Deploy a sample application or bring your own repo, 3. Edit code from the integrated Eclipse Che-based cloud IDE, and 4. Fully explore the Red Hat OpenShift developer experience. To the right of the steps is a screenshot of the Red Hat OpenShift console showing a 'spring-petclinic' application being deployed.

Red Hat Developer

Build Tools Events Training Partners Products

Istio Quarkus Kubernetes CI/CD Serverless Java Linux Microservices DevOps All topics

Welcome to the Developer Sandbox for Red Hat OpenShift Beta

[Launch your Developer Sandbox for Red Hat OpenShift](#)

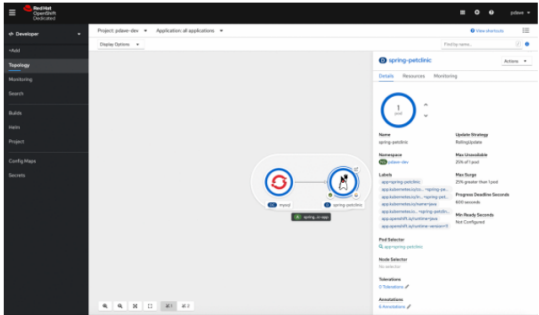
Due to popular demand and repeated sign ups, we are increasing the free trial period to 30 days. Get free access to the Developer Sandbox for Red Hat OpenShift for 30 days and deploy your application code as a container on this self-service, cloud-hosted experience. Skip installations and deployment and jump directly into OpenShift.

[Use with OpenShift](#) [Use with CodeReady Workspaces](#) [FAQ](#)

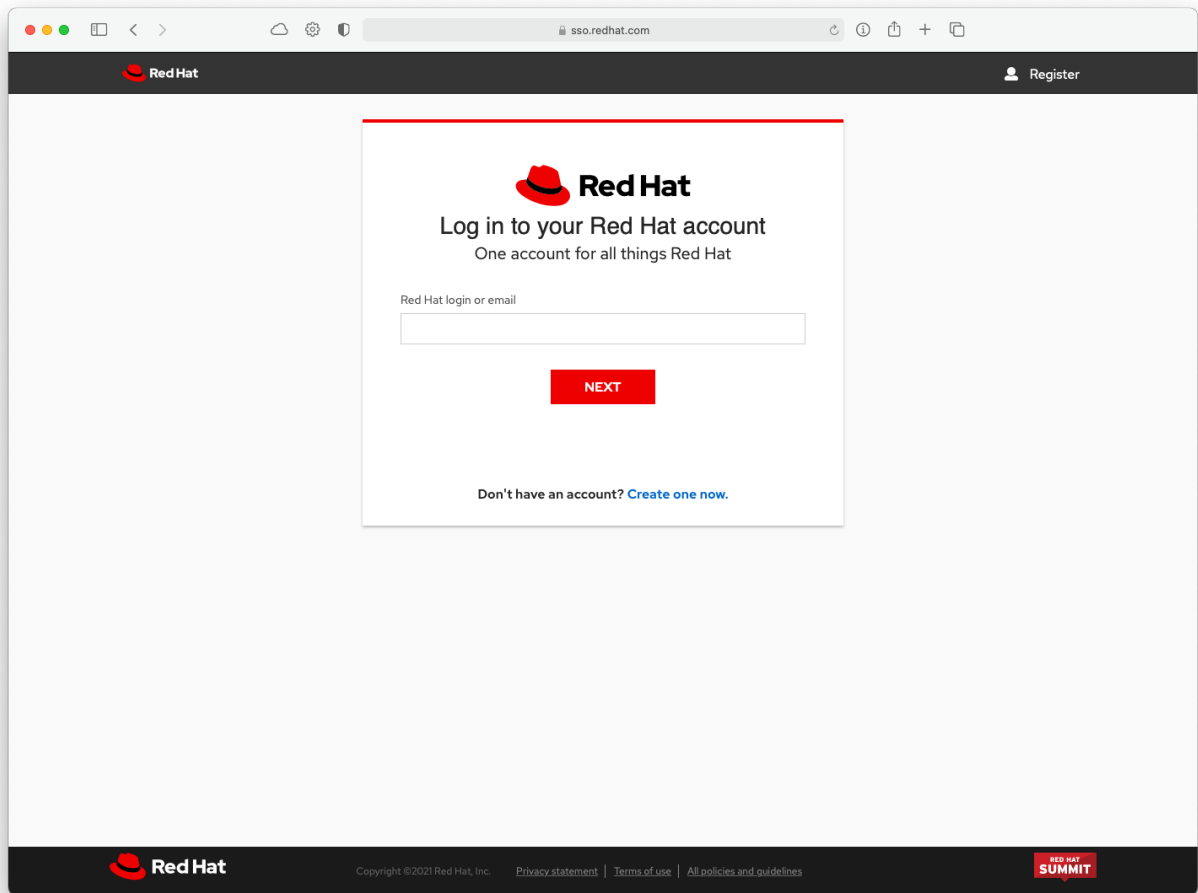
Start your OpenShift experience in four simple steps

If you're exploring how to run your code in containers, our Developer Sandbox makes it simple. Not only can you easily deploy your application from a Git repo, you can also set up a cloud IDE for your entire team. Follow these four steps to quickly get started:

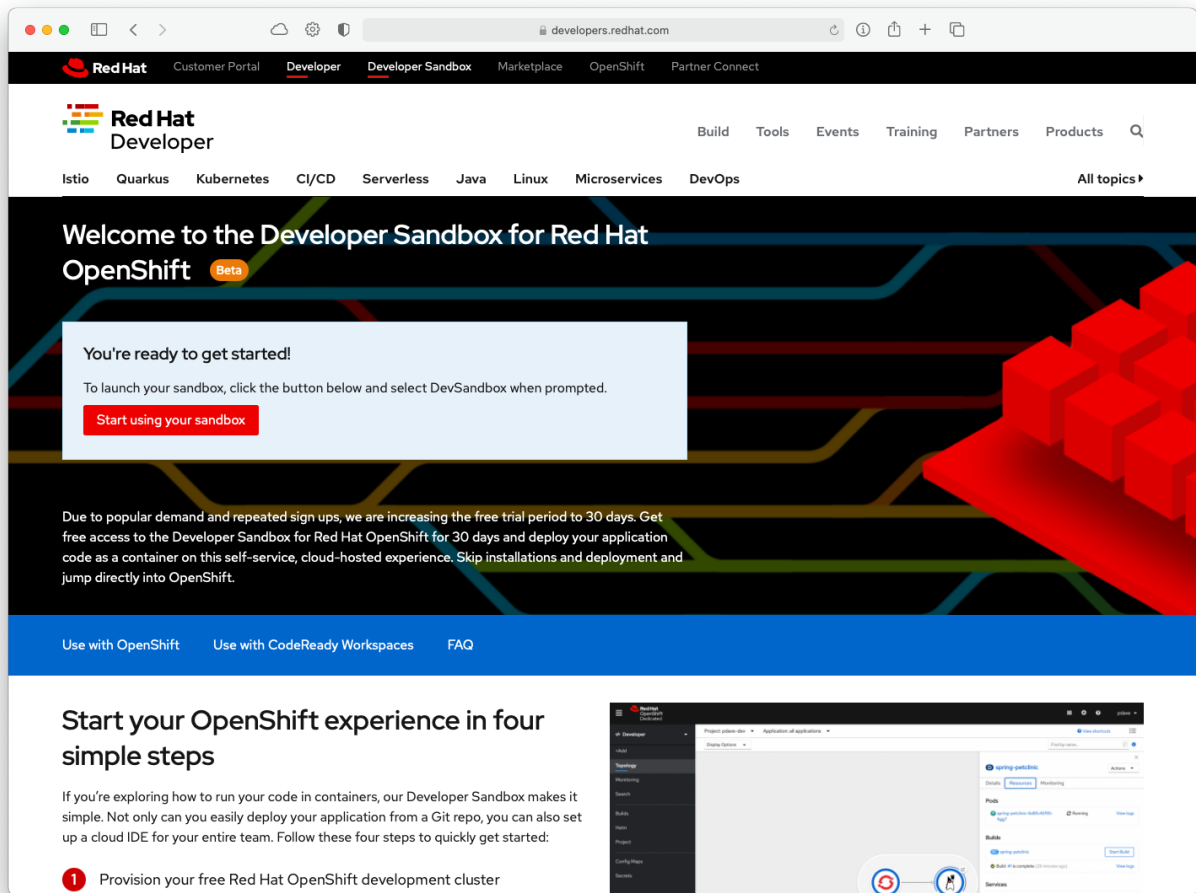
- 1 Provision your free Red Hat OpenShift development cluster
- 2 Deploy a sample application or bring your own repo
- 3 Edit code from the integrated Eclipse Che-based cloud IDE
- 4 Fully explore the Red Hat OpenShift developer experience



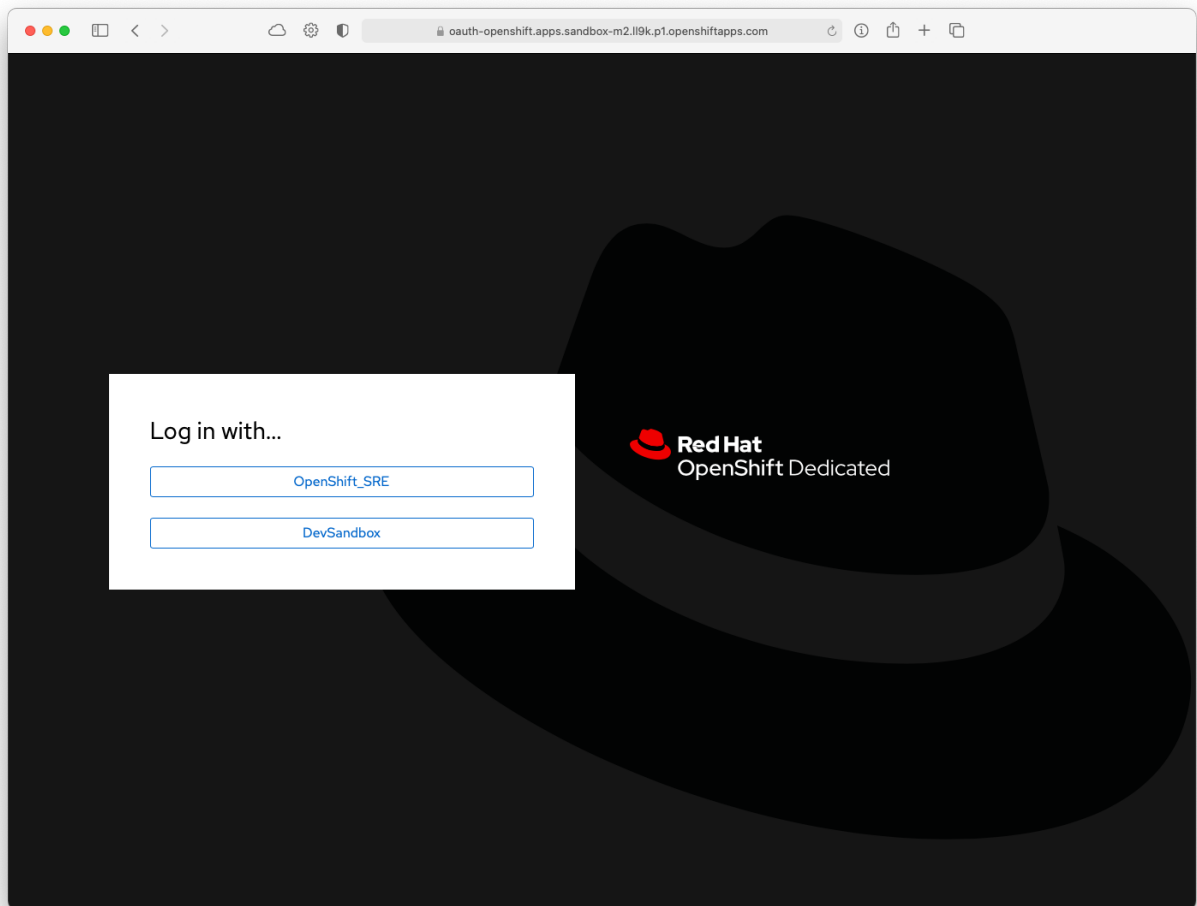
3. Create an account or Login into RedHat Portal



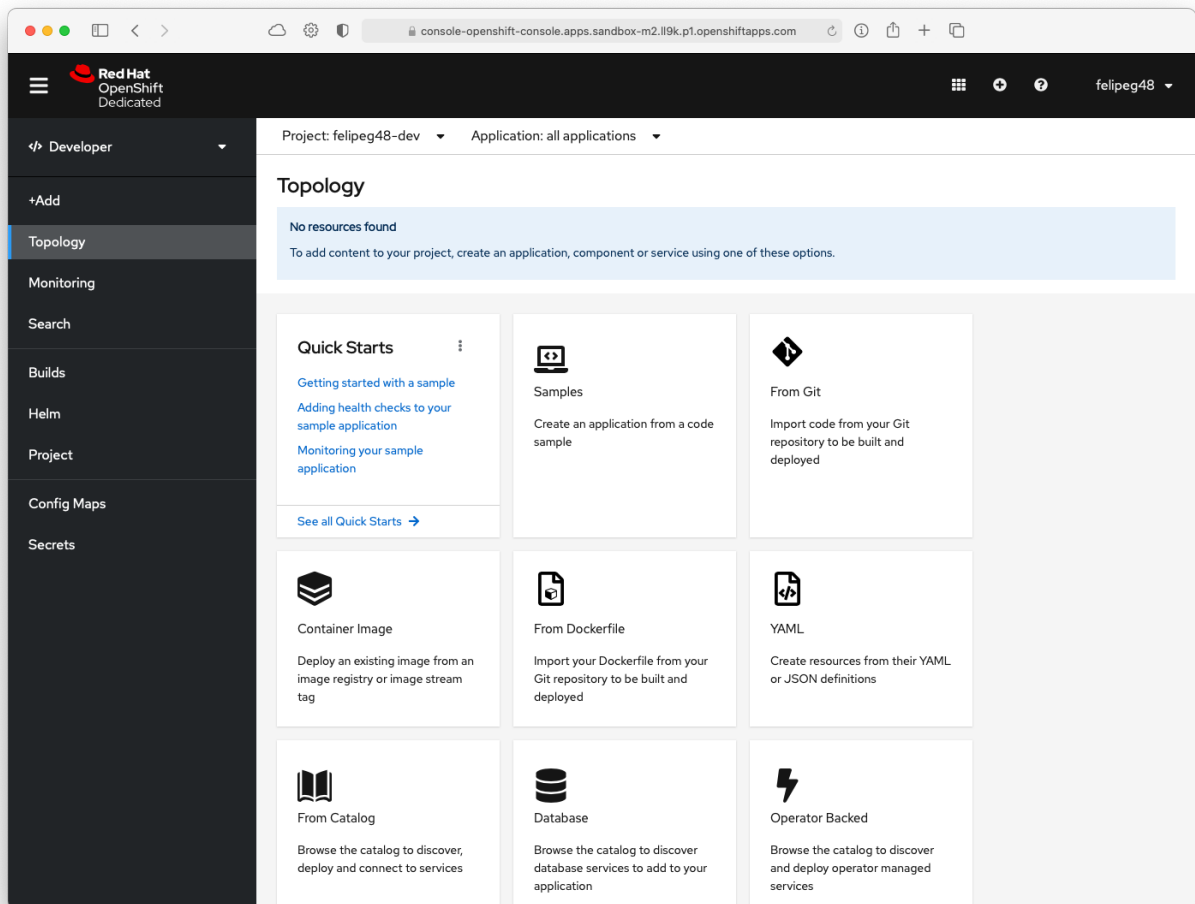
Once you login, click the **Start using your sandbox** button.

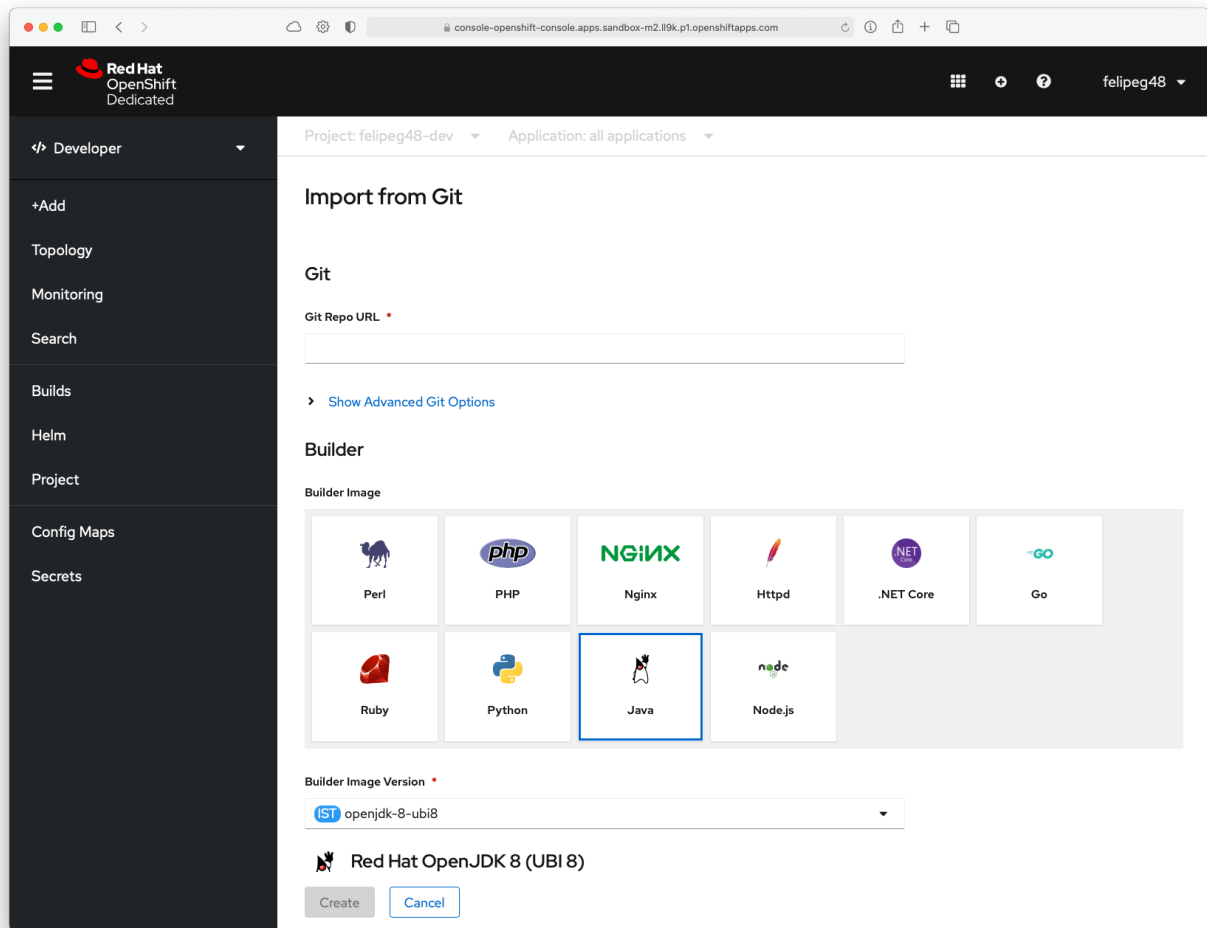


4. Select DevSandBox



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





You can click the **Routes** link and change the port to **8081**.

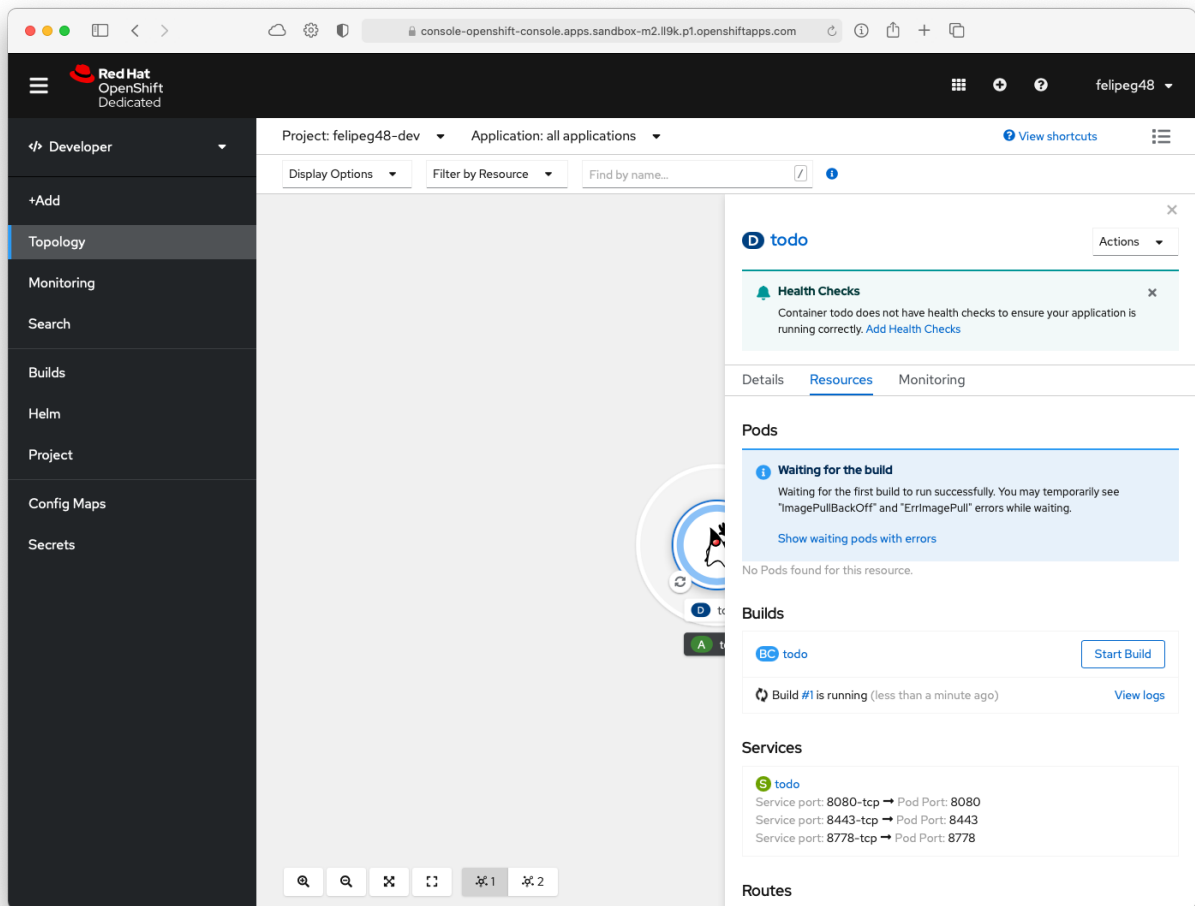


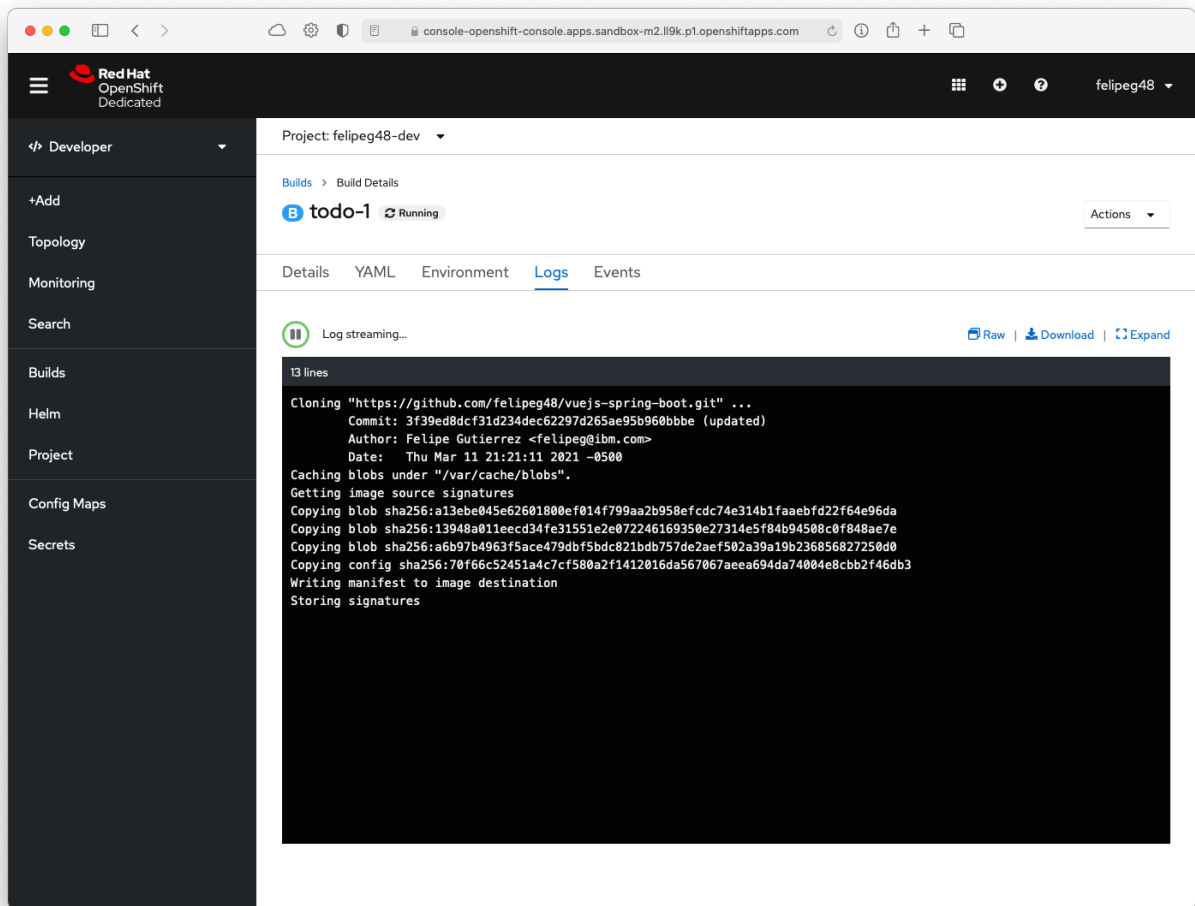
Follow the Instructor for other field's metadata.

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

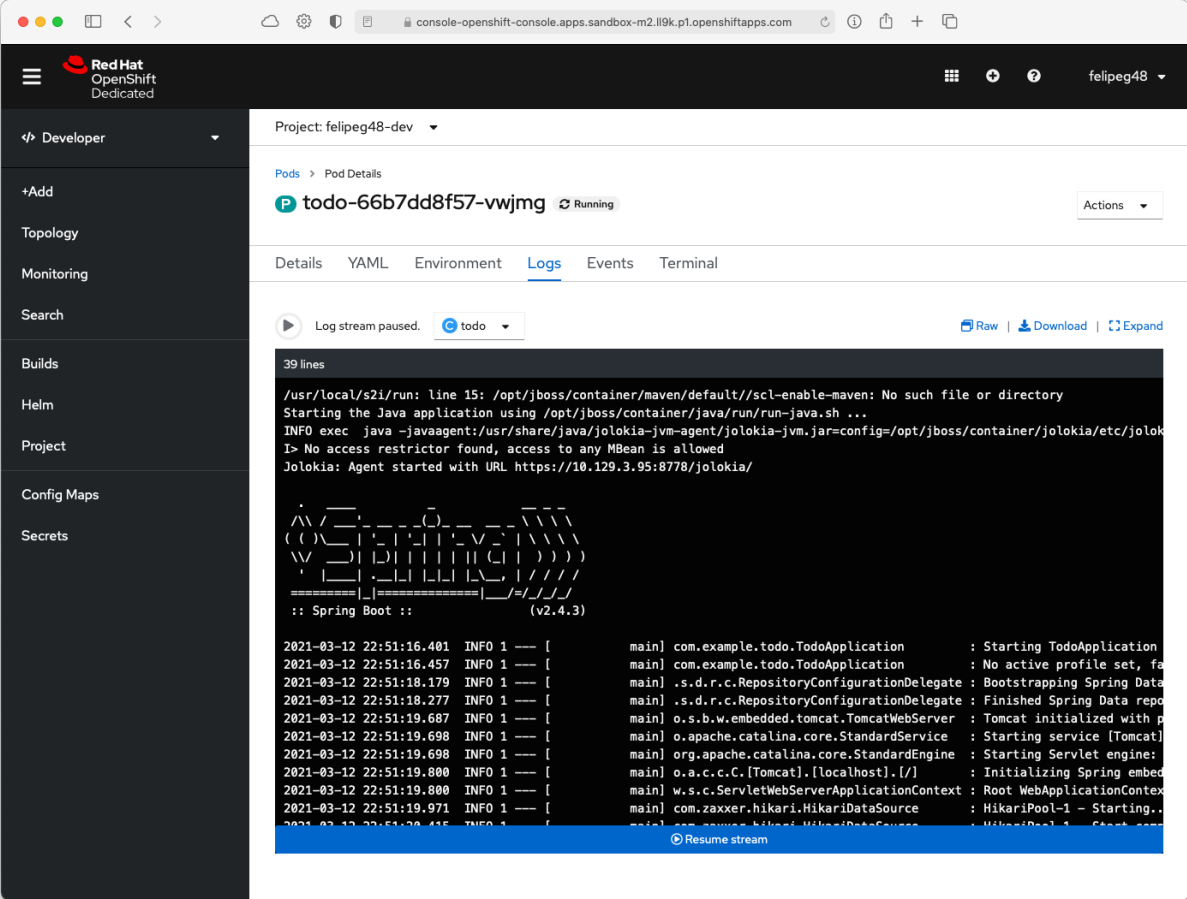
image::images/img10.png

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

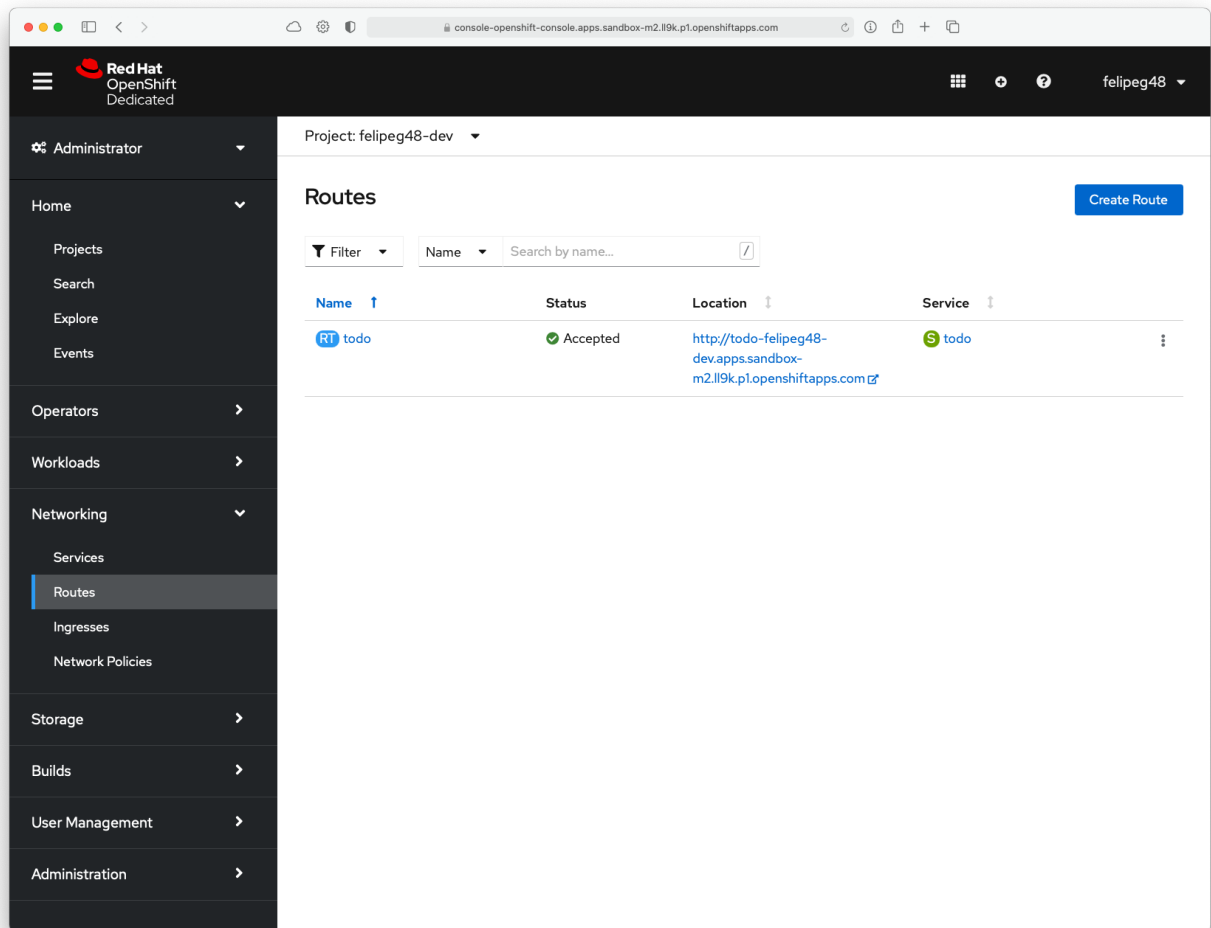


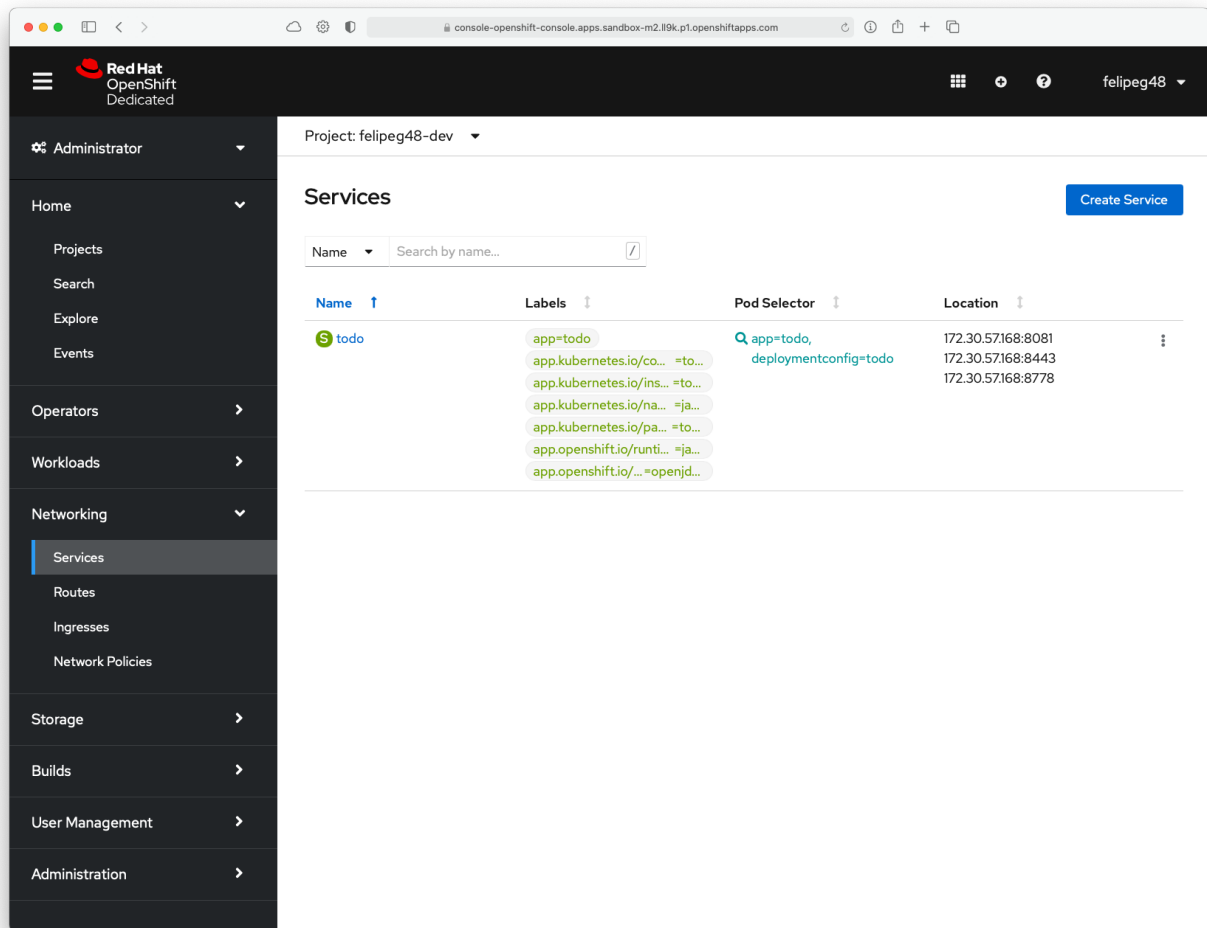


You can also see the logs of the Pods running.



7. Switch to **Administrator** mode and fix the **Route** and **Service** if necessary to point to the 8081.





8. You can access to your **ToDo** app b y clicking into the **Route**

Deploy the ToDo UI App in OpenShift Challenge

- If they apps are not running as it suppose to, fix them!