Multi cloud & multi-tenant Platform

Contents

[Requirements 1](#_Toc192774560)

[Key Features 1](#_Toc192774561)

[Architecture 2](#_Toc192774562)

[Prerequisites 2](#_Toc192774563)

[Dependencies 2](#_Toc192774564)

[User Roles and Responsibilities 3](#_Toc192774565)

[Developer Role 3](#_Toc192774566)

[Admin Role 3](#_Toc192774567)

[Implementation 3](#_Toc192774568)

[Secrets management 3](#_Toc192774569)

[Database setup (DB Admin) 4](#_Toc192774570)

[Deploy the backend to Kubernetes 5](#_Toc192774571)

[Deploy the frontend to Kubernetes 5](#_Toc192774572)

[Access to the platform 5](#_Toc192774573)

[URL: http://multi-cloud-platform.com 5](#_Toc192774574)

Implement a **platform solution** that allows users to deploy on-demand public [Nginx](https://en.wikipedia.org/wiki/Nginx) application available as a docker container at https://hub.docker.com/\_/nginx

## Requirements

* The URL of the application must be publicly available
* All the steps should be automated. It should feel as a one-click deployment with minimal amount of manual actions for the users
* Each trigger would deploy a new instance, independent of the previous one
* The platform should be implemented as code. This means that any resources required for the solution must be defined as code.

Based on the previous requirements, the platform supports the following features:

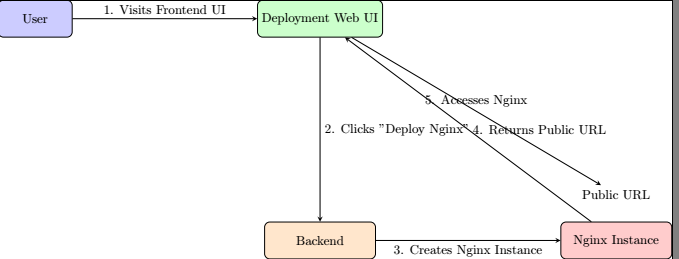
## Key Features

1. **Automated Deployment**: Users can deploy Nginx with a single click.
2. **Public URL**: Each deployment gets a unique, publicly accessible URL.
3. **Independent Instances**: Each deployment is isolated from others.
4. **Infrastructure as Code**: All resources (e.g., Kubernetes deployments, services, ingress) are defined as code.

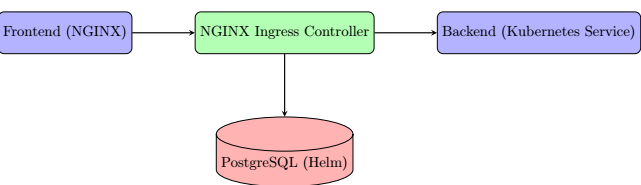
Platform as Code allowing to manage multiple clouds and multiple tenants.

### ****Architecture****

1. **Backend**: Deployed as a Kubernetes service.
2. **Frontend**: Served via NGINX and exposed through the Ingress Controller.
3. **Database**: PostgreSQL deployed using [Helm](https://helm.sh/).
4. **Ingress**: NGINX Ingress Controller for public access.



Processing Workflow



General architecture

### ****Prerequisites****

1. **Kubernetes Cluster**: Set up a Kubernetes cluster (e.g., using Minikube for local development or managed Kubernetes services like GKE, EKS, or AKS).
2. **Ingress Controller**: Install an Ingress Controller (e.g., NGINX Ingress Controller) to expose applications publicly.
3. **PostgreSQL Database**: Set up a PostgreSQL database for storing user and deployment information.
4. **Docker**: Install Docker to containerize the backend and frontend.
5. **Helm**: the package manager that facilitates deployment of Kubernetes resources.

### ****Dependencies****

1. **Flask**: The web framework used to build the backend (python).
2. **Flask-SQLAlchemy**: Provides integration between Flask and SQLAlchemy (database management in python).
3. **Flask-JWT-Extended**: Handles JWT-based authentication and authorization.
4. **kubernetes**: The official Python client for interacting with Kubernetes clusters.
5. **bcrypt**: Used for hashing passwords securely.
6. **gunicorn**: A production-ready WSGI server for serving the Flask app.
7. **psycopg2-binary**: A PostgreSQL adapter for Python (required for connecting to the PostgreSQL database).

## ****User Roles and Responsibilities****

### ****Developer Role****

* **Purpose**: Developers are end-users who deploy and manage their applications.
* **Permissions**:
  + **Deploy new Nginx instances.**
  + View their own deployments.
  + Access logs and metrics for their deployments.
  + **Delete their own deployments.**
* **Restrictions**:
  + Cannot view or modify deployments by other users.
  + Cannot manage platform-wide settings or resources.

### ****Admin Role****

* **Purpose**: Admins manage the platform and oversee all deployments.
* **Permissions**:
  + View all deployments across the platform.
  + Delete any deployment.
  + Manage platform-wide settings (e.g., Kubernetes cluster configuration, ingress rules).
  + Monitor resource usage and performance.
  + Add or remove users and assign roles.
* **Restrictions**:
  + Cannot directly deploy applications (unless they also have developer permissions).

## Implementation

In the following, we will set up the backend, frontend, database, and Kubernetes cluster.

First of all, we will create some secrets using GCP to secure our multi cloud platform.

### Secrets management

Objective: to manage credentials and avoid having unsecure passwords.

#### ****Install Secrets Store CSI Driver and GCP Provider****

# Install Secrets Store CSI driver (Kubernetes uses the CSI driver to provision, attach, and mount storage volumes for pods)

kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/secrets-store-csi-driver/v1.3.4/deploy/rbac-secretproviderclass.yaml

kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/secrets-store-csi-driver/v1.3.4/deploy/csidriver.yaml

kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/secrets-store-csi-driver/v1.3.4/deploy/secrets-store.csi.x-k8s.io\_secretproviderclasses.yaml

kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/secrets-store-csi-driver/v1.3.4/deploy/secrets-store.csi.x-k8s.io\_secretproviderclasspodstatuses.yaml

kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/secrets-store-csi-driver/v1.3.4/deploy/secrets-store-csi-driver.yaml

# Install GCP provider

kubectl apply -f https://raw.githubusercontent.com/GoogleCloudPlatform/secrets-store-csi-driver-provider-gcp/main/deploy/provider-gcp-plugin.yaml

# Check the provisioned pods (optional)

kubectl get pods -n kube-system -l app=csi-secrets-store-provider-gcp

kubectl get pods -n kube-system -l app=csi-secrets-store

# Create the postgresql database URI (including user and password) and the jwt secret

Store Secrets in GCP Secrets Manager (to do using Terraform)

~~Go to the GCP Secrets Manager and create two secrets:~~

~~Name: DATABASE\_URI~~

~~Value: postgresql://user:password@postgres-service:5432/multicloud\_db~~

JWTs (JSON web tokens) are widely used for authentication and authorization in modern web applications. The secret key plays a critical role in ensuring the integrity and security of the tokens.

When the client sends the JWT back to the server (e.g., in an API request), the server needs to verify that the token is valid and hasn't been altered.

(to do using Terraform)

~~Name: JWT\_SECRET\_KEY~~

~~Value: <your-jwt-secret-key>~~

~~Replace the secret key by the output of the command below~~

~~$ openssl rand -base64 32~~

#### Create a SecretProviderClass

A SecretProviderClass custom resource defines which secrets to fetch from GCP Secrets Manager and how to mount them.

$ kubectl apply -f gcp-secret-provider-class.yaml

### Database setup (DB Admin)

to do using Terraform

* Creating the database.
* Creating the necessary tables and schema.
* Seeding the database with initial data.

cd $project\_dir/setup

terraform init

terraform plan

terraform apply

### Deploy the backend to Kubernetes

Suppose that we use GCP to store Docker images

[https://console.cloud.google.com/home/dashboard?project=<your-project-id](https://console.cloud.google.com/home/dashboard?project=%3cyour-project-id)>

gcloud auth login

gcloud auth configure-docker

project\_dir=~/workspace/multi-cloud-multi-tenant

Create the Docker images and push to the remote image registry

$ cd backend

$ docker build -t nginx-backend:v1.x.x .

$ project\_id=$(gcloud config get-value project)

$ docker tag nginx-backend:v1.x.x gcr.io/${project\_id}/nginx-backend:v1.x.x

$ docker push gcr.io/${project\_id}/nginx-backend:v1.x.x

# deploy the configuration of pods

$ kubectl apply -f backend-deployment.yaml

# expose the backend to the network

$ kubectl apply -f backend-service.yaml

### Deploy the frontend to Kubernetes

$ cd frontend

$ docker build -t nginx-frontend:v1.x.x .

$ project\_id=$(gcloud config get-value project)

$ docker tag nginx-frontend:v1.x.x gcr.io/${project\_id}/nginx-frontend:v1.x.x

$ docker push gcr.io/${project\_id}/nginx-frontend:v1.x.x

$ kubectl apply -f frontend-deployment.yaml

$ kubectl apply -f frontend-service.yaml

$ kubectl apply -f frontend-ingress.yaml

## Access to the platform

### URL: <http://multi-cloud-platform.com>