Infrastructure as Code Setup and Guide

Walkthrough Video

Repository Setup

Clone the repository

1. clone repo named "deploy"

   `> git clone git@gitlab.com:vector.ai/deploy.git`

   The repository primarily contains Terraform code (approximately 90%) with a little other formats such as yaml, json etc

Install required applications

2. Install helm, ansible, terraform, kubectl, gcloud

   Here are instructions for installing Helm, Ansible, Terraform, kubectl, and gcloud on common operating systems:

   To install Helm:

   On macOS using Homebrew:

   `> brew install helm`

   On Linux:

   `> curl https://raw.githubusercontent.com/helm/helm/ main/scripts/get-helm-3 | bash`

   On Windows using Chocolatey:

   `> choco install kubernetes-helm`

   Verify the Helm installation:

   `> helm version`

   To install Ansible:

   On Ubuntu/Debian:

   `> sudo apt update`

   `> sudo apt install ansible`

   On macOS using Homebrew:

   `> brew install ansible`

   On Windows:

   Use Windows Subsystem for Linux (WSL) and follow the Linux instructions.

   Verify the Ansible installation:

   `> ansible --version`

   To install Terraform:

   On macOS using Homebrew:

   `> brew tap hashicorp/tap`

   `> brew install hashicorp/tap/terraform`

   On Linux:

   `> wget -O- https://apt.releases.hashicorp.com/gpg | gpg --dearmor | sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg`

   `> echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list`

   `> sudo apt update && sudo apt install terraform`

   On Windows using Chocolatey:

   `> choco install terraform`

   Verify the Terraform installation:

   `> terraform version`

   To install kubectl:

   On macOS using Homebrew:

   `> brew install kubectl`

   On Linux:

   `> curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"`

```
> sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

On Windows using Chocolatey:

```
> choco install kubernetes-cli
```

Verify the installation:

```
> kubectl version --client
```

To install gcloud CLI:

On macOS:

```
> brew install --cask google-cloud-sdk
```

Add the command to your $PATH taking guidance from the output instructions

On Linux:

```
> curl https://sdk.cloud.google.com | bash
```

```
> exec -l $SHELL
```

On Windows:

Download the installer from https://cloud.google.com/sdk/docs/install

Run the installer and follow the prompts

Verify the installation:

```
> gcloud version
```

Initialize gcloud:

```
> gcloud init
```

To install Docker:

Follow the official Docker installation instructions for your specific Linux distribution.

Remember to keep these tools updated regularly for the latest features and security patches.
Each tool may have additional configuration steps depending on your specific use case and environment.

Key Repository Components

`gitlab-ci` folder: Contains templates for CI/CD deployments

Microservice template: eg `.template.yml` - For running of standard microservice setups

GitLab runner deployment script: `> ./deploy_runner.sh` Used to deploy the runner within Kubernetes

At the root of the folder:

`> make` file that can be run using make (and others) and will run a plan on the staging env.

Environment Prerequisites

1. Ensure users have a service account with correct permissions
2. Some assumptions are made about the environment setup
3. Certain requirements must be in place before running the infrastructure

Makefile Usage

1. The root of the repository contains a Makefile
2. Compatible with GNU Make and potentially other Make variants
3. Default action is to run a Terraform plan on the staging environment
4. Keywords are used to direct target eg `> make planstag` will ensure we run against staging. This will pull from a preconfigured tfvars env file
5. `> make swprod` will switch and initialise the working env. This can be used as an alternative to `> terraform plan`

Best Practices

1. Always perform a git pull before running any commands

2. Avoid running the default make command without reviewing

Running Terraform Plans

1. Use the command `make` in the root directory to execute the default plan
2. The system will prompt for confirmation before proceeding

GitLab CI/CD Integration

1. GitLab runner is used as the CI/CD agent
2. Jobs run on our own infrastructure, not GitLab's built-in runners to ensure control and customisation of the build and deployment environment

Custom Infrastructure

1. The setup allows for running CI/CD jobs in our own infrastructure

Template Customization

1. Utilises the templates folder to customise CI/CD deployments for different services

Kubernetes Integration

1. GitLab runner is deployed within a Kubernetes cluster
2. Ensures scalability and containerization of CI/CD processes

DNS

1. The same folder is used for DNS config file `dns_vector.tf` and `dns_raft.tf`
2. We have a number of subdomains eg sftp endpoints (using filemage) and a couple of TLDs

Cloud Run + RDBs

1. Most of this infrastructure is duplicated for staging and production but as an example `cloud_run.tf` call config in `sql.tf`. Here the setups are different by the instance size, ranges allowed access.

VPN

1. The same folder is used for VPN config
2. This is the basic setup via a module.

Credentials

1. This is a directory `creds`. We are using shared service accounts.

Secrets

1. Some secrets are contained that are base64 encoded
2. Possible use of vault, google secrets manager etc can be used.

README

1. Please keep this updated and correct any issues.

Also contained

1. Intrusion Detection Service
2. HELM Charts
3. Composer (a separate K8s cluster running this)

4. K8s config
5. Atlantis setup

Remember to review and understand each step thoroughly before implementation.
This guide provides a high-level overview of the infrastructure as code setup process.