Deploy a "Hello World" Application

In this guide, we will walk you through the process of deploying a "Hello World" application onto the GlueOps platform. We'll start from scratch and cover each step in detail to ensure you have a smooth deployment experience. You may see references to `antoniostacos`, this name is for demo purposes only and can be replaced with your actual project or company name where appropriate.

By the end of this guide, you will have deployed your antoniostacos application in a QA environment on the GlueOps platform.

Create a New Repository

NOTE

In the file structures shown below, we've used the `<` symbol next to files or folders to indicate those you'll be working on or creating during this tutorial.

JavaScript | Python | Ruby

```
app-antoniostacos <
├── Dockerfile
├── .github
│   └── workflows
│       ├── ghcr.yaml
│       └── qa-cd.yaml
├── app.js
├── README.md
```

Go to GitHub and create a new repository. When naming your repository, you can use a format like `app-yourprojectname`, for example: `app-antoniostacos`.

Add your application code and Dockerfile

INFO

Marked below are the files we will be adding:

JavaScript Python Ruby

```
app-antoniostacos
├── Dockerfile <
├── .github
│   └── workflows
│       ├── ghcr.yaml
│       └── qa-cd.yaml
├── app.js <
├── README.md
```

Application Code

Within your repo create the file that contains your code:

JavaScript Python Ruby

<MY_REPO>/app.js

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
    const greeting = process.env.GREETING_MESSAGE || 'Hello, World!';
    res.send(greeting);
});

app.listen(8080, () => {
    console.log('Server running on http://localhost:8080');
});
```

Save the file and commit your changes.

Dockerfile

Next, within your application repository, create a new file named `Dockerfile`. Populate this file using the template provided below based on your chosen programming language.

JavaScript Python Ruby

<MY_REPO>/Dockerfile

```
FROM node:18-alpine

WORKDIR /app

COPY . /app

RUN npm install express

CMD ["node", "app.js"]
```

Save the file and commit your changes.

Add CI to publish a Docker image to GitHub Container Registry

> **INFO**
>
> Marked below are the files we will be adding:
>
> JavaScript Python Ruby
>
> ```
> app-antoniostacos
> ├── Dockerfile
> ├── .github
> │   └── workflows
> │       ├── ghcr.yaml <
> │       └── qa-cd.yaml
> ├── app.js
> ├── README.md
> ```

By creating and saving the following YAML configuration, you're setting up a GitHub Action that will automatically publish a Docker image of your application to the GitHub Container Registry (GHCR.io). This will allow the GlueOps platform to use the latest version of your app in

it's deployments. As a happy path, we have provided this custom action to push Docker images to your GitHub Container Registry (GHCR.io).

To use it, Simply create the file below

<MY_REPO>/.github/workflows/ghcr.yaml

```
name: Publish to GHCR.io

on: [push]

jobs:
  build_tag_push_to_ghcr:
    runs-on: ubuntu-latest
    steps:
      - name: Build, Tag and Push Docker Image to GHCR
        uses: GlueOps/github-actions-build-push-containers@main
```

Save the file and commit your changes and push up your changes.

> **TIP**
>
> Once you push up your changes visit your github repository actions page to view the status. You can find it at: https://github.com/<MY_ORG>/<MY_REPO>/actions. If all the circles next to your actions are "Green", it indicates that the processes completed successfully. If you see any "Red" circles then you may need to revisit the steps above before continuing. In the end you should see an artifact published to: https://github.com/<MY_ORG>/<MY_REPO>/packages.

Let's deploy your app!

> **INFO**
>
> **Deployment Configuration Repository:**
>
> Think of this repository as your application's deployment instruction manual. It contains all the essential rules and settings that determine how and where your application should be launched. You don't need to delve into the nitty-gritty of these configurations. Just follow the steps below, and our system will handle the deployment seamlessly.

> **IMPORTANT**
>
> The exact name and location of this "deployment configurations" repository will be provided by your Platform Administrators.

```
deployment-configurations
├── apps
│   └── app-antoniostacos
│       └── envs
│           └── qa
│               └── values.yaml <
```

Now, within your "deployment configurations" repository, create the following file:

apps/<MY_REPO>/envs/qa/values.yaml

```yaml
image:
  repository: '<MY_ORG>/<MY_REPO>'
  registry: ghcr.io
  pullPolicy: Always
  port: 8080
  tag: main
service:
  enabled: true
deployment:
  replicas: 1
  enabled: true
  imagePullPolicy: Always
  imagePullSecrets: <CONTAINER_REGISTRY_CREDENTIALS>
  resources:
    requests:
      cpu: 100m
      memory: 128Mi

ingress:
  enabled: true
  ingressClassName: public
  entries:
    - name: public
      hosts:
        - hostname: '<MY_APP_NAME>-qa.apps.<MY_CAPTAIN_DOMAIN>'
```

TIP

Ensure you replace the placeholders appropriately:

- `<MY_ORG>` and `<MY_REPO>` with your GitHub organization and application repository names.
- `<MY_APP_NAME>` with your chosen app name.
- `<MY_CAPTAIN_DOMAIN>` with your assigned captain domain, provided by the Platform Administrators.
- `<CONTAINER_REGISTRY_CREDENTIALS>` if your image is in a private registry you will need to specify a value provided by the Platform Administrators. Otherwise use `nil`.

Once you've done the above, commit your changes to the deployment configurations repository and push the changes. In a short while, you should be able to access your app at the URL: `https://<MY_APP_NAME>-qa.apps.<MY_CAPTAIN_DOMAIN>`