**Backstage Hands-on Tutorials**

**1. Setting Up Backstage Locally**

**Prerequisites**

* Node.js (>=14.x)
* Yarn package manager
* Git installed on your machine
* Docker (for optional database setup)

**Installation Steps**

# Install Backstage CLI

npx @backstage/create-app@latest

Follow the prompts to set up your Backstage application.

cd my-backstage-app

yarn install

yarn dev

Your Backstage instance should now be running at http://localhost:3000.

**2. Creating a Simple Backstage Plugin**

cd my-backstage-app

yarn backstage-cli create-plugin --name myplugin

This will generate a new plugin inside the plugins/ folder.

Modify src/components/ExampleComponent.tsx:

import React from 'react';

import { Content } from '@backstage/core-components';

export const ExampleComponent = () => (

<Content>

<h1>Welcome to My Plugin</h1>

</Content>

);

Rebuild and restart the app:

yarn build

yarn dev

**3. Deploying a Simple App Using Backstage**

**Register a Service in the Catalog**

1. Create a catalog-info.yaml file in your repository:

apiVersion: backstage.io/v1alpha1

kind: Component

metadata:

name: my-service

annotations:

github.com/project-slug: my-org/my-service

spec:

type: service

lifecycle: production

owner: team-a

1. Add it to Backstage via the "Create Component" button.

**4. CI/CD Pipeline Integration**

**GitHub Actions Example**

Create .github/workflows/deploy.yml:

name: CI/CD Pipeline

on:

push:

branches:

- main

jobs:

build:

runs-on: ubuntu-latest

steps:

- name: Checkout Code

uses: actions/checkout@v2

- name: Set up Node.js

uses: actions/setup-node@v2

with:

node-version: '16'

- name: Install Dependencies

run: yarn install

- name: Build Project

run: yarn build

- name: Deploy to Server

run: echo "Deploying application..."

Push this workflow to trigger automated CI/CD.

**5. Operational Monitoring and Maintenance**

1. **Enable Prometheus Metrics**
   * Install the Prometheus plugin in Backstage.
   * Configure Backstage to fetch service metrics.
2. **User Role Management**
   * Use app-config.yaml to define access policies.

**💡 1. Setting Up Backstage Locally**

**📌 Prerequisites**

* Node.js (>=14.x recommended)
* Yarn package manager
* Docker (optional for database)
* Git installed on your machine

**📌 Install Backstage**

Run the following command to create a new Backstage app:

npx @backstage/create-app@latest

**Navigate into your Backstage app:**

cd my-backstage-app

yarn install

yarn dev

Your Backstage instance should now be running at http://localhost:3000. 🎉

**💡 2. Setting Up the Service Catalog**

The **Service Catalog** lets you organize and discover services across your engineering teams.

**📌 Register a Service**

1️⃣ **Create a catalog-info.yaml file** in the root of your service repo:

apiVersion: backstage.io/v1alpha1

kind: Component

metadata:

name: my-service

description: "This is a sample service in Backstage"

annotations:

github.com/project-slug: my-org/my-service

spec:

type: service

lifecycle: production

owner: team-a

2️⃣ **Register the service in Backstage:**

* Go to **Backstage UI → Create Component**
* Paste the repo URL containing catalog-info.yaml
* Click **Register**

**💡 3. Working with APIs in Backstage**

**📌 Adding API Documentation**

1️⃣ **Create an api-info.yaml file**:

apiVersion: backstage.io/v1alpha1

kind: API

metadata:

name: my-service-api

description: REST API for My Service

spec:

type: openapi

lifecycle: production

owner: team-a

definition: |

openapi: 3.0.0

info:

title: My Service API

version: 1.0.0

paths:

/users:

get:

summary: Get all users

responses:

'200':

description: OK

2️⃣ **Register API in Backstage:**

* Go to **Backstage UI → Create Component → API**
* Add the URL of your API definition file (Swagger, OpenAPI, GraphQL)

**💡 4. Creating a Custom Backstage Plugin**

Plugins extend Backstage functionality. Let's create a **simple dashboard plugin**.

**📌 Generate a Plugin**

cd my-backstage-app

yarn backstage-cli create-plugin --name myplugin

Modify src/components/ExampleComponent.tsx:

import React from 'react';

import { Content } from '@backstage/core-components';

export const ExampleComponent = () => (

<Content>

<h1>🚀 Welcome to My Plugin</h1>

<p>This is a custom Backstage plugin.</p>

</Content>

);

**Rebuild & Restart:**

yarn build

yarn dev

**💡 5. Deploying an Application**

**📌 Deploy a Sample App Using Backstage**

Let’s deploy a **Node.js + Express App** using Backstage.

1️⃣ **Create a simple server.js file:**

const express = require('express');

const app = express();

app.get('/', (req, res) => {

res.send('🚀 My First Backstage App!');

});

app.listen(3001, () => console.log('Server running on port 3001'));

2️⃣ **Add a Dockerfile for containerization:**

Dockerfile

FROM node:16

WORKDIR /app

COPY package.json .

RUN npm install

COPY . .

CMD ["node", "server.js"]

EXPOSE 3001

3️⃣ **Run the App with Docker:**

docker build -t my-backstage-app .

docker run -p 3001:3001 my-backstage-app

**💡 6. Setting Up a CI/CD Pipeline**

Let’s integrate GitHub Actions to automate deployments.

**📌 GitHub Actions Workflow**

Create a .github/workflows/deploy.yml file:

name: CI/CD Pipeline

on:

push:

branches:

- main

jobs:

build:

runs-on: ubuntu-latest

steps:

- name: Checkout Code

uses: actions/checkout@v2

- name: Set up Node.js

uses: actions/setup-node@v2

with:

node-version: '16'

- name: Install Dependencies

run: yarn install

- name: Build Project

run: yarn build

- name: Deploy to Server

run: echo "Deploying application..."

**💡 7. Observability & Monitoring**

**📌 Integrating with Prometheus**

To enable Prometheus monitoring: 1️⃣ **Add Prometheus Plugin in Backstage**

yarn add @backstage/plugin-prometheus

2️⃣ **Modify app-config.yaml:**

prometheus:

baseUrl: "http://localhost:9090"

3️⃣ **Restart Backstage:**

yarn dev

✅ **Now you can monitor service metrics in Backstage!**

**💡 8. Role-Based Access Control (RBAC)**

**📌 Managing User Roles**

1️⃣ **Modify app-config.yaml:**

permission:

enabled: true

rules:

- action: read

resource: catalog-entity

roles: [developer, admin]

- action: write

resource: catalog-entity

roles: [admin]

2️⃣ **Restart Backstage:**

yarn dev

**📅 Day 1: Backstage Fundamentals & Service Catalog**

**🔹 Session 1: Introduction to Backstage (1 Hour)**

* What is Backstage?
* Why use Backstage for developer productivity?
* Key components:
  + **Service Catalog** (Manage services)
  + **TechDocs** (API & documentation)
  + **Plugins** (Extend functionality)
* Backstage **architecture** (Backend: Node.js, Frontend: React)

**💻 Hands-on: Install & Setup Backstage**  
1️⃣ Install Backstage locally:

npx @backstage/create-app@latest

cd my-backstage-app

yarn install

yarn dev

2️⃣ Explore the Backstage UI (http://localhost:3000)

**🔹 Session 2: Backstage Service Catalog (2 Hours)**

* **Why Service Catalog?** Discover & manage services centrally.
* **Metadata & organization** (Using catalog-info.yaml)
* **Entity types:** Component, API, Resource, System, Group

**💻 Hands-on: Registering a Service in Backstage** 1️⃣ Create a catalog-info.yaml in any GitHub

apiVersion: backstage.io/v1alpha1

kind: Component

metadata:

name: my-service

description: A sample microservice in Backstage

annotations:

github.com/project-slug: my-org/my-service

spec:

type: service

lifecycle: production

owner: team-a

2️⃣ Register the service via **Backstage UI → Create Component**  
3️⃣ View the service in **Service Catalog**

**🔹 Session 3: API Documentation & TechDocs (2 Hours)**

* **What is TechDocs?** (Technical documentation platform)
* **Managing API Docs** using OpenAPI/Swagger
* **Version control & auto-generating docs**

**💻 Hands-on: Adding API Docs to Backstage** 1️⃣ Create an api-info.yaml:

apiVersion: backstage.io/v1alpha1

kind: API

metadata:

name: user-api

description: REST API for managing users

spec:

type: openapi

lifecycle: production

owner: team-a

definition: |

openapi: 3.0.0

info:

title: User API

version: 1.0.0

paths:

/users:

get:

summary: Get all users

responses:

'200':

description: OK

2️⃣ Register the API in Backstage  
3️⃣ View API documentation in **Backstage API Explorer**

**📅 Day 2: Backstage Plugins, CI/CD & Deployments**

**🔹 Session 4: Custom Plugins & Integrations (3 Hours)**

* **Why use plugins?** (Extend Backstage functionality)
* **Types of plugins:** Frontend, Backend, Third-Party
* **Plugin architecture & development workflow**

**💻 Hands-on: Create a Custom Plugin** 1️⃣ Generate a plugin:

cd my-backstage-app

yarn backstage-cli create-plugin --name myplugin

2️⃣ Modify src/components/ExampleComponent.tsx:

import React from 'react';

import { Content } from '@backstage/core-components';

export const ExampleComponent = () => (

<Content>

<h1>🚀 My Custom Plugin!</h1>

<p>This plugin displays custom data.</p>

</Content>

);

3️⃣ Restart Backstage & view the plugin

**🔹 Session 5: CI/CD Integration with Backstage (3 Hours)**

* **Why integrate CI/CD?** (Track builds, deploy apps)
* **CI/CD providers:** GitHub Actions, Jenkins, ArgoCD
* **Viewing builds & deployments in Backstage**

**💻 Hands-on: Automate CI/CD with GitHub Actions** 1️⃣ Add .github/workflows/deploy.yml:

name: Deploy to Production

on:

push:

branches:

- main

jobs:

build:

runs-on: ubuntu-latest

steps:

- name: Checkout Code

uses: actions/checkout@v2

- name: Set up Node.js

uses: actions/setup-node@v2

with:

node-version: '16'

- name: Install Dependencies

run: yarn install

- name: Build Project

run: yarn build

- name: Deploy Application

run: echo "Deploying App..."

2️⃣ Push to GitHub & check pipeline status in **Backstage CI/CD Plugin**

**📅 Day 3: Security, Monitoring & Real-World Use Cases**

**🔹 Session 6: Role-Based Access Control (RBAC) & Security (2 Hours)**

* **Why RBAC?** Secure service access
* **Defining user roles in Backstage**

**💻 Hands-on: Implement RBAC** 1️⃣ Modify app-config.yaml:

permission:

enabled: true

rules:

- action: read

resource: catalog-entity

roles: [developer, admin]

- action: write

resource: catalog-entity

roles: [admin]

2️⃣ Restart Backstage & test access

**🔹 Session 7: Monitoring & Observability (2 Hours)**

* **Integrating Prometheus for metrics**
* **Service health monitoring**

**💻 Hands-on: Monitoring with Prometheus** 1️⃣ Install Prometheus plugin:

yarn add @backstage/plugin-prometheus

2️⃣ Configure app-config.yaml:

prometheus:

baseUrl: "http://localhost:9090"

3️⃣ Restart Backstage & view metrics

**🔹 Session 8: Real-World Use Cases & Case Study (2 Hours)**

* **Backstage adoption in large enterprises**
* **Case Study: Migrating Developer Portals to Backstage**
* **Live Demo: Deploying a Kubernetes-based service via Backstage**

**💻 Hands-on: Deploy a Service to Kubernetes** 1️⃣ Create a k8s.yaml:

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-service

spec:

replicas: 2

selector:

matchLabels:

app: my-service

template:

metadata:

labels:

app: my-service

spec:

containers:

- name: my-service

image: my-org/my-service:latest

ports:

- containerPort: 3000

2️⃣ Deploy using kubectl apply -f k8s.yaml  
3️⃣ View deployment in **Backstage Kubernetes Plugin**

**Day 1: Backstage Introduction & Setup**

**Lab 1: Installing Backstage Locally**

This lab walks you through setting up a Backstage instance on your local machine.

**Prerequisites:**

* Node.js **LTS** version (>= 16.x)
* Yarn package manager
* Git installed

**Step-by-Step Instructions:**

1. **Install Backstage using npx**

npx @backstage/create-app

cd my-backstage-app

yarn install

yarn dev

1. Open **http://localhost:3000** in your browser.
2. Explore the default Backstage interface.

**Expected Output:**

* You should see the Backstage home page with navigation and default plugins.

**Lab 2: Registering a Service in the Backstage Service Catalog**

**Objective:**

Register a new microservice in the Backstage **Service Catalog**.

**Steps:**

1. **Create a catalog-info.yaml file** inside a GitHub repository:

apiVersion: backstage.io/v1alpha1

kind: Component

metadata:

name: my-microservice

description: A sample service registered in Backstage

annotations:

github.com/project-slug: my-org/my-microservice

spec:

type: service

lifecycle: production

owner: team-a

1. **Add the service to Backstage:**
   * Navigate to **Create Component** in Backstage.
   * Paste the GitHub repository URL containing catalog-info.yaml.
   * Click **Analyze** and **Register**.

**Expected Output:**

* The microservice appears under **Service Catalog** in Backstage.

**Day 2: API Documentation, Custom Plugins, and CI/CD Integration**

**Lab 3: API Management with OpenAPI & TechDocs**

**Objective:**

Automate API documentation using **OpenAPI and TechDocs**.

**Steps:**

1. **Create api-info.yaml file** in your repo:

apiVersion: backstage.io/v1alpha1

kind: API

metadata:

name: user-api

description: User Management REST API

spec:

type: openapi

lifecycle: production

owner: team-b

1. **Generate OpenAPI docs using Swagger:**

yarn add swagger-jsdoc swagger-ui-express

1. **Modify your Express.js API to include Swagger UI:**

const swaggerJsdoc = require("swagger-jsdoc");

const swaggerUi = require("swagger-ui-express");

const options = {

definition: {

openapi: "3.0.0",

info: {

title: "User API",

version: "1.0.0",

},

},

apis: ["./routes/\*.js"], // Path to API files

};

const specs = swaggerJsdoc(options);

app.use("/api-docs", swaggerUi.serve, swaggerUi.setup(specs));

1. **Add API to Backstage API Explorer:**
   * Go to **API Explorer** in Backstage.
   * Paste the **URL of the OpenAPI spec** (/api-docs).
   * Click **Register API**.

**Expected Output:**

* The API appears under the **API Catalog** with auto-generated documentation.

**Lab 4: Creating a Custom Plugin**

**Objective:**

Develop a simple **Backstage plugin** to display a custom dashboard.

**Steps:**

1. **Generate a new plugin:**

yarn backstage-cli create-plugin --name myplugin

1. **Modify ExampleComponent.tsx to add content:**

import React from 'react';

import { createPlugin, createRouteRef } from '@backstage/core-plugin-api';

export const plugin = createPlugin({

id: 'myplugin',

routes: {

root: rootRouteRef,

},

});

export default function ExampleComponent() {

return <h1>Welcome to My Custom Plugin!</h1>;

}

1. **Register the plugin:**
   * Open packages/app/src/plugins.ts and add:

import myPlugin from '../plugins/myplugin';

export const plugins = [myPlugin];

1. **Restart Backstage:**

yarn dev

**Expected Output:**

* A new navigation tab appears with "My Custom Plugin".

**Lab 5: CI/CD Integration (GitHub Actions)**

**Objective:**

Integrate GitHub Actions into Backstage to track CI/CD pipelines.

**Steps:**

1. **Create a GitHub Actions workflow file .github/workflows/deploy.yml:**

name: CI/CD Pipeline

on:

push:

branches:

- main

jobs:

build:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v2

- name: Install dependencies

run: yarn install

- name: Run tests

run: yarn test

- name: Build project

run: yarn build

1. **Register the pipeline in Backstage:**
   * Open the service in **Service Catalog**.
   * Add GitHub repository annotation:

annotations:

github.com/project-slug: my-org/my-repo

backstage.io/ci-cd: github-actions

**Expected Output:**

* The **CI/CD status** appears in the service page on Backstage.

**Day 3: Security, Monitoring, Kubernetes, Case Study**

**Lab 6: Role-Based Access Control (RBAC)**

**Objective:**

Secure Backstage with **RBAC** using GitHub OAuth.

**Steps:**

1. **Modify app-config.yaml to enable GitHub auth:**

auth:

environment: development

providers:

github:

development:

clientId: ${GITHUB\_CLIENT\_ID}

clientSecret: ${GITHUB\_CLIENT\_SECRET}

1. **Enable role-based access:**

yaml

CopyEdit

permissions:

policies:

- allow: ['admin', 'developer']

resources: ['catalog:component']

**Expected Output:**

* Users must authenticate via GitHub to access Backstage.

**Lab 7: Monitoring with Prometheus**

**Objective:**

Integrate **Prometheus** to monitor Backstage services.

**Steps:**

1. **Install Prometheus Plugin:**

yarn add @backstage/plugin-prometheus

1. **Modify app-config.yaml to include Prometheus config:**

yaml

CopyEdit

prometheus:

baseUrl: http://prometheus-server:9090

1. **View metrics in Backstage:**
   * Navigate to **Monitoring**.

**Expected Output:**

* View service health and metrics in Backstage.

**Lab 8: Deploying a Service to Kubernetes**

**Objective:**

Deploy a sample service to **Kubernetes** and manage it from Backstage.

**Steps:**

1. **Create a Kubernetes deployment YAML file:**

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-service

spec:

replicas: 2

selector:

matchLabels:

app: my-service

template:

metadata:

labels:

app: my-service

spec:

containers:

- name: my-service

image: myorg/my-service:latest

1. **Apply the deployment:**

kubectl apply -f k8s.yaml

1. **View the service in Backstage Kubernetes Plugin.**

**Expected Output:**

* Service appears in the Kubernetes dashboard in Backstage.

These **detailed step-by-step labs** provide **hands-on learning** across **Backstage architecture, API**

**Lab 2: Understanding Backstage Project Structure**

**Objective:** Explore key components of Backstage.

1. Open **my-backstage-app** directory and review folders:
   * packages/backend/ → Backend services.
   * packages/app/ → Frontend UI.
   * plugins/ → Custom plugins.
2. Run:

yarn workspace backend start

✅ **Expected Output:** Learn Backstage architecture.

**Lab 3: Registering a Service in the Catalog**

**Objective:** Add a microservice to the **Backstage Service Catalog**.

1. Create catalog-info.yaml in a GitHub repo:

apiVersion: backstage.io/v1alpha1

kind: Component

metadata:

name: my-service

annotations:

github.com/project-slug: my-org/my-service

spec:

type: service

lifecycle: production

owner: team-a

1. Register the service in Backstage UI under **Create Component**.  
   ✅ **Expected Output:** Service appears in **Service Catalog**.

**Lab 4: Adding API Documentation**

**Objective:** Integrate **Swagger OpenAPI** with Backstage API Explorer.

1. Create api-info.yaml:

apiVersion: backstage.io/v1alpha1

kind: API

metadata:

name: user-api

annotations:

github.com/project-slug: my-org/user-api

spec:

type: openapi

owner: team-b

1. Add OpenAPI docs to an Express.js service:

app.use("/api-docs", swaggerUi.serve, swaggerUi.setup(swaggerSpecs));

1. Register API in Backstage under **API Catalog**.  
   ✅ **Expected Output:** API appears with OpenAPI documentation.

**Lab 5: Setting Up TechDocs for Service Documentation**

**Objective:** Generate and view documentation in Backstage.

1. Install TechDocs:

yarn add @backstage/plugin-techdocs-node

1. Create mkdocs.yml inside the repo:

yaml

CopyEdit

site\_name: My Service Docs

plugins:

- techdocs-core

1. Run:

npx techdocs-cli generate --no-docker

✅ **Expected Output:** View documentation in **TechDocs**.

**Lab 6: Creating a Custom Plugin**

**Objective:** Develop a simple **Backstage plugin**.

1. Run:

yarn backstage-cli create-plugin --name my-plugin

1. Modify ExampleComponent.tsx:

export default function ExampleComponent() {

return <h1>Welcome to My Plugin!</h1>;

}

1. Register the plugin in packages/app/src/plugins.ts.  
   ✅ **Expected Output:** A new **plugin tab** appears.

**Lab 7: Adding a Custom UI Component**

**Objective:** Modify Backstage UI by adding a **custom dashboard**.

1. Edit packages/app/src/App.tsx:

import { Route } from 'react-router-dom';

import { CustomDashboard } from './components/CustomDashboard';

<Route path="/custom-dashboard" element={<CustomDashboard />} />;

✅ **Expected Output:** A **custom dashboard page**.

**Lab 8: Integrating GitHub Authentication**

**Objective:** Enable authentication via GitHub.

1. Modify app-config.yaml:

auth:

providers:

github:

development:

clientId: ${GITHUB\_CLIENT\_ID}

clientSecret: ${GITHUB\_CLIENT\_SECRET}

✅ **Expected Output:** Users authenticate via GitHub.

**Lab 9: Configuring Backstage Deployment in Docker**

**Objective:** Run Backstage inside a **Docker container**.

1. Create a Dockerfile:

FROM node:16

WORKDIR /app

COPY . .

RUN yarn install && yarn build

CMD ["yarn", "dev"]

1. Run:

docker build -t backstage-app .

docker run -p 3000:3000 backstage-app

✅ **Expected Output:** Backstage runs in a **Docker container**.

**Lab 10: Deploying Backstage to Kubernetes**

**Objective:** Deploy Backstage to **Kubernetes**.

1. Create backstage-deployment.yaml:

apiVersion: apps/v1

kind: Deployment

metadata:

name: backstage

spec:

replicas: 2

selector:

matchLabels:

app: backstage

template:

metadata:

labels:

app: backstage

spec:

containers:

- name: backstage

image: backstage/backstage:latest

1. Deploy to Kubernetes:

kubectl apply -f backstage-deployment.yaml

✅ **Expected Output:** Backstage runs in **Kubernetes**.

**Day 2: Advanced Concepts (10 Labs)**

**Lab 11: Integrating CI/CD Pipelines**

1. Create .github/workflows/deploy.yml:

name: CI/CD

on: [push]

jobs:

build:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v2

- run: yarn install

- run: yarn test

✅ **Expected Output:** CI/CD status in Backstage.

**Lab 12: Automating Deployments with ArgoCD**

1. Install **ArgoCD Plugin**:

yarn add @backstage/plugin-argocd

1. Modify app-config.yaml:

yaml

CopyEdit

argocd:

baseUrl: https://argocd.example.com

✅ **Expected Output:** ArgoCD pipelines visible.

**Lab 13: Monitoring Backstage with Prometheus**

1. Install Prometheus Plugin:

yarn add @backstage/plugin-prometheus

1. Modify app-config.yaml:

prometheus:

baseUrl: http://prometheus-server:9090

✅ **Expected Output:** View Prometheus metrics.

**Lab 14: Implementing Role-Based Access Control (RBAC)**

Modify app-config.yaml:

permissions:

policies:

- allow: ['admin']

resources: ['catalog:component']

✅ **Expected Output:** Only **admins** can edit services.

**Lab 15: Service Ownership Dashboard**

✅ **Expected Output:** View service ownership details.

**Lab 16: Customizing Search with ElasticSearch**

✅ **Expected Output:** Enhanced search in Backstage.

**Lab 17: Creating a Backstage Theme**

✅ **Expected Output:** Custom UI theme.

**Lab 18: Integrating Terraform Module**

✅ **Expected Output:** Infrastructure-as-Code in Backstage.

**Lab 19: Managing Kubernetes Services**

✅ **Expected Output:** View and control Kubernetes pods.

**Lab 20: Incident Management with PagerDuty**

✅ **Expected Output:** Alerting in Backstage.

**Lab 2: Understanding Backstage Project Structure**

**Objective: Explore key Backstage components.**

**Steps:**

1. Inside my-backstage-app, review these folders:
   * packages/backend/ → Backend services.
   * packages/app/ → Frontend UI.
   * plugins/ → Custom plugins.
2. Run:

yarn workspace backend start

1. Open **http://localhost:7007** to see the backend service logs.

**Expected Output:**

✅ Understanding of how Backstage components interact.

**Lab 3: Registering a Service in the Catalog**

**Objective: Add a microservice to the Backstage Service Catalog.**

**Steps:**

1. Create catalog-info.yaml in your GitHub repo:

apiVersion: backstage.io/v1alpha1

kind: Component

metadata:

name: my-service

annotations:

github.com/project-slug: my-org/my-service

spec:

type: service

lifecycle: production

owner: team-a

1. Open Backstage, go to **Catalog → Register Component**.
2. Paste the GitHub repo link and register.

**Expected Output:**

✅ The service appears in **Backstage Service Catalog**.

**Lab 4: Adding API Documentation**

**Objective: Integrate Swagger OpenAPI with Backstage API Explorer.**

**Steps:**

1. Create api-info.yaml:

apiVersion: backstage.io/v1alpha1

kind: API

metadata:

name: user-api

annotations:

github.com/project-slug: my-org/user-api

spec:

type: openapi

owner: team-b

1. Add OpenAPI docs to an Express.js service:

const swaggerUi = require('swagger-ui-express');

app.use("/api-docs", swaggerUi.serve, swaggerUi.setup(swaggerSpecs));

1. Register API in Backstage under **API Catalog**.

**Expected Output:**

✅ The API appears with **Swagger UI** documentation.

**Lab 5: Setting Up TechDocs for Service Documentation**

**Objective: Generate and view documentation in Backstage.**

**Steps:**

1. Install TechDocs plugin:

yarn add @backstage/plugin-techdocs-node

1. Add mkdocs.yml inside the service repo:

site\_name: My Service Docs

plugins:

- techdocs-core

1. Generate docs:

npx techdocs-cli generate --no-docker

1. Register TechDocs in Backstage.

**Expected Output:**

✅ Documentation appears under **TechDocs**.

**Lab 6: Creating a Custom Plugin**

**Objective: Develop a simple Backstage plugin.**

**Steps:**

1. Run:

yarn backstage-cli create-plugin --name my-plugin

1. Modify ExampleComponent.tsx:

export default function ExampleComponent() {

return <h1>Welcome to My Plugin!</h1>;

}

1. Register the plugin in packages/app/src/plugins.ts.

**Expected Output:**

✅ A new **plugin tab** appears.

**Lab 7: Adding a Custom UI Component**

**Objective: Modify Backstage UI by adding a custom dashboard.**

**Steps:**

1. Edit packages/app/src/App.tsx:

import { Route } from 'react-router-dom';

import { CustomDashboard } from './components/CustomDashboard';

<Route path="/custom-dashboard" element={<CustomDashboard />} />;

1. Restart Backstage.

**Expected Output:**

✅ A **custom dashboard page** is visible.

**Lab 8: Integrating GitHub Authentication**

**Objective: Enable authentication via GitHub.**

**Steps:**

1. Modify app-config.yaml:

auth:

providers:

github:

development:

clientId: ${GITHUB\_CLIENT\_ID}

clientSecret: ${GITHUB\_CLIENT\_SECRET}

1. Restart Backstage and try logging in with GitHub.

**Expected Output:**

✅ Users authenticate via GitHub.

**Lab 9: Configuring Backstage Deployment in Docker**

**Objective: Run Backstage inside a Docker container.**

**Steps:**

1. Create a Dockerfile:

FROM node:16

WORKDIR /app

COPY . .

RUN yarn install && yarn build

CMD ["yarn", "dev"]

1. Run:

docker build -t backstage-app .

docker run -p 3000:3000 backstage-app

**Expected Output:**

✅ Backstage runs in a **Docker container**.

**Lab 10: Deploying Backstage to Kubernetes**

**Objective: Deploy Backstage to Kubernetes.**

**Steps:**

1. Create backstage-deployment.yaml:

apiVersion: apps/v1

kind: Deployment

metadata:

name: backstage

spec:

replicas: 2

selector:

matchLabels:

app: backstage

template:

metadata:

labels:

app: backstage

spec:

containers:

- name: backstage

image: backstage/backstage:latest

1. Deploy to Kubernetes:

kubectl apply -f backstage-deployment.yaml

**Expected Output:**

✅ Backstage runs in **Kubernetes**.

**Additional Labs for MEAN, MERN, and Docker**

**Lab: Deploying a MEAN Stack Application in Docker**

**Objective: Deploy a MongoDB + Express + Angular + Node.js stack in Docker.**

**Steps:**

1. Create docker-compose.yml:

version: "3"

services:

mongodb:

image: mongo

ports:

- "27017:27017"

backend:

build: ./backend

ports:

- "5000:5000"

frontend:

build: ./frontend

ports:

- "4200:4200"

1. Run:

docker-compose up -d

**Expected Output:**

✅ The MEAN application runs in **Docker containers**.

**Lab: Deploying a MERN Stack Application in Kubernetes**

**Objective: Deploy MongoDB + Express + React + Node.js in Kubernetes.**

**Steps:**

1. Create mern-deployment.yaml.
2. Deploy:

kubectl apply -f mern-deployment.yaml

**Expected Output:**

✅ The MERN app runs on **Kubernetes**.