**Lab 13 : ADDING MORE ENTITIES ON BACKSTAGE**

**DEPENDENCY GENERATION** :

-- inside raman\_micro\_test/catalog-info.yaml

(venv) root@ip-172-31-87-197:~/

spec:

type: service

lifecycle: experimental

# lifecycle: production

# lifecycle: depreciated

owner: group:platform-team

dependsOn:

- component:default/example-website

# dependencyOf:

# - component:default/artist-web-lookup

==================================================================

**ADDING USER AND GROUP ENTITIES** :

-- created 2 folders catalog and entities under backstage-app/ramanapp/ :

(venv) root@ip-172-31-87-197:~/backstage-app/ramanapp/catalog/entities# pwd

/root/backstage-app/ramanapp/catalog/entities

(venv) root@ip-172-31-87-197:~/backstage-app/ramanapp# cat catalog/entities/group.yaml

apiVersion: backstage.io/v1alpha1

kind: Group

metadata:

name: platform-team

description: The infra team

spec:

type: team

profile:

displayName: Infra team

email: infrastructure@example.com

children: []

members: [raman]

(venv) root@ip-172-31-87-197:~/backstage-app/ramanapp# cat catalog/entities/user.yaml

apiVersion: backstage.io/v1alpha1

kind: User

metadata:

name: raman

spec:

profile:

displayName: Raman

email: raman@example.com

memberOf: [platform-team]

-- refer the location of above entity files in under catalaog section in /root/backstage-app/ramanapp/app-config.yaml :

catalog:

import:

entityFilename: catalog-info.yaml

pullRequestBranchName: backstage-integration

rules:

- allow: [Component, System, API, Resource, Location]

locations:

- type: file

target: /root/backstage-app/ramanapp/catalog/entities/user.yaml

rules:

- allow: [User]

- type: file

target: /root/backstage-app/ramanapp/catalog/entities/group.yaml

rules:

- allow: [Group]

--------------------------

-- than yarn start ....

==========================================================

* We can also add our microservice “raman-micro” github-location under location section in app-config-yaml to make it persistent so that we don’t have register again and again .
* Add below under location section :

- type: url

target: https://github.com/ramannkhanna2/raman\_micro\_test/blob/main/catalog-info.yaml

rules:

- allow: [Component]

**ADDING TEMPLATE ENTITY** :

* Template skeleton is already at https://github.com/ramannkhanna2/backstage\_template.git

- type: url

target: https://github.com/ramannkhanna2/backstage\_template/blob/main/templates/raman-node-template/template.yaml

rules:

- allow: [Template]

**Lab 14: Install the Grafana comunity Plugin on backstage.**

Two community versions available:

* Official Backstage community version: **@backstage-community/plugin-grafana**
* Roadie fork by K‑Phoen: **@k-phoen/backstage-plugin-grafana**

📦 For most users, the **community version** is ideal:

<https://github.com/backstage/community-plugins/blob/main/workspaces/grafana/plugins/grafana/docs/setup.md>

root@ip-172-31-14-172:~/backstage-app/ramanapp/packages/app# yarn add @backstage-community/plugin-grafana

➤ YN0000: · Yarn 4.4.1

➤ YN0000: ┌ Resolution step

➤ YN0000: └ Completed in 0s 789ms

➤ YN0000: ┌ Post-resolution validation

➤ YN0060: │ @testing-library/react is listed by your project

* To vierify if plugin installed :

root@ip-172-31-14-172:~/backstage-app/ramanapp/packages/app# cat package.json

* U shud see the plugin there .
* Go to Grafana and generate a service account and token

**🔧 Step 2: Configure Proxy and Grafana URL (if Grafana requires auth or is remote)**

Edit **app-config.yaml** (at project root):

proxy:

### Example for how to add a proxy endpoint for the frontend.

### A typical reason to do this is to handle HTTPS and CORS for internal services.

# endpoints:

# '/test':

# target: 'https://example.com'

# changeOrigin: true

'/grafana/api':

target: http://54.233.49.51:3001

headers:

# Only needed if your Grafana API requires an auth token

Authorization: Bearer glsaRxAGZz5qCqX8j8dqflkgbut #grafana sa token

grafana:

domain: http://54.233.49.51:3001

# If you're using Grafana's new unified alerting:

unifiedAlerting: false

* Expose the plugin to Backstage:
* Create a file named plugins.tsx in ~/backstage-app/ramanapp/packages/app/src
* // packages/app/src/plugins.tsx
* // other plugins...
* export { grafanaPlugin } from '@backstage-community/plugin-grafana';
* **for testing if ur able to reach Grafana ui from api :**

curl -H "Authorization: Bearer glsa\_RxAGZz5qCqX8j8dqflnr8oi" http://54.233.49.51:3001/api/search?tag=raman-micro

**🧩 Step 3: Update Service Entity Layout to Include Dashboard Card**

[**https://github.com/backstage/community-plugins/blob/main/workspaces/grafana/plugins/grafana/docs/dashboards-on-component-page.md**](https://github.com/backstage/community-plugins/blob/main/workspaces/grafana/plugins/grafana/docs/dashboards-on-component-page.md)

Edit **packages/app/src/components/catalog/EntityPage.tsx** (or wherever you layout service pages):

**Display dashboards on a component page**

Adding the EntityGrafanaDashboardsCard component to an entity's page will display a list of dashboards related to that entity.

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

import { EntityGrafanaDashboardsCard } from '@backstage-community/plugin-grafana';

// ...

const overviewContent = (

<Grid container spacing={3} alignItems="stretch">

<Grid item md={6}>

<EntityAboutCard variant="gridItem" />

</Grid>

<Grid item md={6}>

{/\* Grafana alert card start \*/}

<EntityGrafanaDashboardsCard />

{/\* Grafana alert card end \*/}

</Grid>

<Grid item md={4} xs={12}>

<EntityLinksCard />

</Grid>

<Grid item md={8} xs={12}>

<EntityHasSubcomponentsCard variant="gridItem" />

</Grid>

</Grid>

);

* on Grafana ui add the tag to the "count" named dashboard "raman-micro" so that backstage plugin can find the dashboard of urs in Grafana ..

**🔁 Step 4: Ensure Your catalog-info.yaml Has the Grafana Annotation**

root@ip-172-31-14-172:~/raman-micro# cat catalog-info.yaml

apiVersion: backstage.io/v1alpha1

kind: Component

metadata:

name: raman-micro

description: Flask microservice demo with Prometheus & Grafana

tags:

- flask

- python

- prometheus

- demo

annotations:

github.com/project-slug: ramannkhanna2/raman-micro

backstage.io/techdocs-ref: dir:.

grafana/dashboard-url: http://54.233.49.51:3001/d/1f416562-37ea-4ef1-8df3-d6d1381b7f89/count

grafana/dashboard-selector: raman-micro

#grafana/dashboard-selector: 'tag=raman-micro'

#grafana/tag-selector: raman-micro

spec:

type: service

lifecycle: production

owner: dev-team

system: raman-platform

This ensures the plugin can pick up the correct dashboard to display.

**🌀 Step 5: Restart Backstage & Refresh Entity**

bash

CopyEdit

cd ~/backstage-app/raman-app

yarn dev

OR

Yarn start

* Than again register ur service on backstage .

https://github.com/ramannkhanna2/raman-micro/blob/main/catalog-info.yaml

→ Go to **Catalog → raman-micro**, then click the “Refresh” button (⋮ menu).

You should see a **“Grafana”** card with your dashboard preview and alerts (if any).

**✅ Summary: What You’ve Set Up**

* 🎯 Installed the **Grafana plugin**
* 🔧 Configured proxy and domain in app-config.yaml
* 🛠️ Added **Grafana cards** to service page layout
* ✅ Provided dashboard URL in your service metadata
* 🔄 Restarted Backstage and refreshed

**Lab 15: GitHub Actions Deploy Integration (Click-to-Deploy in Backstage)**

**✅ Goal**

Fully integrate **GitHub Actions plugin** into your Backstage portal such that:

* You can **see GitHub Actions run history**
* You can **trigger deployments manually** (click-to-deploy)
* All developers can inspect, retry, and audit pipelines inside Backstage

**🧩 PART 1: Install GitHub Actions Plugin in Backstage**

**1.1: Install Plugin in Frontend (packages/app)**

bash

CopyEdit

cd ~/backstage-app/raman-app

yarn –cwd packages/app add @backstage-community/plugin-github-actions

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

bash

CopyEdit

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/ramanapp/packages/backend/src# vi index.ts :

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

**🔐 PART 2: GitHub Oauth Setup**

**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://localhost>:3000 |
| Callback URL | <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 export these in your terminal (or .env):

bash

CopyEdit

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

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

**🔗 PART 3: GitHub Integration**

**3.1: Add GitHub integration config in app-config.yaml:**

yaml

CopyEdit

integrations:

github:

- host: github.com

apiBaseUrl: <https://api>.github.com

**🧠 PART 4: Annotate Your Service in Backstage Catalog**

Your service is raman-micro, already in GitHub.

Edit catalog-info.yaml inside your [raman-micro](https://github.com/ramannkhanna2/raman-micro) repo:

yaml

CopyEdit

apiVersion: backstage.io/v1alpha1

kind: Component

metadata:

name: raman-micro

description: Flask microservice

annotations:

github.com/project-slug: ramannkhanna2/raman-micro

spec:

type: service

lifecycle: production

owner: user:raman

✅ This annotation is **required**: github.com/project-slug

Push the updated catalog-info.yaml to GitHub.

**🧩 PART 5: Enable Plugin on the Entity Page**

Edit this file:

bash

CopyEdit

~/backstage-app/raman-app/packages/app/src/components/catalog/EntityPage.tsx

Add this import at the top:

tsx

CopyEdit

import {

EntityGithubActionsContent,

isGithubActionsAvailable,

} from ‘@backstage-community/plugin-github-actions’;

Now add a new tab under your EntityLayout:

const serviceEntityPage = (

<EntityLayout>

{/\* other tabs... \*/}

<EntityLayout.Route path=”/github-actions” title=”GitHub Actions” if={isGithubActionsAvailable}>

<EntityGithubActionsContent view=”cards” />

</EntityLayout.Route>

✅ This will show GitHub Actions as a **tab** in your service view.

**🚀 PART 6: Add Deploy Workflow to Your Repo**

In your [raman-micro](https://github.com/ramannkhanna2/raman-micro) repo, create:

bash

CopyEdit

.github/workflows/deploy.yml

Here’s a basic example:

yaml

CopyEdit

name: Deploy to Dev

on:

workflow\_dispatch: # click-to-deploy trigger

jobs:

deploy:

runs-on: ubuntu-latest

steps:

- name: Checkout Code

uses: actions/checkout@v3

- name: Build Docker Image

run: |

echo “Building Docker Image...”

docker build -t raman-micro:latest .

- name: Deploy (Simulated)

run: |

echo “Simulated deployment...”

✅ The key here is workflow\_dispatch – it **enables manual deploy** from Backstage.

**🧪 PART 7: Run & Test It**

**Start the app:**

bash

CopyEdit

cd ~/backstage-app/raman-app

yarn start

# In another terminal

yarn start-backend

OR

root@ip-172-31-14-172:~/backstage-app/ramanapp# yarn start

Starting app, backend

Loaded config from app-config.yaml

<i> [webpack-dev-server] Project is running a …

**✅ PART 8: Final Test**

1. Go to <http://localhost>:3000
2. Login using GitHub
3. Open your registered service: raman-micro
4. Click the tab: **GitHub Actions**
5. See the Deploy to Dev workflow
6. Click Run Workflow ➜ this will trigger deploy from Backstage (click-to-deploy)

**📌 Summary of What We Did**

| **Step** | **Description** | **Status** |
| --- | --- | --- |
| 🔌 Plugin Installed | Frontend & Backend | ✅ |
| 🔐 GitHub Oauth Setup | Auth + Provider Config | ✅ |
| 🏷️ Service Annotation | With github.com/project-slug | ✅ |
| ⚙️ Workflow Created | With workflow\_dispatch | ✅ |
| 🖼️ UI Integration | GitHub Actions Tab in EntityPage | ✅ |
| 🚀 Click-to-Deploy | Workflow triggers from Backstage | ✅ |

**🧩 Selft Lab Activity: Auto-Start Python App → Auto-Commit & Push → Auto-Trigger**

**🎯 Objective**

Use GitHub Actions to:

1. Start the Python app (python app/main.py)
2. Automatically stage, commit, and push the updated repo
3. Trigger the same or another CI/CD pipeline from the push

**🧪 What Participants Will Do**

1. Manually trigger the GitHub Actions workflow:  
   ✅ Navigate to **Actions > Auto Run Python App and Push > Run Workflow**
2. The workflow will:
   * Start Python app logic
   * Auto commit and push changes
   * GitHub will trigger ci.yml because of the push

**Lab 16 : Integrate KubeClusterUsingTerraformAnsible into Backstage for One-Click Cluster Creation**

* Already have the setup of kubeadm cluster creation in <https://github.com/ramannkhanna2/KubeClusterUsingTerraformAnsible.git>
* Thers the catalog-info.yaml inside .
* Register it as a component in backstage with below url :
* <https://github.com/ramannkhanna2/KubeClusterUsingTerraformAnsible/blob/main/catalog-info.yaml>
* No u shud see your github actions one click workflow to setup kubeadm 3 node Kubernetes cluster .



