Documentation

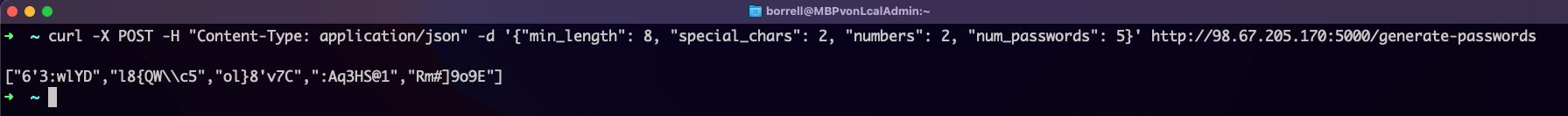
|  |
| --- |
| Challenge: sre-interview \_1\_ |
| Date: 04/30/2023 |
| Developed by: Juan Pablo Borrell |

Let's envision the result first.

The application, as illustrated in the image, operates seamlessly over the internet with its API readily accessible.:

A screenshot of a computer

Description automatically generated



Now, let's backtrack to the inception and dissect the steps required to achieve this functionality.

* Step 1: Docker Image Creation

Starting with the Python code provided, I augmented it to ensure the Flask server listens across all available network interfaces.

Subsequently, I crafted a Docker image housing this modified code.



This image was then deposited into the Dockerhub repository for future deployment iterations.

A screenshot of a computer

Description automatically generated

* Step 2: Infrastructure Setup

Azure served as the backdrop for constructing the infrastructure housing the application. This entailed the establishment of:

* Subscription
* Resource Group
* Kubernetes Cluster (AKS)

Leveraging Terraform, I orchestrated this setup via a series of modules encapsulated within the main.tf file. These modules encompassed:

* Provider: Configuring the requisite provider
* Resource Group (RG): Defining the container for the cluster
* Kubernetes (K8s): Specifying the cluster and its nodes

Furthermore, meticulous variable delineation within each module was adhered to, aligning with best practices. Ensuring uniformity and coherence across team members, I implemented an Azure backend housing the terraform.tfstate file.

* Step 3: ArgoCD Integration

To surpass the initial requirements, I orchestrated the deployment within the ArgoCD cluster. Recognizing its significance in deployment management, this undertaking was executed as standard practice.

A screenshot of a computer

Description automatically generated

ArgoCD, a quintessential automation tool, streamlines the deployment process within Kubernetes environments, increasing development team efficiency and security. It facilitates event monitoring, crucial for maintaining high availability, evidenced by seamless scaling operations.

A screenshot of a computer

Description automatically generated

Augmented with the Helm-charts package manager, ArgoCD empowers swift alterations, rendering it an indispensable addition to the cluster. A glimpse into the app manifest reveals ArgoCD's orchestration of the Helm-chart values ​​file, nestled within my GitHub repository.

A screenshot of a computer

Description automatically generated

* Additional data:

I also use Lens to monitor the Kubernetes cluster. This app allows me to do everything I would do from the console and execute the kubectl commands but from its web interface.

A screenshot of a computer

Description automatically generated

* Repositorio GitHub: <https://github.com/JuanPabloBorrell/challenge.git> --> app and terraform files
* Repositorio Docker: <https://hub.docker.com/repository/docker/laubru1518/password-generator/general>
* App: <http://98.67.204.248:5000/>
* ArgoCD: <https://98.67.205.162/login?return_url=https%3A%2F%2F98.67.205.162%2Fapplications>
* Commands:

• To check the operation of the API:

curl -X POST -H "Content-Type: application/json" -d '{"min\_length": 8, "special\_chars": 2, "numbers": 2, "num\_passwords": 5}' <http://98.67.204.248:5000/generate-passwords>

• Autoscaling – simulate workload:

AUTOSCALING

hey -n 1000 -c 100 -z 300s http://98.67.204.248:5000/

CHECK PODS

kubectl top pods -n pass-gen-blue-env

CHECK HPA

kubectl get hpa -n pass-gen-blue-env

• ArgoCD installation.

Create namespace

kubectl create namespace argocd

Install ArgoCD

kubectl apply -n argocd -f <https://raw.githubusercontent.com/argoproj/argo-cd/v2.10.5/manifests/install.yaml>