**Password Generator Project Deployment Checklist**

**1. Prepare Infrastructure with Terraform**

* **Set up VPC**: Define a Virtual Private Cloud (VPC) in Terraform with necessary subnets, security groups, and route tables.
* **Define EKS Cluster**: Create an Amazon EKS cluster using Terraform, specifying the node group (e.g., t3.small instances).
* **Create IAM Roles**: Define IAM roles and policies for the EKS cluster and node groups.
* **Create Networking Components**: Ensure load balancers, target groups, and necessary networking components are defined in Terraform.
* **Output Configuration Details**: Use Terraform outputs to capture important details like cluster endpoint, security group IDs, and subnet IDs.

**2. Docker Image Preparation**

* **Create Dockerfile**: Define the Dockerfile for your Flask application, ensuring all dependencies are installed.
* **Build Docker Image**: Build the Docker image locally using the Dockerfile.
* **Push Image to DockerHub**: Tag the Docker image and push it to your DockerHub repository.
  + Example Commands:

bash

Copy code

docker build -t maorbrantz/password-generator:latest .

docker push maorbrantz/password-generator:latest

**3. Kubernetes Manifests Preparation**

* **Create Deployment YAML**: Define the deployment file (deployment.yaml) to specify the number of replicas, the Docker image from DockerHub, environment variables, and resource limits.
* **Create Service YAML**: Define the service file (service.yaml) to expose your application. Use LoadBalancer type for external access.
* **ConfigMap (Optional)**: If you have configuration files, define them in a ConfigMap.
* **Secrets (Optional)**: Store sensitive data (like API keys) in Kubernetes Secrets.

**4. Deploying to Kubernetes**

* **Connect to EKS Cluster**: Use kubectl to connect to your EKS cluster.
  + Example Command:

bash

Copy code

aws eks --region us-west-2 update-kubeconfig --name Password-Generator-Cluster

* **Apply Deployment**: Use kubectl apply -f deployment.yaml to deploy the application pods.
* **Apply Service**: Use kubectl apply -f service.yaml to expose the application.
* **Verify Pods and Services**: Use kubectl get pods and kubectl get services to ensure everything is running as expected.

**5. CI/CD with GitHub Actions**

* **Create GitHub Repository**: Store your application code and configuration files.
* **Define GitHub Actions Workflow**: Create a .github/workflows/deploy.yml file for automating the build, push, and deploy process.
  + Example Workflow Steps:
    - Checkout the code.
    - Build Docker image.
    - Push Docker image to DockerHub.
    - Use kubectl commands to apply the latest deployment.
* **Add Secrets to GitHub**: Store sensitive information like DockerHub credentials and AWS credentials in GitHub secrets.

**6. Monitoring and Logging (Optional)**

* **Set up Prometheus and Grafana**: Use Helm charts or manifests to deploy Prometheus and Grafana for monitoring the Kubernetes cluster.
* **Create Dashboards**: Configure Grafana to show dashboards for CPU, memory, and storage usage.

**7. Testing and Validation**

* **Access Application**: Ensure the application is accessible through the load balancer's external IP or DNS.
* **Run Functional Tests**: Verify that the application behaves as expected.
* **Check Logs**: Use kubectl logs to view application logs and debug if necessary.

**8. Scaling and Updates**

* **Define Auto-Scaling (Optional)**: Configure Horizontal Pod Autoscaler (HPA) if you anticipate varying loads.
* **Rolling Updates**: Update the deployment configuration for rolling updates without downtime.