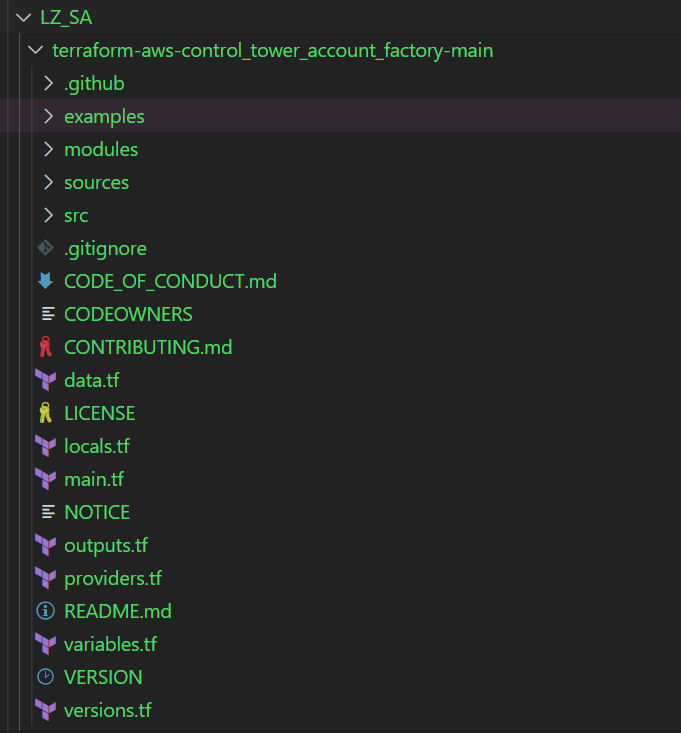
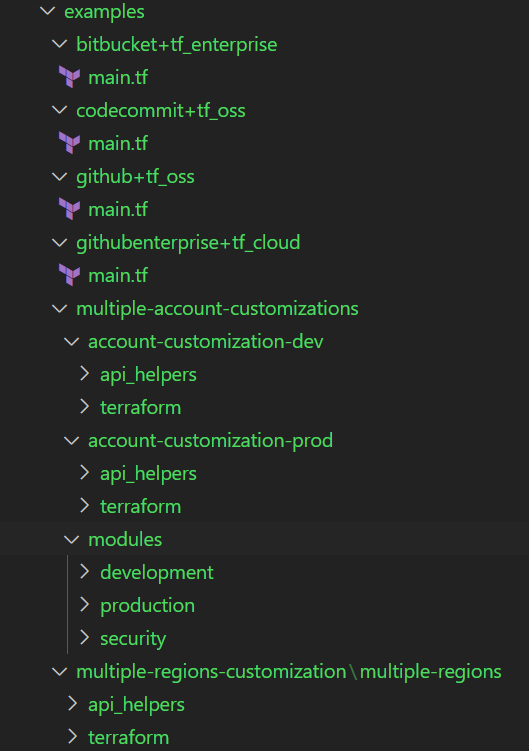
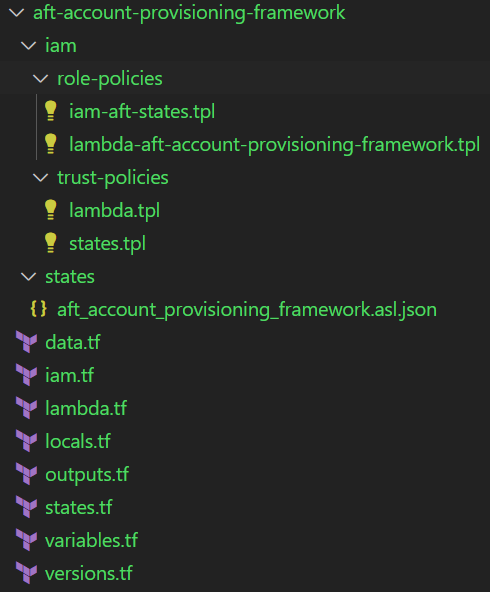
The Terraform language includes a few kinds of blocks for requesting or publishing named values.

* **Resource Block**: A resource block declares a resource of a given type with a given local name. The name is used to refer to this resource from elsewhere in the same Terraform module, but has no significance outside of the scope of a module.
* [Input Variables](https://www.terraform.io/language/values/variables) serve as parameters for a Terraform module, so users can customize behavior without editing the source.
* [Output Values](https://www.terraform.io/language/values/outputs) are like return values for a Terraform module. Output value for consumption by another module or a human interacting via the UI.
* [Local Values](https://www.terraform.io/language/values/locals) are a convenience feature for assigning a short name to an expression. Terraform locals are named values that you can refer to in your configuration. You can use local values to simplify your Terraform configuration and avoid repetition. Local values (locals) can also help you write more readable configuration by using meaningful names rather than hard-coding values. Local values assigning names to expressions, so you can use these multiple times without repetition e.g., service\_name = "forum"
* **Versions**: You can also set a version constraint for each provider defined in the required\_providers block. The version attribute is optional, but we recommend using it to constrain the provider version so that terraform does not install a version of the provider that does not work with your configuration. If you do not specify a provider version, terraform will automatically download the most recent version during initialization.
* **Data Block**: A data block requests that Terraform read from a given data source and export the result under the given local name. The name is used to refer to this resource from elsewhere in the same Terraform module, but has no significance outside of the scope of a module.

**Root File Structure of git:**



* **terraform-aws-control\_tower\_account\_factory**  
  will be storing all the files.
  1. **.github** :  
     it has pull request file  
     
  2. **examples**:  
     it contains some organizational based examples to setup the account for dev and prod with some predefined modules for the same.  
     
  3. **Modules**:  
     - **aft-account-provisioning-framework**  
     

|  |  |
| --- | --- |
| **IAM** | 1. role-policies - iam-aft-states.tpl  - IAM aft policies for states and allow and deny rule with effect.   - lambda-aft-account-provisioning-framework.tpl  - Lambda aft policy for framework resources 2. trust-policies  - lambda and states.tpl  - it has the allow property for lambda and states services to be used via sts:Assume role. |
| **States** | it has json state of the aft framework for aft-account-provisioning-framework. |
| **data.tf** | IAM is using this data methods for resource creation with the help of role-based policy. |
| **Iam.tf** | IAM.tf is performing various tasks like invoke\_aft\_account\_provisioning\_framework, inside this it is using various data sets to create and attach the tags to resources: 1. Validate Request Lambda Permissions  2. Get Account Info Lambda Permissions  3. Create Role Lambda Permissions  4. Tag Account Lambda Permissions  5. Persist Metadata Lambda Permissions  6. states\_execution\_role |
| **Lambda.tf** | It is creating functions for the following: ### VALIDATE REQUEST FUNCTION  ### GET ACCOUNT INFO FUNCTION  ### CREATE ROLE FUNCTION  ### TAG ACCOUNT FUNCTION ### PERSIST METADATA FUNCTION  ### Account Metadata SSM Function  It is using various things like roles, variables etc to create and execute the functions. |
| **Locals.tf** | Locals is taking data from data.tf for lambda execution role. |
| **Outputs.tf** | It will provide the details of the given resources that are being created in account, Now, the output basically contains all the output attributes that you might have set in your TF: state\_machine\_arn, validate\_request\_function\_arn  get\_account\_info\_function\_arn, create\_role\_function\_arn,  tag\_account\_function\_arn, persist\_metadata\_function\_arn |
| **States.tf** | This is using lambda resources as locals and it is creating an aws\_sfn\_state\_machine resource. |
| **Variables.tf** | This contains all the required variables that are going to be used to execute the **aft-account-provisioning-framework** module with type. |
| **Versions.tf** | It is maintaining the versions of terraform binary and hashicorp aws provider as well. (It is required) |

- **aft-account-request-framework**   
This module deploys the components responsible for processing an account request. Functionality includes:

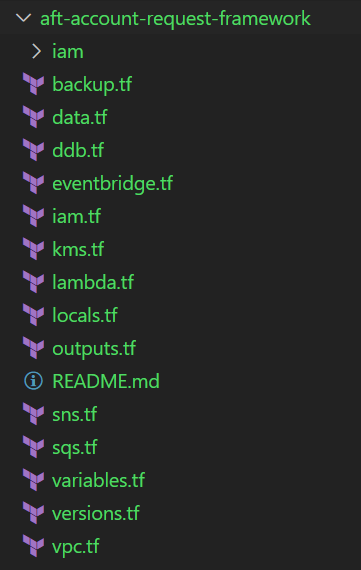
- Creating an audit for account requests/updates in the DynamoDB audit table

- Decision logic on whether a Service Catalog/Control Tower API call is needed

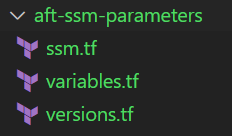
- Queueing Service Catalog requests by leveraging an SQS queue

- Forwarding and storing Control Tower events from the ct-management account into a DynamoDB table

- Triggering Step Functions based on a request payload

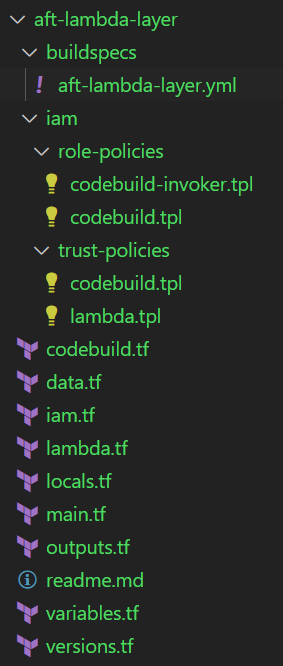


|  |  |
| --- | --- |
| IAM |  |
| Backup.tf |  |
| Data.tf |  |
| Ddb.tf |  |
| Eventbridge.tf |  |
| Iam.tf |  |
| Kms.tf |  |
| Lambda.tf |  |
| Locals.tf |  |
| Outputs.tf |  |
| Sns.tf |  |
| Sqs.tf |  |
| Variables.tf |  |
| Versions.tf |  |
| Vpc.tf |  |

* **aft-ssm-parameters**  
  

-

|  |  |
| --- | --- |
| Ssm.tf | It is using variables and creating resources for SSM like request queue, request table, events etc. |
| Variables.tf | This contains all the required variables that are going to be used to execute the **aft-ssm-parameters** module with type. |
| Versions.tf | It is maintaining the versions of terraform binary and hashicorp aws provider as well. (It is required) |

- **aft-lambda-layer**  
 

|  |  |
| --- | --- |
| **buildspecs** | **- aft-lambda-layer.yml** it contains configuration file, and some commands to execute for creating virtual environment via python 3. |
| **iam** | It has role and trust-based policy for the lambda layer with codebuild |
| **Codebuild.tf** | It is creating a resource of Codebuild project to create lambda layer |
| **Data.tf** |  |
| **Iam.tf** | It is creating the roles for codebuild and attaching the policy to that role. Taking details from variables, locals, and data.tf files. |
| **Lambda.tf** | It is creating a resource of lambda and assigning the role and the function to perform the task, and also it is configuring the VPC as well. It will give an output of the resource build status. |
| **Locals.tf** | Check the top Description |
| **Main.tf** | Creating the resources for the lambda layer, is the main execution file for the **aft-lambda-layer.** |
| **Outputs.tf** | Check the top Description |
| **Variables.tf** | This contains all the required variables that are going to be used to execute the **aft-lambda-layer** module with type. |
| **Versions.tf** | It is maintaining the versions of terraform binary and hashicorp aws provider as well. (It is required) |

**# Python Layer Builder**

Python Layer Builder deploys a codebuild job which creates an AWS Lambda Layer version from a github repository.

This project deploys an IAM role, S3 bucket, Codebuild project, and Cloudwatch Event which triggers the codebuild project.

The codebuild project runs `pip install` using the specified version of python and installs the required packages to a virtual environment,

It then zips the python/lib/pythonx.x folder, and uploads it to S3.

Terraform then waits for approximately 3 minutes and attempts to create a lambda layer from this S3 bucket and key.

If no object is found at this location, the Terraform apply will fail. If this is an update, and something goes wrong with the buid process, the terraform apply will not record any errors. You are encouraged to impement error-handling notifications and integrate them with the python-layer-builder installation in your environment.

**# Requirements File**

The build process expects a python package list at the location: `./layer/requirements.txt`

**# Variables**

layer\_name - the name of the lambda layer

aws\_region - the region to deploy the layer in

source\_url - the url of the github repository which contains the ./layer/ folder, custom packages, and requirements.txt

source\_branch - the branch to clone from the source repository.

source\_type - Currently, only GITHUB has been tested.

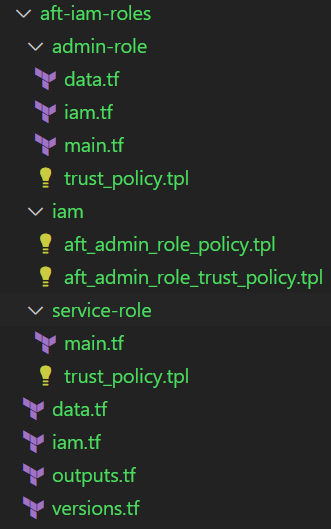
lambda\_layer\_python\_version - Major python version. Defaults to 3.9

github\_token - Set $TF\_VAR\_github\_token to securely configure this variable with a personal access token to github.

**# Outputs**

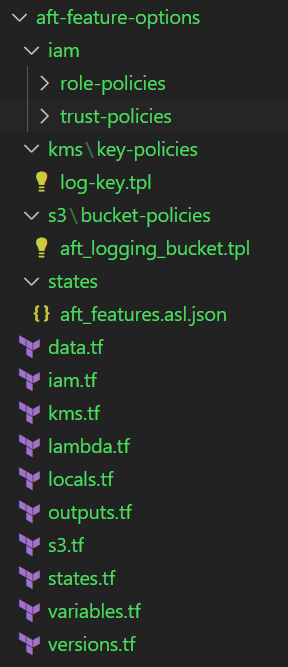
layer\_version\_arn - the ARN of the lambda layer version

- **aft-iam-roles**



|  |  |
| --- | --- |
| Admin-role | 1. Data.tf - aws partition will be readed and exported from here 2. Main.tf - it has terraform provider. 3. Iam.tf - Iam have variables and it is creating a role with name “AWSAFTExecution”  - and attaching the administrator access to role in another resource. - and giving output of the arn. 4. Trust-policy - it has sts:AssumeRole policy. |
| iam | It has trust policy for Iam roles. |
| Service-role | 1. Main.tf - it has terraform provider - it has variable declared - it is creating a role and assigning administrator access to that role 2. Trust-policy - sts:AssumeRole role policy |
| Data.tf | Check the previous discussion. |
| Iam.tf | It is creating roles by using the iam folder policy and iam admin policy and it is also using some modules as well. |
| Outputs.tf | Check the previous discussion. |
| Versions.tf | It is maintaining the versions of terraform binary and hashicorp aws provider as well. (It is required)  it is using configuration\_aliases alias, for using the same provider with different configurations for different resources; version, which we no longer recommend (use provider requirements instead) Unlike many other objects in the Terraform language, a provider block may be omitted if its contents would otherwise be empty. Terraform assumes an empty default configuration for any provider that is not explicitly configured. |

- **aft-feature-options**



|  |  |
| --- | --- |
| iam | Role-policies   * aft\_features\_states.tpl: it has a policy to perform action for sns and lambda invoke. * aft\_enroll\_support.tpl: it has a policy to perform action for ssm, kms, sts, and sns. * aft\_enable\_cloudtrail.tpl: it has a policy to perform action for ssm, kms, sts, and sns. * aft\_delete\_default\_vpc\_lambda.tpl : it has a policy to perform action for EC2, ssm, kms, sts, and sns.   Trust-policies- it has sts assume role policy for lambda and states. |
| KMS | It has policy to accept the services for delivery-logs, vpc-flow-logs, cloudtrail logs + enable IAM user permission for root. |
| S3 |  |
| states |  |
| Data.tf |  |
| Iam.tf |  |
| Kms.tf | KMS key for encrypt/decrypt log files  It is using the policy-> /kms/key-policies/log-key.tpl |
| Lambda.tf |  |
| Locals.tf |  |
| outputs.tf | Check Top discussion |
| S3.tf |  |
| States.tf | It is accessing -> /states/aft\_features.asl.json, roles as well,Lambda.tf, and variable.tf  aft\_features.asl.json - it is deleting the default VPC, and enrolling into the enterprise support by enabling the cloudtrail for success and failure notification. |
| Variables.tf | This contains all the required variables that are going to be used to execute the **aft-lambda-layer** module with type. |
| Versions.tf | It is maintaining the versions of terraform binary and hashicorp aws provider as well. (It is required) |