# ****Setting Up Nginx on Minikube****

## ****Step 1: Install Minikube – optional [if not done]****

If you haven't already installed Minikube, follow the official documentation for your operating system:

* **Linux**: Minikube Installation Guide for Linux
* **macOS**: Minikube Installation Guide for macOS
* **Windows**: Minikube Installation Guide for Windows

## ****Step 2: Start Minikube****

Once Minikube is installed, start it using the following command:

minikube start

This command will create a single-node Kubernetes cluster locally.

## ****Step 3: Verify Minikube Status****

Ensure that Minikube is running correctly:

minikube status

You should see output indicating that Minikube is running.

## ****Step 4: Deploy Nginx on Minikube****

You can deploy Nginx using a Kubernetes deployment. Create a file named **nginx-deployment.yaml** with the following content:

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

replicas: 2

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:latest

ports:

- containerPort: 80

Apply the deployment using the following command:

kubectl apply -f nginx-deployment.yaml

## ****Step 5: Expose the Nginx Deployment****

To access the Nginx service, you need to expose it. Create a file named **nginx-service.yaml** with the following content:

apiVersion: v1

kind: Service

metadata:

name: nginx-service

spec:

selector:

app: nginx

ports:

- protocol: TCP

port: 80

targetPort: 80

type: NodePort

Apply the service using the following command:

kubectl apply -f nginx-service.yaml

## ****Step 6: Access the Nginx Service****

To access the Nginx service, use the Minikube IP and the NodePort assigned to the service.

### **Get the Minikube IP:**

minikube ip

### **Get the NodePort assigned to the Nginx service:**

kubectl get svc nginx-service

Look for the PORT(S) column, which will show something like **80:3XXXX/TCP**. The **3XXXX** is the NodePort.

Now, access Nginx by navigating to:

http://<minikube-ip>:<NodePort>

in your web browser.

## ****Step 7: Verify Nginx is Running****

You should see the default Nginx welcome page in your browser, indicating that Nginx is running successfully.

## ****Step 8: Clean Up (Optional)****

If you want to remove the resources you created, delete the deployment and service:

kubectl delete -f nginx-deployment.yaml

kubectl delete -f nginx-service.yaml

# ****Templates - Step 1: Create a GitHub Repository****

Before creating a Backstage template, **create a GitHub repo** where the template and source files will be stored.

1️⃣ **Create a GitHub repository** (e.g., backstage-templates)  
2️⃣ **Clone the repo locally:**

git clone https://github.com/my-org/backstage-templates.git

cd backstage-templates

# ****🔹 Example 1: Simple "Hello World" Repository Template****

## ****📌 Step 1: Create a New Template****

mkdir hello-world-template

cd hello-world-template

## ****📌 Step 2: Create**** template.yaml

Create hello-world-template/template.yaml:

apiVersion: scaffolder.backstage.io/v1beta3

kind: Template

metadata:

name: hello-world-template

title: Hello World Repository

description: A simple starter repository

spec:

owner: user@example.com

type: service

parameters:

- title: Provide Repository Information

required: [repoName]

properties:

repoName:

title: Repository Name

type: string

steps:

- id: create-repo

name: Create Repository

action: github:repo:create

input:

owner: my-org

repo: ${{ parameters.repoName }}

visibility: public

- id: fetch-template

name: Clone and Push Code

action: fetch:template

input:

url: ./template-content

targetPath: ./result

- id: publish

name: Publish to GitHub

action: publish:github

input:

repoUrl: github.com/my-org/${{ parameters.repoName }}

branch: main

output:

links:

- title: Repository

url: https://github.com/my-org/${{ parameters.repoName }}

## ****📌 Step 3: Add Template Source Code****

Create a template-content/ directory:

mkdir template-content

cd template-content

Create a simple index.js:

console.log("Hello, Backstage!");

Create a package.json:

{

"name": "hello-world",

"version": "1.0.0",

"main": "index.js",

"scripts": {

"start": "node index.js"

}

}

## ****📌 Step 4: Push to GitHub****

1️⃣ Navigate back to the repo root:

cd ../../

2️⃣ Add and commit changes:

git add .

git commit -m "Add Hello World Template"

3️⃣ Push to GitHub:

git push origin main

## ****📌 Step 5: Register the Template in Backstage****

1️⃣ Navigate to **Backstage UI → Create → Register Template**  
2️⃣ Enter the GitHub URL of template.yaml:

https://github.com/my-org/backstage-templates/blob/main/hello-world-template/template.yaml

3️⃣ Click **Analyze** → **Import**

📍 **View in Backstage UI** → /create → **"Hello World Repository"**

# ****📌 Template 2: React Frontend App****

This template generates a **React application** and pushes it to GitHub.

### **📌 Step 1: Create** template.yaml

Inside backstage-templates/react-template/template.yaml:

apiVersion: scaffolder.backstage.io/v1beta3

kind: Template

metadata:

name: react-template

title: React App

description: A simple React frontend application

spec:

owner: user@example.com

type: website

parameters:

- title: Provide Repository Information

required: [repoName]

properties:

repoName:

title: Repository Name

type: string

steps:

- id: create-repo

name: Create Repository

action: github:repo:create

input:

owner: my-org

repo: ${{ parameters.repoName }}

visibility: public

- id: fetch-template

name: Clone and Push Code

action: fetch:template

input:

url: ./template-content

targetPath: ./result

- id: publish

name: Publish to GitHub

action: publish:github

input:

repoUrl: github.com/my-org/${{ parameters.repoName }}

branch: main

output:

links:

- title: Repository

url: https://github.com/my-org/${{ parameters.repoName }}

### **📌 Step 2: Add React App Source Code**

Inside react-template/template-content/, create:

**📂 src/App.js**

import React from 'react';

function App() {

return (

<div>

<h1>Hello, Backstage React App!</h1>

</div>

);

}

export default App;

**📂 package.json**

{

"name": "react-backstage-app",

"version": "1.0.0",

"dependencies": {

"react": "^18.0.0",

"react-dom": "^18.0.0"

},

"scripts": {

"start": "react-scripts start"

}

}

# ****📌 Template 2: Angular Frontend App****

This template generates a **basic Angular application**.

### **📌 Step 1: Create** template.yaml

Inside backstage-templates/angular-template/template.yaml:

apiVersion: scaffolder.backstage.io/v1beta3

kind: Template

metadata:

name: angular-template

title: Angular App

description: A simple Angular frontend application

spec:

owner: user@example.com

type: website

parameters:

- title: Provide Repository Information

required: [repoName]

properties:

repoName:

title: Repository Name

type: string

steps:

- id: create-repo

name: Create Repository

action: github:repo:create

input:

owner: my-org

repo: ${{ parameters.repoName }}

visibility: public

- id: fetch-template

name: Clone and Push Code

action: fetch:template

input:

url: ./template-content

targetPath: ./result

- id: publish

name: Publish to GitHub

action: publish:github

input:

repoUrl: github.com/my-org/${{ parameters.repoName }}

branch: main

output:

links:

- title: Repository

url: https://github.com/my-org/${{ parameters.repoName }}

### **📌 Step 2: Add Angular App Source Code**

Inside angular-template/template-content/, create:

**📂 src/app/app.component.ts**

import { Component } from '@angular/core';

@Component({

selector: 'app-root',

template: `<h1>Hello, Backstage Angular App!</h1>`,

styleUrls: ['./app.component.css']

})

export class AppComponent { }

**📂 package.json**

{

"name": "angular-backstage-app",

"version": "1.0.0",

"dependencies": {

"@angular/core": "^13.0.0",

"@angular/cli": "^13.0.0"

},

"scripts": {

"start": "ng serve"

}

}

# ****📌 Template 3: Node.js Express API****

This template creates a **simple Node.js Express API**.

### **📌 Step 1: Create** template.yaml

Inside backstage-templates/node-template/template.yaml:

apiVersion: scaffolder.backstage.io/v1beta3

kind: Template

metadata:

name: node-template

title: Node.js API

description: A simple Node.js API with Express

spec:

owner: user@example.com

type: service

parameters:

- title: Provide Repository Information

required: [repoName]

properties:

repoName:

title: Repository Name

type: string

steps:

- id: create-repo

name: Create Repository

action: github:repo:create

input:

owner: my-org

repo: ${{ parameters.repoName }}

visibility: public

- id: fetch-template

name: Clone and Push Code

action: fetch:template

input:

url: ./template-content

targetPath: ./result

- id: publish

name: Publish to GitHub

action: publish:github

input:

repoUrl: github.com/my-org/${{ parameters.repoName }}

branch: main

output:

links:

- title: Repository

url: https://github.com/my-org/${{ parameters.repoName }}

### **📌 Step 2: Add Node.js API Source Code**

Inside node-template/template-content/, create:

**📂 server.js**

const express = require('express');

const app = express();

app.get('/api/hello', (req, res) => {

res.json({ message: "Hello, Backstage!" });

});

app.listen(3000, () => console.log('Server running on port 3000'));

**📂 package.json**

{

"name": "node-backstage-app",

"version": "1.0.0",

"dependencies": {

"express": "^4.18.0"

},

"scripts": {

"start": "node server.js"

}

}

# ****🔹 Step 3: Push All Templates to GitHub****

1️⃣ Go to the root of backstage-templates repo:

cd ~/backstage-templates

2️⃣ Add and commit changes:

git add .

git commit -m "Add React, Angular, and Node.js templates"

3️⃣ Push to GitHub:

git push origin main

# ****🔹 Step 4: Register in Backstage****

1️⃣ Open **Backstage UI** → Navigate to **Create → Register Template**  
2️⃣ Enter the GitHub URL for each template YAML:

https://github.com/my-org/backstage-templates/blob/main/react-template/template.yaml

https://github.com/my-org/backstage-templates/blob/main/angular-template/template.yaml

https://github.com/my-org/backstage-templates/blob/main/node-template/template.yaml

3️⃣ Click **Analyze** → **Import**

### **CI/CD Pipeline with Backstage, GitHub, and Docker**

Setting up a simple CI/CD pipeline using Backstage, GitHub, Docker, and a Node.js "Hello World" app.

## ****Step 1: Prerequisites****

Before starting, ensure you have the following:

* A **GitHub repository** for your Node.js application
* A **Docker Hub account** (krishnamurtyp)
* A **Backstage instance** running for service catalog integration
* Docker and Node.js installed locally

## ****Step 2: Store Docker Credentials Securely in GitHub****

Since your Docker Hub credentials should not be stored in plain text, follow these steps:

1. **Go to Your GitHub Repository**
   * Open [GitHub](https://github.com/) and navigate to your repository.
2. **Add Secrets for Docker Hub Credentials**
   * Click **Settings** → **Secrets and Variables** → **Actions**
   * Click **New repository secret**
   * **Add two secrets**:
     + **DOCKER\_USERNAME** → krishnamurtyp
     + **DOCKER\_PASSWORD** → Google@123

## ****Step 3: Create a Dockerfile for Your Node.js App****

Create a Dockerfile in the root of your repository:

# Use an official Node.js runtime as a base image

FROM node:18-alpine

# Set the working directory

WORKDIR /app

# Copy package.json and install dependencies

COPY package.json package-lock.json ./

RUN npm install

# Copy the application source

COPY . .

# Expose the application port

EXPOSE 3000

# Start the Node.js application

CMD ["node", "server.js"]

## ****Step 4: Create a GitHub Actions Workflow****

Create a **GitHub Actions workflow** in .github/workflows/ci-cd.yml:

name: CI/CD Pipeline

on:

push:

branches: [ main ]

jobs:

build-and-deploy:

runs-on: ubuntu-latest

steps:

- name: Checkout repository

uses: actions/checkout@v3

- name: Set up Docker Buildx

uses: docker/setup-buildx-action@v2

- name: Log in to Docker Hub

uses: docker/login-action@v2

with:

username: ${{ secrets.DOCKER\_USERNAME }}

password: ${{ secrets.DOCKER\_PASSWORD }}

- name: Build and push Docker image

run: |

docker build -t krishnamurtyp/hello-world-node:latest .

docker push krishnamurtyp/hello-world-node:latest

## ****Step 5: Commit and Push Your Workflow****

Now, commit and push everything to GitHub:

git add .

git commit -m "Add CI/CD pipeline with Docker and GitHub Actions"

git push origin main

Go to **GitHub → Actions** to verify if the workflow runs successfully. It should:  
✅ Build the Docker image  
✅ Push it to Docker Hub under krishnamurtyp/hello-world-node:latest

## ****Step 6: Deploy & Run the Container****

Once the Docker image is pushed, you can pull and run it locally:

docker pull krishnamurtyp/hello-world-node:latest

docker run -p 3000:3000 krishnamurtyp/hello-world-node:latest

Your app should now be running at [**http://localhost:3000**](http://localhost:3000) 🚀

## ****Step 7: Integrate with Backstage UI****

To monitor deployments using **Backstage**, follow these steps:

1. **Add a Backstage catalog-info.yaml file** in your repo:

apiVersion: backstage.io/v1alpha1

kind: Component

metadata:

name: hello-world-node

description: A simple Node.js application with CI/CD

annotations:

github.com/project-slug: krishnamurtyp/hello-world-node

backstage.io/techdocs-ref: dir:.

spec:

type: service

owner: default-team

lifecycle: production

1. **Register the component in Backstage**
   * Open Backstage UI
   * Go to **Create Component**
   * Provide the GitHub repo URL
   * Click **Register**

Your service will now appear in Backstage with CI/CD and deployment details.