To create a simple Backstage UI with cards that allow you to provision Kubernetes clusters (GKE, AKS, EKS) with a single click, you can use **Backstage plugins** and **Terraform** for infrastructure provisioning. Below is a step-by-step guide to achieve this:

**Step 1: Set Up Backstage**

1. **Install Backstage**:  
   If you haven’t already, set up Backstage by following the [official Backstage getting started guide](https://backstage.io/docs/getting-started/).
2. **Create a Custom Plugin**:  
   Create a new plugin to handle the UI and backend logic for provisioning Kubernetes clusters.

cd packages/app

yarn create-plugin

Name the plugin kubernetes-provisioner.

**Step 2: Create Cards in the Backstage UI**

1. Open the packages/app/src/App.tsx file and add a new route for the Kubernetes provisioner plugin:

import { KubernetesProvisionerPage } from '@internal/plugin-kubernetes-provisioner';

const routes = (

<FlatRoutes>

<Route path="/kubernetes-provisioner" element={<KubernetesProvisionerPage />} />

{/\* Other routes \*/}

</FlatRoutes>

);

1. Create the KubernetesProvisionerPage component in your plugin:

// packages/plugin-kubernetes-provisioner/src/components/KubernetesProvisionerPage.tsx

import React from 'react';

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

export const KubernetesProvisionerPage = () => (

<div>

<h1>Provision Kubernetes Clusters</h1>

<div style={{ display: 'flex', gap: '16px' }}>

<Card>

<CardHeader title="GKE" />

<CardContent>

<Button variant="contained" color="primary" onClick={() => provisionCluster('gke')}>

Create GKE Cluster

</Button>

</CardContent>

</Card>

<Card>

<CardHeader title="AKS" />

<CardContent>

<Button variant="contained" color="primary" onClick={() => provisionCluster('aks')}>

Create AKS Cluster

</Button>

</CardContent>

</Card>

<Card>

<CardHeader title="EKS" />

<CardContent>

<Button variant="contained" color="primary" onClick={() => provisionCluster('eks')}>

Create EKS Cluster

</Button>

</CardContent>

</Card>

</div>

</div>

);

const provisionCluster = async (provider: string) => {

const response = await fetch(`/api/kubernetes-provisioner/provision?provider=${provider}`);

if (response.ok) {

alert(`${provider.toUpperCase()} cluster provisioned successfully!`);

} else {

alert(`Failed to provision ${provider.toUpperCase()} cluster.`);

}

};

**Step 3: Set Up the Backend API**

1. Create a backend API to handle cluster provisioning:

// packages/backend/src/plugins/kubernetesProvisioner.ts

import { Router } from 'express';

import { exec } from 'child\_process';

export default async function createPlugin(): Promise<Router> {

const router = Router();

router.get('/provision', async (req, res) => {

const { provider } = req.query;

if (!provider) {

return res.status(400).json({ error: 'Provider is required' });

}

try {

// Call Terraform or a script to provision the cluster

exec(`./scripts/provision-${provider}.sh`, (error, stdout, stderr) => {

if (error) {

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

return res.status(500).json({ error: 'Failed to provision cluster' });

}

console.log(`stdout: ${stdout}`);

console.error(`stderr: ${stderr}`);

res.status(200).json({ message: `${provider} cluster provisioned successfully` });

});

} catch (error) {

console.error(error);

res.status(500).json({ error: 'Internal server error' });

}

});

return router;

}

1. Register the plugin in the backend:

// packages/backend/src/index.ts

import kubernetesProvisioner from './plugins/kubernetesProvisioner';

const kubernetesProvisionerRouter = await kubernetesProvisioner();

apiRouter.use('/kubernetes-provisioner', kubernetesProvisionerRouter);

**Step 4: Write Provisioning Scripts**

1. Create a scripts directory in the root of your Backstage project:

mkdir scripts

1. Write provisioning scripts for each provider using Terraform or CLI commands.

**Example: provision-gke.sh**

#!/bin/bash

gcloud container clusters create backstage-gke-cluster \

--zone us-central1-a \

--num-nodes 1 \

--machine-type e2-medium

**Example: provision-aks.sh**

#!/bin/bash

az aks create \

--resource-group myResourceGroup \

--name backstage-aks-cluster \

--node-count 1 \

--enable-addons monitoring \

--generate-ssh-keys

**Example: provision-eks.sh**

#!/bin/bash

eksctl create cluster \

--name backstage-eks-cluster \

--region us-west-2 \

--nodegroup-name standard-workers \

--node-type t3.medium \

--nodes 1 \

--nodes-min 1 \

--nodes-max 1

1. Make the scripts executable:

chmod +x scripts/provision-\*.sh

**Step 5: Test the Workflow**

1. Start the Backstage app:

yarn start

1. Navigate to the Kubernetes provisioner page (/kubernetes-provisioner).
2. Click the buttons to provision GKE, AKS, or EKS clusters.
3. Check your cloud provider console to verify the clusters are created.

**Step 6: Deploy to Production**

1. Update the provisioning scripts to include production configurations (e.g., larger node sizes, additional regions).
2. Deploy Backstage to your production environment (e.g., GKE, AKS, EKS).

**Using Terraform**

**Step 1: Set Up Backstage**

1. **Install Backstage**:  
   Follow the [official Backstage getting started guide](https://backstage.io/docs/getting-started/) to set up Backstage.
2. **Create a Custom Plugin**:  
   Create a new plugin to handle the UI and backend logic for provisioning Kubernetes clusters.

cd packages/app

yarn create-plugin

Name the plugin kubernetes-provisioner.

**Step 2: Create Cards in the Backstage UI**

1. Open the packages/app/src/App.tsx file and add a new route for the Kubernetes provisioner plugin:

import { KubernetesProvisionerPage } from '@internal/plugin-kubernetes-provisioner';

const routes = (

<FlatRoutes>

<Route path="/kubernetes-provisioner" element={<KubernetesProvisionerPage />} />

{/\* Other routes \*/}

</FlatRoutes>

);

1. Create the KubernetesProvisionerPage component in your plugin:

// packages/plugin-kubernetes-provisioner/src/components/KubernetesProvisionerPage.tsx

import React from 'react';

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

export const KubernetesProvisionerPage = () => (

<div>

<h1>Provision Kubernetes Clusters</h1>

<div style={{ display: 'flex', gap: '16px' }}>

<Card>

<CardHeader title="GKE" />

<CardContent>

<Button variant="contained" color="primary" onClick={() => provisionCluster('gke')}>

Create GKE Cluster

</Button>

</CardContent>

</Card>

<Card>

<CardHeader title="AKS" />

<CardContent>

<Button variant="contained" color="primary" onClick={() => provisionCluster('aks')}>

Create AKS Cluster

</Button>

</CardContent>

</Card>

<Card>

<CardHeader title="EKS" />

<CardContent>

<Button variant="contained" color="primary" onClick={() => provisionCluster('eks')}>

Create EKS Cluster

</Button>

</CardContent>

</Card>

</div>

</div>

);

const provisionCluster = async (provider: string) => {

const response = await fetch(`/api/kubernetes-provisioner/provision?provider=${provider}`);

if (response.ok) {

alert(`${provider.toUpperCase()} cluster provisioned successfully!`);

} else {

alert(`Failed to provision ${provider.toUpperCase()} cluster.`);

}

};

**Step 3: Set Up Terraform for Kubernetes Provisioning**

1. **Install Terraform**:  
   Download and install Terraform from [terraform.io](https://www.terraform.io/downloads.html).
2. **Create Terraform Configurations**:  
   Create a directory for Terraform configurations:

mkdir -p terraform/{gke,aks,eks}

1. **GKE Terraform Configuration**:  
   Create terraform/gke/main.tf:

provider "google" {

project = "your-gcp-project-id"

region = "us-central1"

}

resource "google\_container\_cluster" "primary" {

name = "backstage-gke-cluster"

location = "us-central1-a"

remove\_default\_node\_pool = true

initial\_node\_count = 1

}

resource "google\_container\_node\_pool" "primary\_nodes" {

name = "backstage-node-pool"

location = "us-central1-a"

cluster = google\_container\_cluster.primary.name

node\_count = 1

node\_config {

machine\_type = "e2-medium"

}

}

1. **AKS Terraform Configuration**:  
   Create terraform/aks/main.tf:

provider "azurerm" {

features {}

}

resource "azurerm\_resource\_group" "rg" {

name = "backstage-rg"

location = "East US"

}

resource "azurerm\_kubernetes\_cluster" "aks" {

name = "backstage-aks-cluster"

location = azurerm\_resource\_group.rg.location

resource\_group\_name = azurerm\_resource\_group.rg.name

dns\_prefix = "backstageaks"

default\_node\_pool {

name = "default"

node\_count = 1

vm\_size = "Standard\_D2\_v2"

}

identity {

type = "SystemAssigned"

}

}

1. **EKS Terraform Configuration**:  
   Create terraform/eks/main.tf:

provider "aws" {

region = "us-west-2"

}

resource "aws\_eks\_cluster" "eks" {

name = "backstage-eks-cluster"

role\_arn = aws\_iam\_role.eks\_cluster.arn

vpc\_config {

subnet\_ids = ["subnet-xxxxxxxx", "subnet-yyyyyyyy"]

}

}

resource "aws\_iam\_role" "eks\_cluster" {

name = "eks-cluster-role"

assume\_role\_policy = jsonencode({

Version = "2012-10-17"

Statement = [

{

Action = "sts:AssumeRole"

Effect = "Allow"

Principal = {

Service = "eks.amazonaws.com"

}

}

]

})

}

**Step 4: Set Up the Backend API**

1. Create a backend API to handle cluster provisioning:

// packages/backend/src/plugins/kubernetesProvisioner.ts

import { Router } from 'express';

import { exec } from 'child\_process';

export default async function createPlugin(): Promise<Router> {

const router = Router();

router.get('/provision', async (req, res) => {

const { provider } = req.query;

if (!provider) {

return res.status(400).json({ error: 'Provider is required' });

}

try {

exec(`terraform -chdir=terraform/${provider} init && terraform -chdir=terraform/${provider} apply -auto-approve`, (error, stdout, stderr) => {

if (error) {

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

return res.status(500).json({ error: 'Failed to provision cluster' });

}

console.log(`stdout: ${stdout}`);

console.error(`stderr: ${stderr}`);

res.status(200).json({ message: `${provider} cluster provisioned successfully` });

});

} catch (error) {

console.error(error);

res.status(500).json({ error: 'Internal server error' });

}

});

return router;

}

1. Register the plugin in the backend:

// packages/backend/src/index.ts

import kubernetesProvisioner from './plugins/kubernetesProvisioner';

const kubernetesProvisionerRouter = await kubernetesProvisioner();

apiRouter.use('/kubernetes-provisioner', kubernetesProvisionerRouter);

**Step 5: Test the Workflow**

1. Start the Backstage app:

yarn start

1. Navigate to the Kubernetes provisioner page (/kubernetes-provisioner).
2. Click the buttons to provision GKE, AKS, or EKS clusters.
3. Check your cloud provider console to verify the clusters are created.