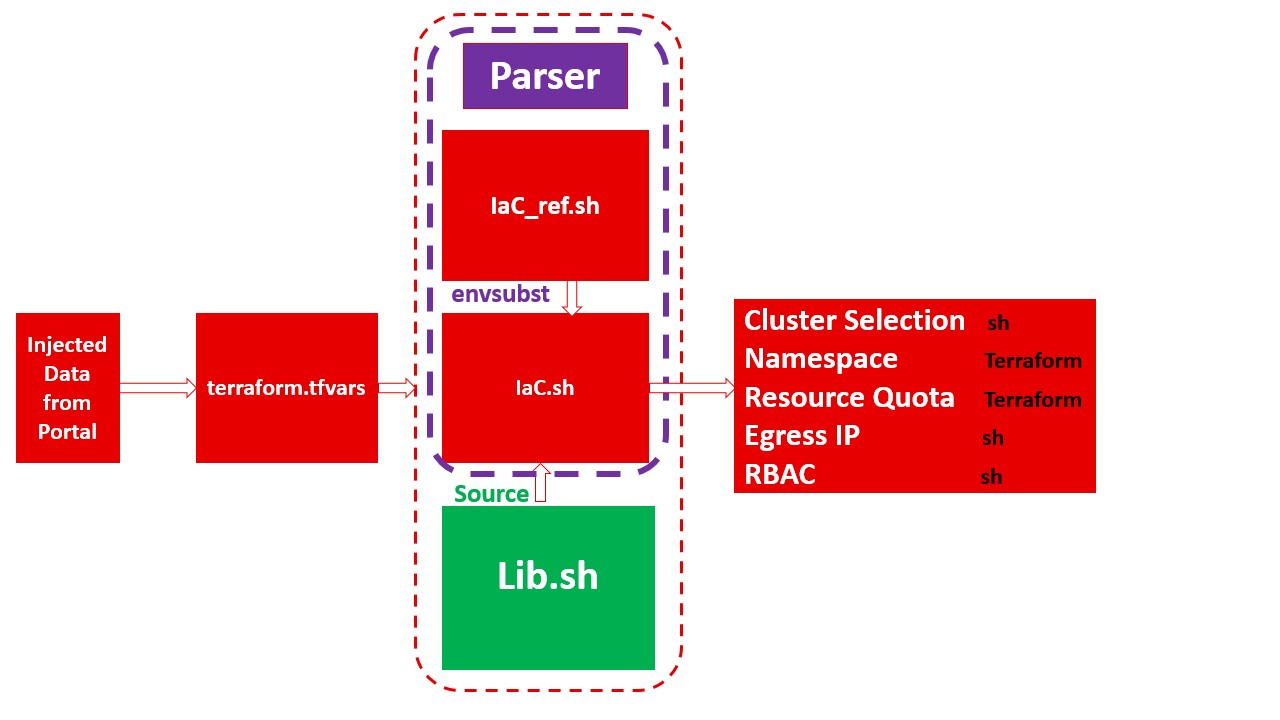
**Vodafone Infrastructure as a code**

Open shift Part – Database configuration – Runners preparations - Keysotre\_Ldap-Integration

Index

* Open Shift Part
  + Diagram
  + Overview
  + Components:
    - GitLab Pipeline
    - Bash files
    - Terraform files
* Database Configuration
* KEYSTORE Configuration

Diagram



The Parser.sh bash script is used to set up infrastructure as code. It first checks to see if the necessary files, such as terraform.tfvars and IaC files, are present and valid. If they are, it then proceeds to assign variables values to EgressIP, project, Site, NT, and GroupDL, which are defined in the terraform.tfvars file. It then uses envsubst to substitute these variable values in the IaC\_ref.sh file and write them to the IaC.sh file. Finally, the IaC.sh file is executed to set up the infrastructure as code. The Lib.sh bash library provided contains three separate functions:

* EgressIPadder
* GroupAdministration
* SiteSelector.

After that, a terraform plan is executed and then terraform apply is executed with the arguments from the KUBECONFIG. Finally, the OpenShift node selector is annotated, and the namespace is labeled argument. The Terraform YAML files Main.tf and project.tf set up a provider configuration to connect to a Kubernetes cluster and create a namespace resource with the name specified in the "var.project" variable.

The project.tf file also sets up a resource quota in the same namespace as Main.tf. The "hard" parameter sets quotas for each resource type this resource quota allows the Kubernetes cluster to limit the resources available to each namespace, ensuring that the resources are utilized effectively and efficiently.

**GitLab Pipeline:**



This Gitlab pipeline file is setting up four stages:

* test
* provision
* Email-failure
* Email-success.

In the test stage, it is running the command terraform --version to validate the Terraform version being used.

In the provision stage, it is running the Parser.sh script from the Terraform directory.

In the email-failure stage, it is sending an email to specified receivers by Vodafone Egypt team with the subject line "Project Pipeline" and the details message.

And in the email-success stage, it is sending an email to specified receivers by Vodafone Egypt team with the subject line "Project Pipeline" and the details message.

**Bash files**

**Parser.sh**



This bash script performs the task of setting up and configuring an infrastructure as code. It first checks if the necessary files, such as terraform.tfvars and IaC files, are present and valid. If they are, it then proceeds to assign variable values to the EgressIP, project, Site, NT, and GroupDL, which are defined in the terraform.tfvars file. It then uses envsubst to substitute these variable values in the IaC\_ref.sh file and write them to the IaC.sh file. Finally, the IaC.sh file is executed to set up and configure the infrastructure as code.

**Main commands:**

The main commands in this script are "export", "Variable\_subst", "envsubst", and #!/bin/bash".

**Export** is a shell command that is used to set environment variables. It takes the format of "export VARIABLE=value".

**Variable**\_**subst** is a function that is used to extract variables from a file and set them as environment variables. It takes a single argument which is the name of the variable to be extracted.

**Envsubst** is a command-line tool that is used to substitute environment variables with their values. It takes a list of variables in the form of ${VARIABLE} and replaces them with the corresponding values.

**#!/bin/bash** is a The #! Syntax is used in scripts to indicate an interpreter for execution under UNIX / Linux operating systems

**The library "Lib.sh"**

****

****



The bash script provided contains three separate functions: EgressIPadder, GroupAdministration, and SiteSelector.

1. The first function, **EgressIPadder**, is used to add an egress IP address to selected nodes in a specified network namespace. This is done by first checking to ensure that the OpenShift CLI tools are installed, and then proceeding to patch the network namespace and hostsubnet objects with the specified IP address.
2. The second function, **GroupAdministration**, is used to add an edit role to the requester’s group in the specified OpenShift cluster. This is accomplished by first checking to see if the requester’s group exists in the cluster, and then by adding the edit role to the requester’s group, as well as to the devops group. If the requester’s group does not exist in the cluster, the system administrator is contacted.
3. The third function, **SiteSelector**, provided is used to select the site for the application and configure the corresponding environment. The script starts by checking the input argument and then sets the corresponding KUBECONFIG depending on the argument. After that, a terraform plan is executed and then terraform apply is executed with arguments from the KUBECONFIG. Finally, the open shift node selector is annotated, and the namespace is then labeled with the third argument for the script. The script is repeated for each environment and is completed by labeling the namespace with the third argument.

This shell script contains several main commands that are used to set up different configurations on an Open shift cluster. The commands include:

1. EgressIPadder: This function adds an egress IP address to the selected nodes.
2. GroupAdministration: This function adds or removes users from a group, or creates a group and assigns permissions to it.
3. oc: This command is used to manage OpenShift clusters.
4. oc patch: This command is used to patch resources in an OpenShift cluster.
5. oc policy: This command is used to manage user and group roles in an OpenShift cluster.
6. oc adm policy: This command is used to manage roles in an OpenShift cluster.
7. export KUBECONFIG: This command sets the environment variable KUBECONFIG to the specified path. This variable is used to point to the Kubernetes configuration file, which contains the cluster’s credentials and settings.
8. terraform plan: This command is used to generate an execution plan for Terraform, which is a tool for managing infrastructure as code.
9. terraform apply: This command is used to apply the Terraform plan that was generated with the terraform plan command.
10. oc annotate: This command is used to add or modify annotations in Kubernetes objects. Annotations are used to store additional information about the object.
11. oc label: This command is used to add or modify labels in Kubernetes objects. Labels are used to group and categorize objects.
12. SiteSelector: This is a function used to set up the different configurations depending on the specified cluster. It sets the KUBECONFIG environment variable, runs the terraform commands, and adds annotations and labels to the namespace.

**IaC\_ref.sh and IaC.sh**

****

**IaC\_ref.sh** is used to select a site, add an egress IP address to the selected nodes, and administer a group. The script begins by sourcing the Lib.sh library. It then uses the SiteSelector function to select a site, a project, and a label. Then, the EgressIPadder function is used to add an egress IP address to the project. Lastly, the GroupAdministration function is used to adminstrate Groups on the specified project, or create it and assign permissions. Finally, the KUBECONFIG variable is unset, and the terraform.tfstate file is removed.

**IaC.sh** is using variable values after envsubst at parser.sh , As with the first script, the KUBECONFIG variable is unset, and the terraform.tfstate file is removed.

The main commands in this shell script are:

1. SiteSelector: This function is used to select a specific site for the script to run on.
2. EgressIPadder: This function adds an egress IP address to the selected nodes.
3. GroupAdministration: This function adds or removes users from a group, or creates a group and assigns permissions to it.
4. unset: This command unsets environment variables used by the KUBECONFIG utility.
5. rm: This command is used to remove files or directories.
6. terraform.tfstate: This file is used to store the state of Terraform resources.

**Terraform YAML files**

****

****

**Main.tf** sets up the provider configuration to connect to a Kubernetes cluster, and then creates a namespace resource with the name specified in the "var.project" variable.

**project.tf** sets up a resource quota in the same namespace as Main.tf. The "hard" parameter sets the quotas for each resource type to the values specified in the "var.quotapods", "var.quotacpurequest", "var.quotamemoryrequest", "var.quotacpulimit", "var.quotamemorylimit" and "var.quotastorage" variables. Using the resource quota, the Kubernetes cluster is able to limit the resources available to each namespace, ensuring that the resources are utilized effectively and efficiently.

The Terraform scripts contains the following commands:

1. provider "kubernetes": This command is used to configure the Kubernetes provider, which provides access to the Kubernetes API. The "config\_path" parameter specifies the path to the configuration file needed to connect to the Kubernetes cluster.
2. resource "kubernetes\_namespace": This command is used to create a Kubernetes namespace. The "metadata" block defines the name of the namespace.
3. resource "kubernetes*resource*quota": This command is used to set hard limits on the resources allocated to a Kubernetes namespace. The "spec" block defines the limits for various resources such as pods, CPU, memory, and storage.

**Terraform.tfvars** is a file used by Terraform to store input variables. The file is used to define values for variables used in Terraform configurations. These values can be used to customize the behavior of Terraform and to define the parameters of a Terraform configuration such as the Open shift cluster, resource quota, EgressIP, and more. The file is typically stored alongside the Terraform configuration files and allows portal to easily inject variables without needing to modify the configuration itself.

**Key-store**

A keystore is a secure storage facility used to store private keys and digital certificates. When used for LDAPS (LDAP over SSL/TLS), the keystore is used to store the public/private key pair needed for authentication and encryption. The keystore is created using the keytool command, which is part of the Java Development Kit (JDK).

To create the keystore for use with LDAPS, the following command must be executed:

keytool -genkey -alias user -keyalg RSA -keystore keystore.jks

This command will prompt the user for the necessary information needed to generate the keystore. Once the keystore has been generated, it must be configured in the LDAP server in order to enable LDAPS.

For application integration with LDAPS, the keystore must also be configured. This is done by using the keytool command with the following options:

keytool -import -alias user -file cert.cer -keystore keystore.jks

The cert.cer file is the certificate obtained from the LDAP server. This command will add the certificate to the keystore and make it available for use with the application.

In summary, a keystore is a secure storage facility used to store private keys and digital certificates. For use with LDAPS, the keytool command must be used to generate the keystore. For application integration with LDAPS, the keytool command must be used to import the certificate obtained from the LDAP server into the keystore.

Installing a keystore in the trust store /etc/pki/ca-trust/source/anchors/ is a relatively straightforward process:

* First, copy the root certificate to the /etc/pki/ca-trust/source/anchors directory.
* Then, run the command “update-ca-trust” to ensure that the keystore is updated with the new certificate.
* Finally, verify that the keystore is in use and that the certificate is trusted by running the command “openssl verify -CAfile <path-to-keystore>.”
* Once these steps have been completed, the keystore is installed and trusted on the system, allowing for secure connections to be established.

**PostgreSQL 12 replication**

PostgreSQL 12 offers powerful replication capabilities that allow for real-time data replication between databases, ensuring that data is always up-to-date and consistent across multiple systems. With PostgreSQL 12, you can set up streaming replication to keep the primary and standby servers in sync, as well as log-based replication and multi-master replication, which allows for multiple servers to all be updated at the same time. PostgreSQL 12 also offers point-in-time recovery, ensuring that you can restore to a specific point in time in the event of a data loss or corruption. Additionally, PostgreSQL 12 supports native replication slots, allowing for the replication process to be paused and resumed at any time. With PostgreSQL 12, you can rest assured that your data is always safe and secure.

**Install the PostgreSQL 12 server:**

* Use your OS package manager to install the PostgreSQL 12 server. Alternatively, you can also download and install the PostgreSQL 12 server from the official website.

**Create a replication user:**

* Connect to the PostgreSQL server as a superuser
* Run the 'CREATE USER' command to create a replication user
* Grant replication privileges to the replication user

**Configure the replication settings:**

* Edit the postgresql.conf file to set the appropriate replication settings
* Restart the PostgreSQL server for the changes to take effect
* Connect to the server using the replication user and run the 'CREATE REPLICATION SLOT' command to create a replication slot

**Start replicating:**

* Run the 'START\_REPLICATION' command to start the replication process
* Monitor the replication process using the 'SHOWREPLICATIONSTATUS' command
* Also select usename,application\_name,client\_addr,backend\_start,state,sync\_state from pg\_stat\_replication.

**To check the connection between master and standby servers**

By following these steps, you should have a PostgreSQL 12 server with replication up and running.