Documentation on the process and how to trigger the service.

The Hello World application is a Python Flask application that runs inside a Docker container.

In the Backstage application, a new template has been created, enabling us to generate a new repository in the GitHub repository. The name of the template is "Example Hello World App." With the assistance of this template, we can create a new component that establishes a Git repository and migrates all files from helloworldbackstage/template/content to GitHub.

The file structure of the created GitHub repo will resemble the following, containing files Dockerfile, app.py, and catalog-info.yaml.

Now, a new component is created using the template, using the same name as that used while creating the GitHub repo.

The component now has access to GitHub in the interface, and CI/CD pipeline workflows are created with the help of GitHub Actions in the GitHub repo. Below is the GitHub Action configuration file committed in the Git repo:

```yaml
name: Docker Image CI

on:
  workflow_dispatch:
  push:
    branches: [ "master" ]
  pull_request:
    branches: [ "master" ]

jobs:

  build:

    runs-on: ubuntu-latest
```

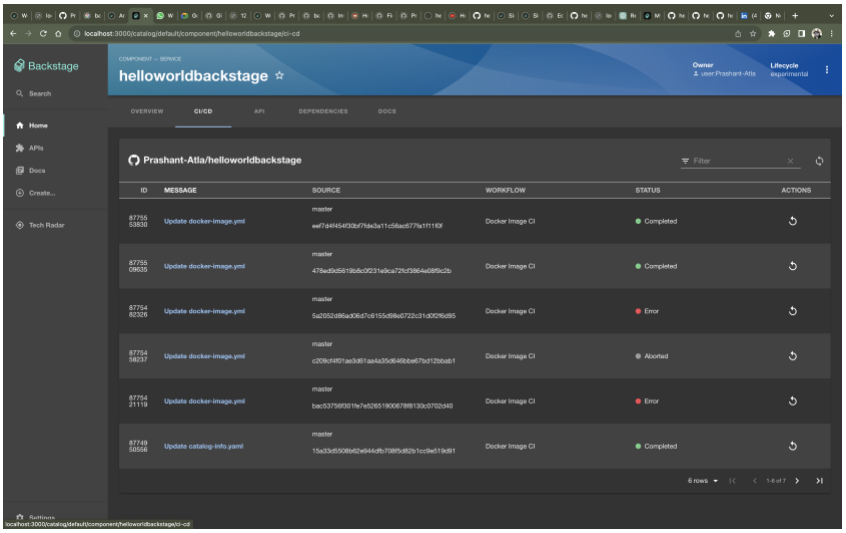
```
  steps:
  - name: Checkout code
    uses: actions/checkout@v3

  - name: Build the Docker image
    run:
      docker build . --file Dockerfile --tag hello-world-backstage:latest
  - name: Run the Docker image
    run:
      docker run -d -p 8000:8000 hello-world-backstage:latest

  - name: Wait for Docker container to start
    run: sleep 2 # Adjust the sleep duration as needed

  - name: Test the Docker container
    run: curl http://localhost:8000/
```

Whenever a new change is added or done in the file, the CI/CD pipeline is triggered, and the Docker image is built. Deployment in this solution is done on the same VM, i.e., through GitHub Actions. A simple curl command is used to display the output of the Docker container.



Yarn packages installed during development include:

1. GitHub Actions
2. Plugin Authentication Backend
3. Backend Module GitHub Provider

There is a "db" folder in the submission, necessary for running the component as it was created for persistence of the data. It needs to be placed in the location based on the configuration in `app-config.yaml`, in the database and directory section.

Changes made in the Backstage application:

In the `app-config.yaml` file, the database connection was modified from ":memory" to a custom location. To ensure persistent data storage, it is crucial to specify an absolute path. In this instance, the path has been set to `/Users/mac/Desktop`.

Additionally, within the integration section, a personal token is required for GitHub integration. This token should be added to the "integrations" section as follows:

```yaml
integrations:
  github:
    - host: github.com
      token: ghp_XXXXXXX
```

Authentication is set up for the app and is used in the CI/CD pipeline for the created component.

```yaml
auth:
  environment: development
  providers:
    github:
      development:
        clientId: [to be set in the OAuth app in GitHub]
        clientSecret: [to be set in the OAuth app in GitHub]
        signIn:
          resolvers:
            - resolver: usernameMatchingUserEntityName
```

Below are the changes made in the file to display in the catalog:

```yaml
- type: file
```

```
  target: ../../helloworldbackstage/entities.yaml

# Local example template
- type: file
  target: ../../helloworldbackstage/template/template.yaml
  rules:
    - allow: [Template]

# Local example organizational data
- type: file
  target: ../../helloworldbackstage/org.yaml
  rules:
    - allow: [User, Group]
```

In `org.yaml`, GitHub username is added under the metadata section.

For the purpose of custom component changes, modifications are made to `org.yaml`, `entities.yaml`, `template.yaml`, and `catalog-info.yaml`. The name of the component is "HelloWorld-Python-Template".

Challenges faced:

Totally new to the conecept of Backstage, needed sometime to understand on how it works and how to setup a new environment.

There was very less documentation on how to setup a dockerised application on backstage, given the time constraint had chance only to read up few annotations concepts in the backstage component.

plugin related to authentication was a bit confusing, had to fix so many problems to make it work. Found solutions in the github and open issues pages and some private documentation on web.

The current solution is based on github action, but I would say we can extend the solution to be able publish the image to docker hub or other registry and then can be pulled from there locally. Have found that using docker compose can help in local deployment of the solution.

AWS integration as some alternative for docker capabilities
For Backstage to deploy on AWS we need to install an appropriate plugin that allows to integrate to AWS some useful links to read can be found here
https://backstage.io/docs/reference/plugin-catalog-backend-module-aws/
It would be really interesting to deploy our services there

for replacing the docker solution with AWS we can make use of AWS tools that can be used an alternative for Docker applications

Container Orchestration:

- **Docker Capability**: Docker Compose or Docker Swarm for container orchestration.
- **AWS Replacement**: AWS offers multiple services for container orchestration, such as Amazon Elastic Container Service (ECS) or Amazon Elastic Kubernetes Service (EKS). These services manage the deployment, scaling, and monitoring of containers in a highly available and scalable manner.

2. **Container Registry**:

- **Docker Capability**: Docker Hub or Docker Trusted Registry for storing Docker images.
- **AWS Replacement**: Amazon Elastic Container Registry (ECR) is a fully-managed Docker container registry that makes it easy to store, manage, and deploy Docker container images. You can push your Docker images to ECR and then deploy them to ECS or EKS.

3. **Continuous Integration/Continuous Deployment (CI/CD)**:

- **Docker Capability**: Docker CLI or Docker Compose for building and deploying containers locally.
- **AWS Replacement**: GitHub Actions can be used for CI/CD workflows. You can use AWS CodeBuild for building Docker images, AWS CodePipeline for automating the deployment pipeline, and AWS CodeDeploy for deploying your application to ECS or EKS.

4. **Infrastructure as Code (IaC)**:

- **Docker Capability**: Docker Compose files or Dockerfiles for defining infrastructure.
- **AWS Replacement**: AWS CloudFormation or AWS CDK (Cloud Development Kit) for defining infrastructure as code. You can define your AWS resources, such as EC2 instances, load balancers, databases, and container services, using CloudFormation templates or CDK constructs.