**Backstage Component Card for GKE Autopilot Deployment**

**1. Set Up Your Backstage Environment**

Ensure you have a **Backstage application** set up. If not, install Backstage using the command:

npx @backstage/create-app

**2. Create a New Backstage Plugin**

Navigate to your Backstage application directory and create a new plugin:

cd packages/app

yarn backstage-cli create-plugin

Provide a name for your plugin (e.g., gke-autopilot-plugin).

**3. Develop the Component Card**

Create a UI component to display the deployment functionality.

**File: plugins/gke-autopilot-plugin/src/components/GkeAutopilotCard.tsx**

import React from 'react';

import { Card, CardHeader, CardContent, CardActions, Button } from '@material-ui/core';

import { useApi } from '@backstage/core-plugin-api';

import { gkeAutopilotApiRef } from '../api/GkeAutopilotApi';

export const GkeAutopilotCard = () => {

const gkeAutopilotApi = useApi(gkeAutopilotApiRef);

const handleDeploy = async () => {

try {

await gkeAutopilotApi.createClusterAndDeployNginx();

alert('GKE Autopilot cluster created and NGINX deployed successfully!');

} catch (error) {

alert(`Error: ${error.message}`);

}

};

return (

<Card>

<CardHeader title="GKE Autopilot Cluster" />

<CardContent>

<p>Click the button below to spin up a GKE Autopilot cluster and deploy NGINX.</p>

</CardContent>

<CardActions>

<Button variant="contained" color="primary" onClick={handleDeploy}>

Deploy NGINX on GKE Autopilot

</Button>

</CardActions>

</Card>

);

};

**4. Create the API Client**

Define an API to interact with **Google Cloud** and **Kubernetes**.

**File: plugins/gke-autopilot-plugin/src/api/GkeAutopilotApi.ts**

import { ContainerClient } from '@google-cloud/container';

import { CoreV1Api, KubeConfig } from '@kubernetes/client-node';

import { createApiRef } from '@backstage/core-plugin-api';

export const gkeAutopilotApiRef = createApiRef<GkeAutopilotApi>({

id: 'plugin.gke-autopilot.service',

});

export class GkeAutopilotApi {

private containerClient: ContainerClient;

private kubeConfig: KubeConfig;

constructor() {

this.containerClient = new ContainerClient();

this.kubeConfig = new KubeConfig();

this.kubeConfig.loadFromDefault();

}

async createClusterAndDeployNginx() {

const projectId = 'your-project-id';

const zone = 'us-central1';

const clusterName = 'backstage-cluster';

console.log('Creating GKE Autopilot cluster...');

const [operation] = await this.containerClient.createCluster({

parent: `projects/${projectId}/locations/${zone}`,

cluster: { name: clusterName, autopilot: { enabled: true } },

});

await operation.promise();

console.log('GKE Autopilot cluster created successfully!');

console.log('Deploying NGINX...');

const coreV1Api = this.kubeConfig.makeApiClient(CoreV1Api);

const nginxDeployment = {

metadata: { name: 'nginx-deployment' },

spec: {

replicas: 1,

template: {

metadata: { labels: { app: 'nginx' } },

spec: {

containers: [{ name: 'nginx', image: 'nginx:latest', ports: [{ containerPort: 80 }] }],

},

},

},

};

await coreV1Api.createNamespacedDeployment('default', nginxDeployment);

console.log('NGINX deployed successfully!');

}

}

**5. Register the Plugin**

**File: plugins/gke-autopilot-plugin/src/plugin.ts**

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

import { GkeAutopilotApi, gkeAutopilotApiRef } from './api/GkeAutopilotApi';

export const gkeAutopilotPlugin = createPlugin({

id: 'gke-autopilot',

apis: [

createApiFactory({

api: gkeAutopilotApiRef,

deps: {},

factory: () => new GkeAutopilotApi(),

}),

],

});

export const GkeAutopilotPage = gkeAutopilotPlugin.provide(

createComponentExtension({

name: 'GkeAutopilotPage',

component: {

lazy: () => import('./components/GkeAutopilotCard').then(m => m.GkeAutopilotCard),

},

}),

);

**File: packages/app/src/App.tsx**

Add a route for your plugin:

import { gkeAutopilotPlugin } from '@internal/plugin-gke-autopilot';

<Route path="/gke-autopilot" element={<GkeAutopilotCard />} />

**6. Implement Google Cloud and Kubernetes Logic**

The logic for creating a GKE cluster and deploying NGINX is implemented in GkeAutopilotApi.ts. Ensure you have:

* A **Google Cloud Service Account** with Kubernetes Engine permissions.
* **kubectl** configured with credentials to deploy resources.

**7. Add Authentication**

Ensure your **Google Cloud credentials** are available in your Backstage application:

* Use a **service account key** stored in a **Google Cloud Secret Manager**.
* Authenticate using **Workload Identity Federation**.

**8. Test the Plugin**

Start your Backstage app and navigate to http://localhost:3000/gke-autopilot.

Run the development server: yarn dev

Click the **"Deploy NGINX on GKE Autopilot"** button and verify the deployment.

# ****Backstage Harness Integration for NGINX-Based App Deployment on GKE Autopilot****

How to integrate **Backstage** with **Harness** for a simple **"Hello World"** NGINX-based app deployment on **Google Kubernetes Engine (GKE) Autopilot** and enable refreshing using a card in the Backstage UI.

## ****1. Prerequisites****

Ensure you have the following:

* **Backstage Instance**: A running Backstage instance.
* **Harness Account**: A Harness account configured for CI/CD.
* **GKE Autopilot Cluster**: A created GKE Autopilot cluster with access.
* **NGINX Hello World App**: A simple NGINX-based **"Hello World"** app (e.g., a Docker image).

## ****2. Set Up Harness for CI/CD****

### ****Create a Harness Project****

1. Log in to Harness and create a new project.
2. Set up **connectors** for:
   * GKE cluster
   * Docker registry (e.g., Docker Hub, GCR)

### ****Create a CI/CD Pipeline****

Define a pipeline with the following stages:

* **Build**: Build and push the Docker image.
* **Deploy**: Deploy the app to GKE Autopilot using a Kubernetes manifest or Helm chart.

### ****Example Kubernetes Manifest****

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-hello-world

spec:

replicas: 1

selector:

matchLabels:

app: nginx-hello-world

template:

metadata:

labels:

app: nginx-hello-world

spec:

containers:

- name: nginx

image: nginx:latest

ports:

- containerPort: 80

### ****Test the Pipeline****

Run the pipeline and verify that the app is successfully built and deployed to GKE Autopilot.

## ****3. Integrate Harness with Backstage****

### ****Install Harness Backstage Plugin****

Run the following command to install the Harness plugin:

yarn add @harnessio/backstage-plugin

### ****Configure the Plugin****

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

harness:

baseUrl: https://app.harness.io

apiKey: YOUR\_HARNESS\_API\_KEY

### ****Add a Harness Card to the Backstage UI****

Edit your **catalog-info.yaml** file to include Harness pipeline details:

apiVersion: backstage.io/v1alpha1

kind: Component

metadata:

name: nginx-hello-world

description: A simple NGINX Hello World app

annotations:

harness.io/pipeline-id: YOUR\_PIPELINE\_ID

spec:

type: service

lifecycle: production

owner: team-a

### ****Display the Harness Card****

Modify your app's overview page in Backstage to include the Harness card:

import { HarnessPipelineWidget } from '@harnessio/backstage-plugin';

const OverviewPage = () => (

<Grid container spacing={3}>

<Grid item xs={12}>

<HarnessPipelineWidget />

</Grid>

</Grid>

);

## ****4. Enable Refreshing Using a Card****

### ****Add a Refresh Button****

Add a button to trigger the Harness pipeline:

<Button

variant="contained"

color="primary"

onClick={() => {

// Trigger Harness pipeline run

harnessClient.runPipeline(YOUR\_PIPELINE\_ID);

}}

>

Refresh Deployment

</Button>

### ****Poll for Pipeline Status****

Use the Harness plugin's API to dynamically update the status of the pipeline card.

## ****5. Verify the Integration****

1. Open **Backstage UI** and navigate to your app’s overview page.
2. Ensure the **Harness card** displays the pipeline status.
3. Click the **refresh button** to trigger a new pipeline run.
4. Verify the app deployment in **GKE Autopilot**.

## ****How to Run This Integration****

This step-by-step guide assumes familiarity with **Kubernetes, Harness, Backstage, and GKE**.

### ****Step 1: Set Up Your NGINX Hello World App****

#### **Create a Dockerfile**

dockerfile

CopyEdit

FROM nginx:latest

COPY index.html /usr/share/nginx/html/index.html

#### **Create an** index.html **File**

<!DOCTYPE html>

<html>

<head>

<title>Hello World</title>

</head>

<body>

<h1>Hello, World!</h1>

</body>

</html>

#### **Build and Push the Docker Image**

docker build -t your-dockerhub-username/nginx-hello-world:latest .

docker push your-dockerhub-username/nginx-hello-world:latest

### ****Step 2: Set Up GKE Autopilot****

#### **Create a GKE Autopilot Cluster**

1. Go to **Google Cloud Console**.
2. Navigate to **Kubernetes Engine > Clusters**.
3. Click **Create Cluster** and select **Autopilot mode**.
4. Configure the cluster (e.g., **name, region**) and click **Create**.

#### **Configure** kubectl

gcloud container clusters get-credentials YOUR\_CLUSTER\_NAME --region YOUR\_REGION

### ****Step 3: Set Up Harness****

1. **Sign up at Harness**.
2. **Create a project** for your app.
3. **Set up connectors** for:
   * Docker Registry
   * Kubernetes Cluster (GKE Autopilot)
4. **Create a pipeline** with **Build & Deploy** stages.

#### **Example Kubernetes Manifest**

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-hello-world

spec:

replicas: 1

selector:

matchLabels:

app: nginx-hello-world

template:

metadata:

labels:

app: nginx-hello-world

spec:

containers:

- name: nginx

image: your-dockerhub-username/nginx-hello-world:latest

ports:

- containerPort: 80

#### **Run the Pipeline**

Trigger the pipeline and verify that the app is deployed successfully.

### ****Step 4: Set Up Backstage****

1. **Install Backstage** (if not already installed).
2. **Install the Harness Plugin**

yarn add @harnessio/backstage-plugin

1. **Configure the Plugin in app-config.yaml**

harness:

baseUrl: https://app.harness.io

apiKey: YOUR\_HARNESS\_API\_KEY

1. **Add Your App to the Backstage Catalog (catalog-info.yaml)**

apiVersion: backstage.io/v1alpha1

kind: Component

metadata:

name: nginx-hello-world

annotations:

harness.io/pipeline-id: YOUR\_PIPELINE\_ID

### ****Step 5: Enable Refreshment Using a Card****

#### **Add a Refresh Button**

<Button

variant="contained"

color="primary"

onClick={() => {

harnessClient.runPipeline(YOUR\_PIPELINE\_ID);

}}

>

Refresh Deployment

</Button>