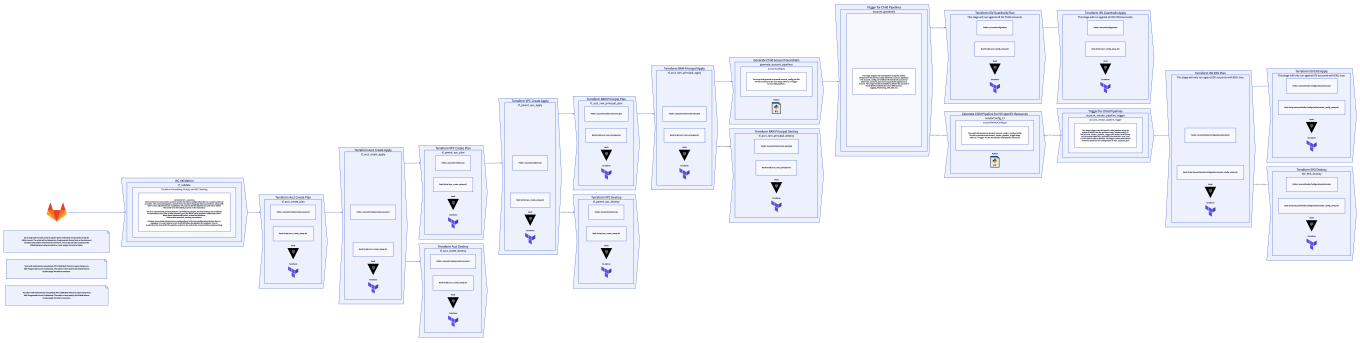
Gitlab/HashiCorp Account Creation & Infrastructure Pipeline

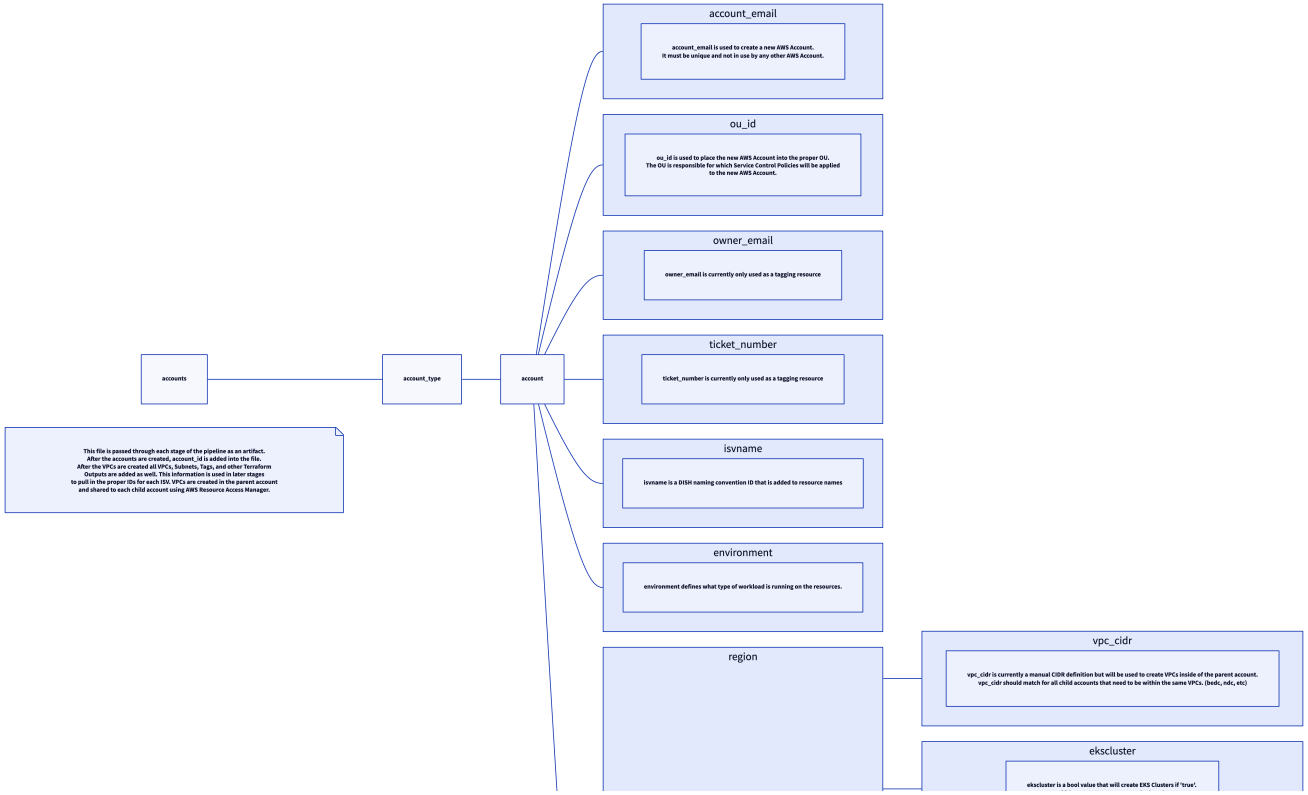
📋 Overview

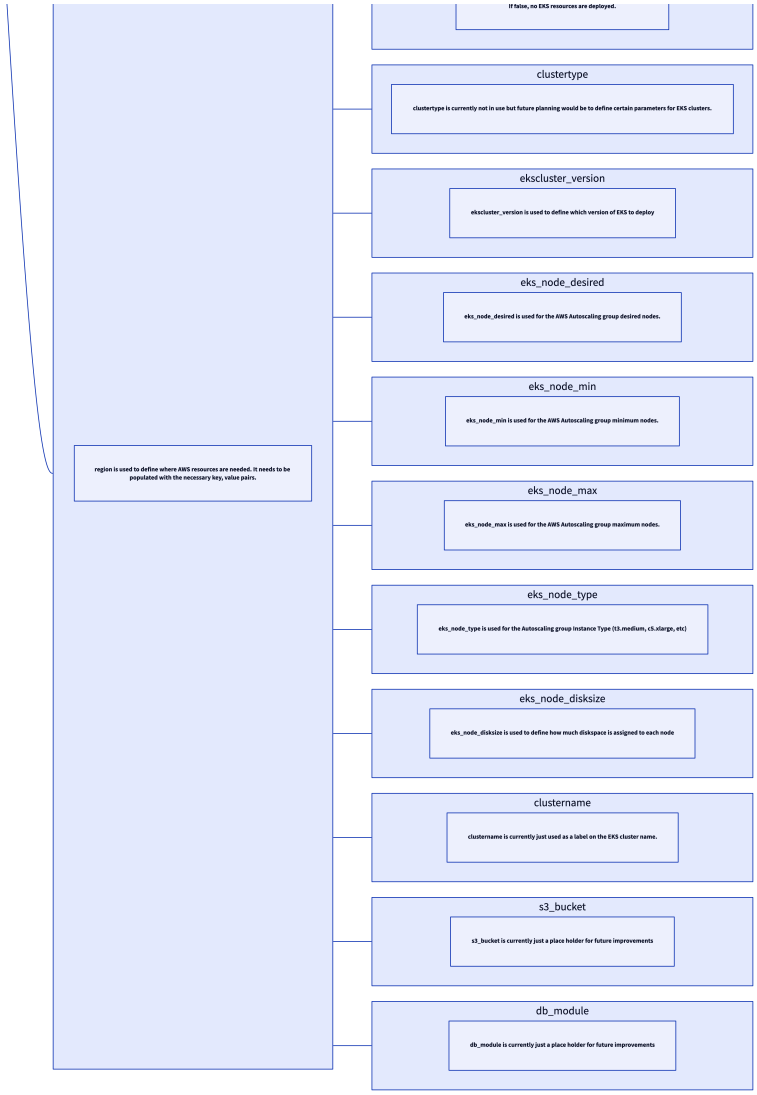
The purpose of the Gitlab pipeline is to create AWS Accounts and Infrastructure utilizing Terraform. Populating a detailed JSON document (aws_accounts.json) with desired parameters will create the defined accounts and infrastructure. The JSON file is passed from stage to stage as a Gitlab Pipeline Artifact and will evolve as resources are created and appended to the document.

CICD Architecture (Infrastructure)



aws_accounts.json





JSON Formatting (aws_account.json)

```
```{
  "accounts": {
      "<AccountType>": {
          "<IsvAccount>": {
              "account_email": "<UniqueEmail>",
              "ou_id": "<OuId>",
              "owner_email": "<IsvContact>",
              "ticket_number": "<TicketNumber>",
              "isvname": "<IsvAbbreviation>",
              "environment": "<EnvType>",
              "<Region>":{
                "vpc_cidr": "<VpcCidr>",
                "secondary_cidr": "<MultusCidr>",
                "ekscluster": <Boolean, true/false>,
                "clustertype": "<TypeOfCluster>",
                "ekscluster_version": <VersionNumber>,
                "eks_node_desired": <Number; Desired Number of
AutoScaling Instances>,
                "eks_node_min": <Number; Minimum Number of AutoScaling
Instances>,
                "eks_node_max": <Number; Maximum Number of AutoScaling
Instances>,
                "eks_node_type": "<InstanceType>",
                "eks_node_disksize": <SizeInGb>,
                "clustername": "<ClusterID/Name>",
                "s3_bucket": <Boolean, true/false>,
                "db_module": <Boolean, true/false>
              }
          }
      }
  }
}```
```

- Not all parameters are currently in use (Future State)
  - clustertype
  - s3_bucket
  - db_module
- Formatting is particularly important because each resource is parsed from a particular key:value pair. If values are not provided properly the automation will fail.

Execution Process

The entire pipeline runs based on the provided aws_accounts.json file. Each value is parsed from the JSON document and used as Terraform inputs. The JSON inputs are used as Terraform Locals which are then passed as variables to the Terraform Modules. The pipeline currently executes in essentially 3 stages.

1. Account Creation (Parent Account)
   a. AWS Account creation
   b. Parent Account VPC creations based on provided `vpc_cidr`
   c. RAM Sharing to relevant ISV Accounts (i.e. Accounts with the same provided `vpc_cidr`)
   d. Repository PATH: accountCreation
2. Account Guardrails (All ISV Child Accounts)
```

    a. AWS Config, CloudTrail, IAM Permissions, etc.
    b. VPC/Subnet Tagging. This is a limitation of AWS RAM Sharing, the script will apply proper tagging inside of each new AWS Child Account.
    c. Repository PATH: accountConfiguration
3. ISV Specific Infrastructure (Specific ISV Child Account)
    a. IF `"ekscluster": true`, a EKS cluster will be created within the ISV account based on the passed parameters.
    b. Repository PATH: accountVendorConfiguration

Background Execution Process Notes

To execute these stages there are some other important details to be aware of:

1. Each Stage that does a Terraform Plan/Apply/Destroy will rely on a Terraform State file. This is done using Gitlab State Management.
2. Stages achieve permissions using HashiCorp Vault so temporary Programatic Access keys are created and removed after 15 minutes. This is done by calling a .sh script at the beginning of each Pipeline Stage. Account creation includes a role that can be assumed by Vault inside of each child account.
3. The first pipeline uses a Python Library (gcip) to create dynamic downstream pipelines. The script will create a new Gitlab-CI file that instructs Gitlab which stages need to run downstream.
4. The process is repeated in the Guardrail stage to create another downstream pipeline. The file then runs stages for ISV specific resources.
    a. If `"ekscluster": false` no stage will be created for that ISV. Essentially only running necessary stages.
5. Most stages execute shell and/or Python scripts. They are included within the PATH for each stage.
6. README files should be included for each folder and contain useful information on what the scripts are executing.

Limitations

- The deployments are currently limited to the AWS Region `"us-west-2"`. Multi-region will require some modifications to the stages being created. Potential solution would be to add region variable to the Terraform Apply command as well as have stages for each deployment region.
- Terraform State is setup for a single pipeline execution. If multiple Beta environments are to be spun up this will need to be modified. Potential solution would be to include the branch name, commit id, or something unique for the deployment so resources do not conflict on deployments.
- Currently these resources are not running on DISH Infrastructure. Once Gitlab, AWS, and HashiCorp Vault are in place it can be moved and tested.
- Currently all stages run a vanilla `image: alpine:latest` that is customized at launch. Custom runners would add value as the DISH Gitlab environment is built out.
- The existing infrastructure is deployed using a local modules folder. This should be split out into a separate repository before moving into production. This would allow modules to be managed separately from the deployments. Highly recommend tagging modules, otherwise module updates may conflict with previously working code.

Code Testing/Validation

The first stage of the pipeline will execute Terraform validation and testing. The validation stage includes built in Terraform Validation as well as Checkov and TFLint. Currently these errors do not cause the pipeline to fail which may not be desired in Production.

📚 Resources

Python Library: gcip - Write your Gitlab CI pipelines in Python

Gitlab State Management: https://docs.gitlab.com/ee/administration/auth/jwt.html