**Lab1: Set up Backstage Developer Portal (Local/Cloud)**

**🎯 Objective**

Install and run **Backstage** as the face of your Internal Developer Platform (IDP).  
Explore **Service Catalog**, **Plugins**, and **TechDocs basics**.

**🧠 Why This Matters**

Backstage is the **developer’s home base**.  
All self‑service workflows, CI/CD triggers, documentation, and infrastructure requests will surface here.

**📋 Prerequisites**

* **Ubuntu 20.04/22.04** VM or local machine
* Node.js **v18.x**
* Yarn (via Corepack)
* Git installed (git --version)
* **Optional**: Docker (needed later for container builds)
* **Cloud VMs**: Ability to open ports **3000** and **7007** in firewall/security group

**🛠 Step 1 – Install Node.js 18**

bash

CopyEdit

# Update package index

sudo apt update -y

# Download and set up NodeSource repo for Node 18

curl -fsSL https://deb.nodesource.com/setup\_18.x -o nodesource\_setup.sh

sudo bash nodesource\_setup.sh

# Install Node.js

sudo apt-get install -y nodejs

**Why Node 18?** Backstage supports LTS versions; Node 20+ introduces breaking changes.

**🛠 Step 2 – Install Yarn (via Corepack)**

bash

CopyEdit

sudo npm install -g corepack

corepack enable

**🛠 Step 3 – Create Backstage App**

bash

CopyEdit

# Create a project folder

mkdir backstage-app && cd backstage-app

# Scaffold Backstage

npx @backstage/create-app@latest

# When prompted:

# - Confirm with "y"

# - Enter app name: raman-idp-portal

**🛠 Step 4 – Start Backstage**

bash

CopyEdit

cd raman-idp-portal

yarn install

yarn start

Visit [**http://publiIP:3000**](http://publiIP:3000) in your browser.

**🛠 Step 5 – Configure for Cloud/VM Access**

If running in the cloud, edit app-config.yaml:

app:

title: My Backstage Portal

baseUrl: http://<VM\_PUBLIC\_IP>:3000

listen:

host: 0.0.0.0

backend:

baseUrl: http://<VM\_PUBLIC\_IP>:7007

listen:

host: 0.0.0.0

cors:

origin: http://<VM\_PUBLIC\_IP>:3000

methods: [GET, HEAD, PATCH, POST, PUT, DELETE]

credentials: true

**🛠 Step 6 – Restart Backstage**

bash

CopyEdit

yarn start

Access via:

cpp

CopyEdit

http://<VM\_PUBLIC\_IP>:3000

**🛠 Step 7 – Open Firewall Ports (Cloud Only)**

Allow inbound traffic for:

* **3000** (frontend)
* **7007** (backend API)

**🔍 Verification**

✅ Portal loads in browser.  
✅ Service Catalog visible.  
✅ App works locally or from public IP.

**Lab 2 – Explore Backstage UI**

**🎯 Objective**

Familiarize yourself with the Backstage Developer Portal (IDP) environment and understand its **core navigation**, **components**, and **terminology**.

By the end of this lab, you should be able to:

* Navigate the **Service Catalog**
* Understand different **entity types**
* Explore **TechDocs**
* Access the **Create** menu
* Use the **Search** and **Plugin** features

**🧠 Why This Matters**

Your IDP is **not just a UI** — it’s a **central hub** for:

* Discovering **all services & resources** in your organization
* Accessing **self‑service workflows**
* Finding **documentation** without switching tools
* Triggering **pipelines** and **infrastructure requests**

Before you start **scaffolding services**, you need to **understand the landscape** of the portal.

**📋 Prerequisites**

* Completed **Lab 1 – Backstage Setup**
* Backstage running locally or on a cloud VM
  + Local: http://localhost:3000
  + Cloud: http://<VM\_PUBLIC\_IP>:3000

**🛠 Step‑by‑Step Instructions**

**Step 1 – Launch the Backstage Portal**

1. Make sure your Backstage app is running:

bash

CopyEdit

cd <your-backstage-app>

yarn start

1. Open in your browser:
   * Local: http://localhost:3000
   * Cloud: http://<VM\_PUBLIC\_IP>:3000

**Step 2 – Home Page Overview**

When Backstage loads:

* **Top Navigation Bar**
  + **Catalog** → View all registered entities (services, APIs, resources)
  + **Create** → Start new service/project via templates (Golden Paths)
  + **Docs** → Access TechDocs documentation
  + **Search** → Find services, docs, APIs, plugins
* **Left Sidebar**
  + Your organization’s navigation (customizable)
  + Links to plugins like GitHub, Kubernetes, CI/CD, Grafana (once installed)

**Tip:** Point out that this navigation is **configurable** per organization.

**Step 3 – Explore the Service Catalog**

**Reference for now :** **https://demo.backstage.io/home**

1. Click **Catalog** in the top menu.
2. You’ll see a list of **entities** (may be empty if nothing registered yet).
3. Understand **Entity Types**:
   * **Component** → A service, library, or tool your team owns
   * **API** → An interface specification for services
   * **Resource** → Infrastructure like databases, buckets, queues
   * **System** → A grouping of components and resources
   * **Domain** → High‑level business area grouping systems
4. Try the **Filters**:
   * Filter by Owner
   * Filter by Kind (Component, API, etc.)
   * Filter by Lifecycle (production, experimental, etc.)

**Tip:**  
This is the **single source of truth** for your internal software ecosystem.

**Step 4 – View a Service (Entity Page)**

1. Click on any service/component in the catalog (or register one quickly if empty).
2. Explore tabs:
   * **Overview** → Metadata, description, tags
   * **Docs** → Service documentation (if enabled)
   * **CI/CD** → Builds & deployment history (once integrated)
   * **API** → API details & schema
   * **Links** → Related resources, Git repo, dashboards

**Step 5 – Explore TechDocs**

1. Click **Docs** in the top navigation.
2. Select any available documentation (or the sample documentation if scaffolded by Backstage).
3. Notice:
   * Navigation on the left
   * Search bar for docs
   * How docs stay inside the Backstage portal (no context‑switching)

**Step 6 – Explore the Create Menu**

1. Click **Create** in the top navigation.
2. You’ll see available templates (Golden Paths).
3. These may include:
   * Service templates (Node.js, Python, Java, etc.)
   * Infrastructure templates (Terraform modules)
   * Documentation sites
4. Click on a template to preview:
   * Form fields you need to fill (service name, owner, repo)
   * Steps it will take (create repo, register in catalog, etc.)

**Step 7 – Try the Search Feature**

1. Click **Search** in the top menu.
2. Try searching for:
   * A service name
   * An API name
   * A keyword in documentation
3. Notice how results are grouped by **entity type**.

**Step 8 – Explore Plugins**

1. Scroll through the left sidebar and check any pre‑installed plugins.
2. Depending on your Backstage setup, you might see:
   * GitHub PRs
   * Kubernetes overview
   * CI/CD builds
   * Grafana dashboards
3. Click into one to explore its UI and purpose.

**🔍 Verification**

✅ You can navigate to **Service Catalog** and explain each **entity type**  
✅ You can open and view **TechDocs** inside Backstage  
✅ You can find templates in the **Create** menu  
✅ You can search for services, docs, and APIs  
✅ You can locate and open installed plugins

**📌 End‑of‑Lab Talking Points**

* **The Service Catalog is your org’s single source of truth** — everything starts here.
* **TechDocs keeps documentation close to the code and inside the portal**.
* **Create menu is your gateway to Golden Paths** .
* **Plugins extend Backstage** to integrate your existing developer tools.
* **Search makes everything discoverable** without hunting across tools.

**Lab 3 – Add GitHub Pull Requests Plugin to Backstage**

**🎯 Objective**

Integrate the **GitHub Pull Requests Plugin** into your Backstage Developer Portal so that developers can **view, filter, and analyze pull requests** directly inside the IDP.

By the end of this lab, you will:

* Install the **GitHub PR plugin**
* Configure **service entity pages** to show PRs
* Add **overview widgets** to service pages
* Understand **how to enable GitHub authentication** (required for real data)

**🧠 Why This Matters**

In most organizations, **code review is central** to software delivery.  
Developers spend a lot of time:

* Switching between GitHub and internal tools
* Tracking PRs waiting for review
* Checking merge stats & cycle time

By integrating GitHub PRs into Backstage:

* Everything is **in one portal**
* Review bottlenecks become **visible to teams**
* Managers get **review velocity insights**
* No more **tab‑switching** between GitHub and Backstage

**📋 Prerequisites**

* Completed **Lab 1 – Backstage Setup**
* Completed **Lab 2 – Explore Backstage UI**
* **GitHub account** and a repository with PRs
* **GitHub Personal Access Token (PAT)** with repo scope *(for local testing)*
* GitHub Authentication configured in Backstage *( wil do Oauth auth afterwards)*

**🛠 Step‑by‑Step Instructions**

**Step 1 – Install the Plugin**

From your **frontend app package**:

bash

CopyEdit

cd packages/app

yarn add @roadiehq/backstage-plugin-github-pull-requests

**Step 2 – Add Plugin Tab to Service Entity Pages**

Open:

bash

CopyEdit

packages/app/src/components/catalog/EntityPage.tsx

Add imports:

import {

EntityGithubPullRequestsOverviewCard,

EntityGithubPullRequestsContent,

isGithubPullRequestsAvailable

} from '@roadiehq/backstage-plugin-github-pull-requests';

Add the **Pull Requests** tab inside serviceEntityPage:

ts

CopyEdit

<EntityLayout.Route

path="/pull-requests"

title="Pull Requests"

// Optional: only show if entity has GitHub PR annotations

// if={isGithubPullRequestsAvailable}

>

<EntityGithubPullRequestsContent />

</EntityLayout.Route>

**Explanation:**

* Creates a **new “Pull Requests” tab** for each registered service entity.
* Displays a list of PRs with filters, search, and basic stats.

**Step 3 – Add Overview Widget**

Still in:

bash

CopyEdit

packages/app/src/components/catalog/EntityPage.tsx

Import:

ts

CopyEdit

import {

EntityGithubPullRequestsOverviewCard,

} from '@roadiehq/backstage-plugin-github-pull-requests';

Inside the **overviewContent** section, add:

ts

CopyEdit

<Grid item md={6}>

<EntityGithubPullRequestsOverviewCard />

</Grid>

**Explanation:**

* Shows **PR summary metrics** directly in the **Overview** tab:
  + Average time to merge
  + Merge‑to‑close ratio
  + Avg PR size
  + Avg files changed

**Prepare a Sample Repo Beforehand (Simulation)**

You can quickly create a **demo repo** in your GitHub account:

1. Go to **GitHub → New repository**
2. Name it: sample-service
3. Initialize with a README
4. Make **one or two branches** (e.g., feature-1, bugfix-1)
5. Create **two Pull Requests**
   * Merge one
   * Leave one open
   * This gives demo data.

Then:

* Add the annotation in catalog-info.yaml
* Install PR plugin (already done)
* You will see **real PRs** from your demo repo.

**Step 4 – Simultae a Service in the Catalog**

For the PR plugin to work, services in your catalog must include the **GitHub repo annotation**.

Example catalog-info.yaml:

apiVersion: backstage.io/v1alpha1

kind: Component

metadata:

name: sample-service

description: demo service

annotations:

github.com/project-slug: ramannkhanna2/sample-service

spec:

type: service

owner: team-a

lifecycle: experimental

**2️⃣ How to use a GitHub Token with the PR Plugin (Local Only)**

**Step 1 – Create a GitHub Personal Access Token**

1. Go to **GitHub → Settings → Developer settings → Personal Access Tokens → Tokens (classic)**.
2. Click **Generate new token (classic)**.
3. Name it: backstage-demo-token
4. Select scopes:
   * repo (full repo access)
   * read:org *(optional for org repos)*
5. Generate and **copy the token**.

**Step 2 – Add Token to Backstage Environment**

Edit your **backend environment** variables (local only):

bash

CopyEdit

export GITHUB\_TOKEN=ghp\_xxxxxxxxxxxxxxxxxxxxxx

You can also add it to:

bash

CopyEdit

.env

with:

bash

CopyEdit

GITHUB\_TOKEN=ghp\_xxxxxxxxxxxxxxxxxxxxxx

**Step 3 – Pass Token to the GitHub API Integration**

In your Backstage app-config.yaml :

yaml

CopyEdit

integrations:

github:

- host: github.com

token: ${GITHUB\_TOKEN}

**Step 4 – Restart Backstage**

bash

CopyEdit

yarn start

NOTE : MAKE SURE U HAVE CHECKED UR PUBLIC IP IN APP.CONFIG.YAML FOR FRNTEND , BAKEND AND CORS ..

**Step 5 – Test**

* Open your service in **Service Catalog**.
* Go to **Pull Requests** tab.
* You should now see **real PRs** from your repo.

**Step 7 – Verify**

1. Open a service in **Service Catalog** that has a GitHub annotation.
2. See **Pull Requests** tab → List of PRs with filters/search.
3. Check **Overview** tab → PR summary card visible.

**🔍 Verification Checklist**

✅ Plugin installed via yarn  
✅ “Pull Requests” tab added to service entity pages  
✅ Overview widget showing PR stats  
✅ GitHub annotations present in service metadata  
✅ Data loads (if authentication configured)

**📌 End‑of‑Lab Talking Points**

* This is how **Backstage integrates external developer workflows**.
* Same pattern applies to **Grafana dashboards**, **Kubernetes views**, **CI/CD history**, etc.
* By adding GitHub PR visibility into the IDP:
  + Reduces **context switching**
  + Improves **review velocity**
  + Makes **review metrics** part of DevEx

**Lab 4 : Register sample-service in Backstage Catalog**

Backstage does not auto-discover services — you must **register** them manually.

**2️⃣ If using GitHub repo**

1. Push catalog info.yml file to your **sample-service** GitHub repo.
2. Go to **GitHub → your repo → click catalog-info.yaml → Copy the file URL**.  
   Example:

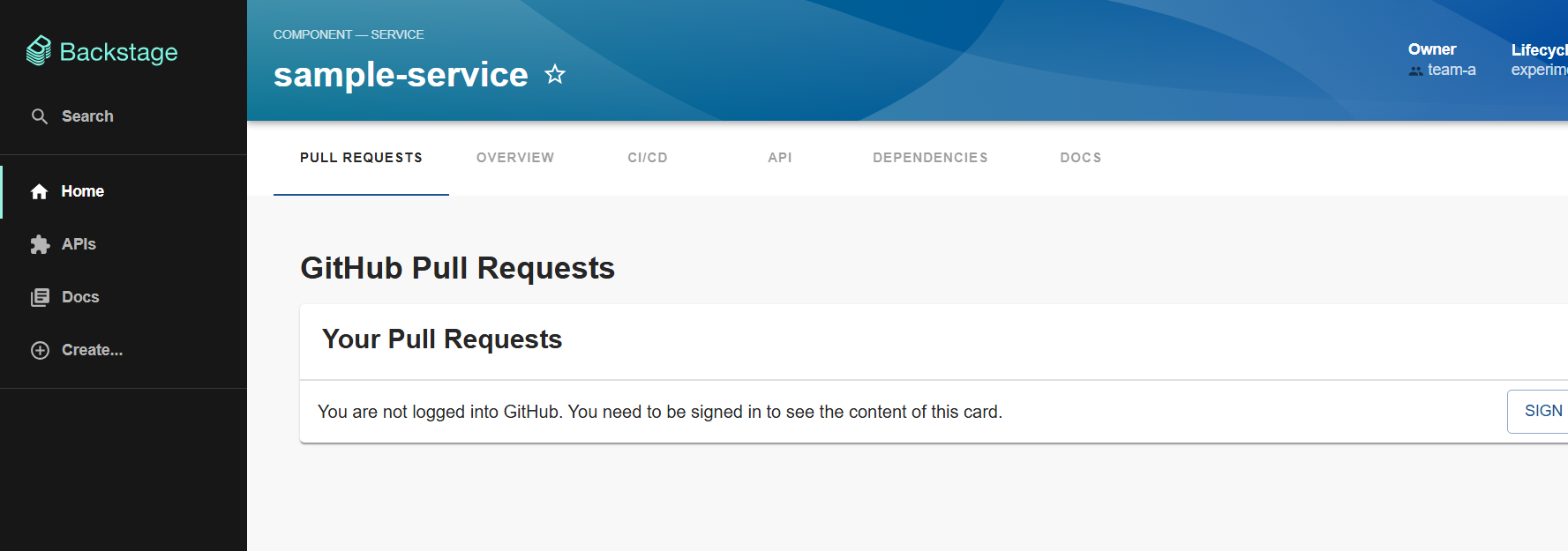
https://github.com/ramannkhanna2/sample-service/blob/main/catalog-info.yaml

**3️⃣ Register in Backstage**

1. Open your Backstage UI.
2. In the sidebar → click **Create… → Register Existing Component**.
3. Paste the **file URL** (if in GitHub) or file:/path/to/catalog-info.yaml .
4. Click **Analyze** → **Import**.
5. Wait for success message.

**4️⃣ Verify**

1. Go to **Catalog → Components**.
2. You should see sample-service.
3. Click it → open **Overview**.
4. Check if **Pull Requests** tab shows up (PR plugin).



**Lab5 : GitHub Oauth Setup configuration:**

**GitHub Token** – Added earlier so Backstage backend can fetch public/private repo metadata (e.g., PR lists) from GitHub.  
**GitHub OAuth** – Adding now in next activity so individual users can log in and see their **personalized GitHub data** (e.g., their PRs, review requests, activity).

**Install GitHub Auth in Backend (if not already done)**

yarn --cwd packages/backend add @backstage/plugin-auth-backend-module-github-provider

Important addition :

And add the following dependency to your backend index file:

(venv) root@ip-172-31-14-172:~/backstage-app/raman-idp-portal/packages/backend/src# vi index.ts :

backend.add(import(‘@backstage/plugin-auth-backend-module-github-provider’));

**https://backstage.io/docs/auth/**

**2.1: Create GitHub Oauth App**

Go to: [**https://github.com/settings/developers**](https://github.com/settings/developers) → Oauth Apps → “New Oauth App”

| **Field** | **Value** |
| --- | --- |
| App Name | Backstage Raman |
| Homepage URL | <http://PubIP> :3000 |
| Callback URL | [http:// PubIP](http://localhost) :7007/api/auth/github/handler/frame |

After creating it, copy the **Client ID** and **Client Secret**

**2.2: Add Credentials to app-config.yaml**

In ~/backstage-app/raman-app/app-config.yaml:

yaml

CopyEdit

auth:

providers:

github:

development:

clientId: ${AUTH\_GITHUB\_CLIENT\_ID}

clientSecret: ${AUTH\_GITHUB\_CLIENT\_SECRET}

Now add these in your or .env:

AUTH\_GITHUB\_CLIENT\_ID=your-client-id

AUTH\_GITHUB\_CLIENT\_SECRET=your-client-secret

**LOAD .env file in the backstage env :**

-- Run this in your repo root (where package.json is):

yarn workspace backend add dotenv

-- Load .env in backend

-- In packages/backend/src/index.ts, add this at the very top:

- vi packages/backend/src/index.ts

import path from 'path';

import dotenv from 'dotenv';

// Load environment variables from root .env file

dotenv.config({

path: path.resolve(\_\_dirname, '../../../.env'),

});

* + - Yarn start again

