**Deployment Pipeline Documentation**

This documentation explains the steps involved in the deployment pipeline, including Terraform initialization, planning, and application, as well as the use of Ansible playbooks for node and EC2 instance configuration management.

The infrastructure flow is loosely coupled with a few manual tasks (Terraform, Ansible, and secrets integration to GitHub Actions), while the rest is handled by CI/CD pipelines. It follows a GitLab branching strategy where changes are made to the Testing branch and then deployed on Main.

Since AWS free tier is in use, the deployment is minimal. However, this is merely a testbed for future deployment of a more complex and scalable application. With this in mind, some of the files reference locations on localhost, which should be changed to S3 storage. Further improvements include:

* **TLS via Let's Encrypt**: Implementing TLS certificates for secure communication using Let's Encrypt.
* **HTTPS Support**: Enabling HTTPS support for secure web traffic.
* **Slack Notifications**: Setting up Slack notifications to alert about deployment statuses.
* **Terraform State in S3**: Initiating and automating Terraform state in S3.

**Structure of the Project**

*Note: Some files are omitted or temporary.*

.

├── Dockerfile

├── LICENSE

├── README.md

├── ansible\_playbooks

│   ├── ec2\_k3s.yml

│   ├── ec2\_k3s.yml.bak

│   ├── fetch\_kubeconfig.yml

│   ├── fetch\_secrets.yml

│   ├── get\_nodes.yml

│   ├── inventory

│   ├── kubeconfig.txt

│   ├── kubeconfig\_base64.txt

│   └── vars.yml

├── app

│   ├── rn\_jesus.py

│   ├── static

│   │   └── RNJ.png

│   └── templates

│   └── index.html

├── docs

│   └── documentation

├── k8s

│   ├── deploy.yml

│   ├── ingress.yml

│   ├── ingress.yml.j2

│   ├── service.yml

│   └── service.yml.j2

├── requirements.txt

├── terraform

│   ├── main.tf

│   ├── my\_key\_pair.pem

│   ├── outputs.tf

│   ├── terraform.tfstate.d

│   │   └── rn\_jesus

│   │   ├── terraform.tfstate

│   │   └── terraform.tfstate.backup

│   ├── terraform.tfvars

│   ├── terraform.tfvars.gpg

│   ├── terraform\_inventory.yml

│   ├── terraform\_outputs.json

│   ├── terraform\_outputs.json.old

│   └── variables.tf

└── terraform\_inventory.yml

**1. Terraform Initialization, Planning, and Application**

**Step 1: Initialize Terraform**

terraform init

This command initializes the Terraform configuration, setting up the backend and downloading the necessary provider plugins.

**Step 2: Plan Terraform Deployment**

terraform plan

This command creates an execution plan, showing the changes that will be made to the infrastructure.

**Step 3: Apply Terraform Deployment**

terraform apply

This command applies the changes required to reach the desired state of the configuration.

**Step 4: Output Terraform Results**

terraform output -json > terraform\_outputs.json

This command exports the Terraform output variables to a JSON file.

**Step 5: Upload Outputs to S3**

aws s3 cp terraform\_outputs.json s3://rnj-bucket/terraform\_outputs.json

This command uploads the JSON file containing the Terraform outputs to an S3 bucket. This information will then be used in the pipeline.

**2. Ansible Playbooks**

**Step 1: Get Nodes Ansible Playbook** This playbook retrieves information about the nodes in the infrastructure and updates the ingress and service manifests.

**Step 2: EC2 Ansible Playbook** This playbook manages EC2 instances, including provisioning and configuration.

**Step 3: Fetch Kubeconfig and Secrets** This step involves fetching the Kubernetes configuration file and SSH key.

**Step 4: Upload to GitHub Secrets** Upload the contents from the files to GitHub Secrets for KUBECONFIG and KUBE\_CONFIG (the SSH key).

Deployment Pipeline Documentation

This documentation explains the steps involved in the deployment pipeline, including Terraform initialization, planning, and application, as well as the use of Ansible playbooks for node and EC2 instance configuration management.

The infrastructure flow is loosely coupled with a few manual tasks (terraform, ansible and secrets integration to Github actions), the rest is handled by CI/CD pipelines.

It follows a Gitlab branching strategy where changes are made to the Testing branch and then deployed on main.

Since AWS free tier is in use the deployment is minimal, however this is merely a testbed for future deployment of a more complex and scalable application.

With this in mind, some of the files are refferencing location on localhost, these should be changed to an S3 storage. Further improvements include:

\* TLS via Let's Encrypt: Implementing TLS certificates for secure communication using Let's Encrypt.

\* HTTPS Support: Enabling HTTPS support for secure web traffic.

\* Slack Notifications: Setting up Slack notifications to alert about deployment statuses.

\* Terraform state being initiated and automated in S3

Structure of the project:

NB: Some file are ommited or temporary

.

├── Dockerfile

├── LICENSE

├── README.md

├── ansible\_playbooks

│   ├── ec2\_k3s.yml

│   ├── ec2\_k3s.yml.bak

│   ├── fetch\_kubeconfig.yml

│   ├── fetch\_secrets.yml

│   ├── get\_nodes.yml

│   ├── inventory

│   ├── kubeconfig.txt

│   ├── kubeconfig\_base64.txt

│   └── vars.yml

├── app

│   ├── rn\_jesus.py

│   ├── static

│   │   └── RNJ.png

│   └── templates

│   └── index.html

├── docs

│   └── documentation

├── k8s

│   ├── deploy.yml

│   ├── ingress.yml

│   ├── ingress.yml.j2

│   ├── service.yml

│   └── service.yml.j2

├── requirements.txt

├── terraform

│   ├── main.tf

│   ├── my\_key\_pair.pem

│   ├── outputs.tf

│   ├── terraform.tfstate.d

│   │   └── rn\_jesus

│   │   ├── terraform.tfstate

│   │   └── terraform.tfstate.backup

│   ├── terraform.tfvars

│   ├── terraform.tfvars.gpg

│   ├── terraform\_inventory.yml

│   ├── terraform\_outputs.json

│   ├── terraform\_outputs.json.old

│   └── variables.tf

└── terraform\_inventory.yml

1. Terraform Initialization, Planning, and Application

Step 1: Initialize Terraform

terraform init

This command initializes the Terraform configuration, setting up the backend and downloading the necessary provider plugins.

Step 2: Plan Terraform Deployment

terraform plan

This command creates an execution plan, showing the changes that will be made to the infrastructure.

Step 3: Apply Terraform Deployment

terraform apply

This command applies the changes required to reach the desired state of the configuration.

Step 4: Output Terraform Results

terraform output -json > terraform\_outputs.json

This command exports the Terraform output variables to a JSON file.

Step 5: Upload Outputs to S3

aws s3 cp terraform\_outputs.json s3://rnj-bucket/terraform\_outputs.json

This command uploads the JSON file containing the Terraform outputs to an S3 bucket. This information will be then used in the pipeline.

2. Ansible Playbooks

Step 1: Get Nodes Ansible Playbook This playbook retrieves information about the nodes in the infrastructure and updates the ingress and service manifests.

Step 2: EC2 Ansible Playbook This playbook manages EC2 instances, including provisioning and configuration.

Step 3: Fetch Kubeconfig and Secrets This step involves fetching the Kubernetes configuration file and SSH KEY.

Step 4: Upload the contents from the files to GitHub Secrets for KUBECONFIG and KUBE\_CONFIG (This is the SSH key)