Git Repo Topology - Data Platform Repos

Document Owner(s)	@Athar Shah (Deactivated) @Jay Jiang
Work Package	WP4
Status	REVIEWED
List of Approvers	☑ @Sampath Jagannathan / @Vishnu Devarajan (Unlicensed) ☑ @Neil Belford ☑ @Former user (Deleted) / @Brandon Lay (Unlicensed) / @Davood Shaiek ☑ @Wee Lih Lee / @Nikilesh Chivukula / @Lawrence Law

Overview

This data platform repo design seeks to strike a balance between 3 competing principles.

Isolation

Environments and BUs should be isolated from each other to prevent collisions. We want to ensure that a change in a development environment does not inadvertently force a refresh of the production state (same applies for changes in other BUs).

Reusability

Also known as DRY (Don't Repeat Yourself); platform code is often used to deploy for multiple instances of the same resource with minor variations. We will use terraform modules to abstract the common patterns across BUs to maximise code re-usability. This principle often conflicts with `isolation` principle, as making code re-usable also create dependencies between `modules` and callers of `modules`; i.e. if 2 BUs share a common module, and BU-1 requires additional functionality added to a shared `module`, changes to the shared `module` must **either** be regression tested to ensure BU-2 will not be impacted **or** the `module` needs to have semantic version tags to adhere to `isolation` principle.

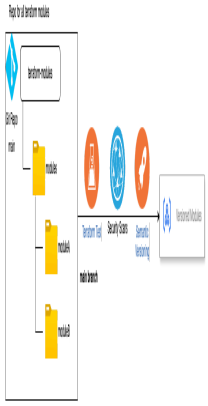
Complexity

The conflict between `isolation` and `reusability` often result in added `complexity`. In the previous example, having semantic versions enabled on the `modules` often mean that you need to break the modules out into their own repos. Considering the size of the operational team, we want to minimise the number of repos for Jemena (as typically more repos means more complexity); however, this means that one repo will be able to have multiple terraform statefiles (corresponding to subdirectories in the repo). The one-to-many relationship between repo and states can be seen as complexity, however, this is the type of complexity that the majority of CI/CD patterns are designed to handle.

Topology Options

Infrastructure Git Topology Options

Separate Terraform Modules Repo

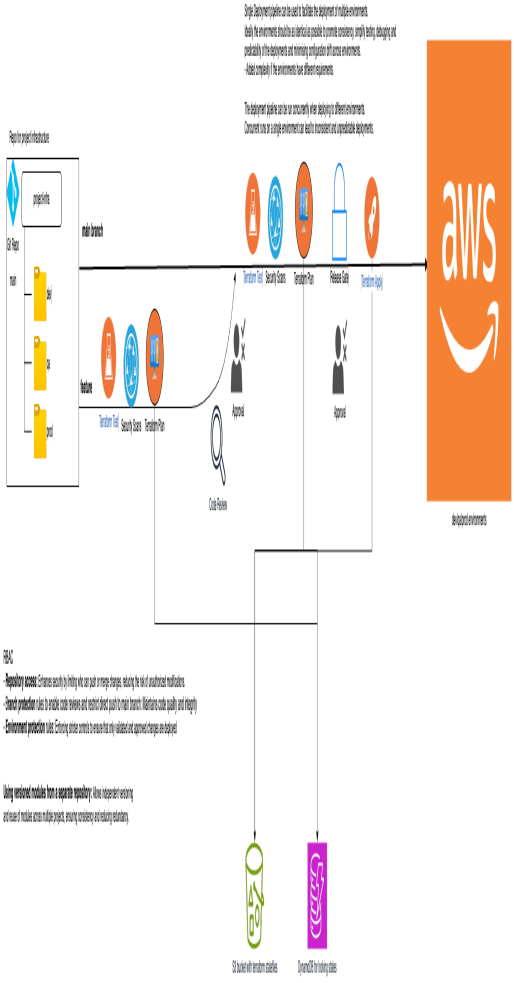


Allows independent versioning, and reuse of modules across multiple projects, ensuring consistency and reducing redundancy.

Modules can be pinned in the infrastructure code to (eg. branch, commit)

Approach 1: Separate Folder for each environment - Single Deployment Pipeline

Trunk based approach



RBAC

- **Repository access:** Enhances security by limiting who can push or merge changes, reducing the risk of unauthorized modifications.
- **Branch protection** rules to enable code reviews and restrict direct push to main branch. Maintains code quality and integrity
- **Environment protection** rules: Enforcing stricter controls to ensure that only validated and approved changes are deployed

Using versioned modules from a separate repository: Allow independent versioning and reuse of modules across multiple projects, ensuring consistency and reducing redundancy.

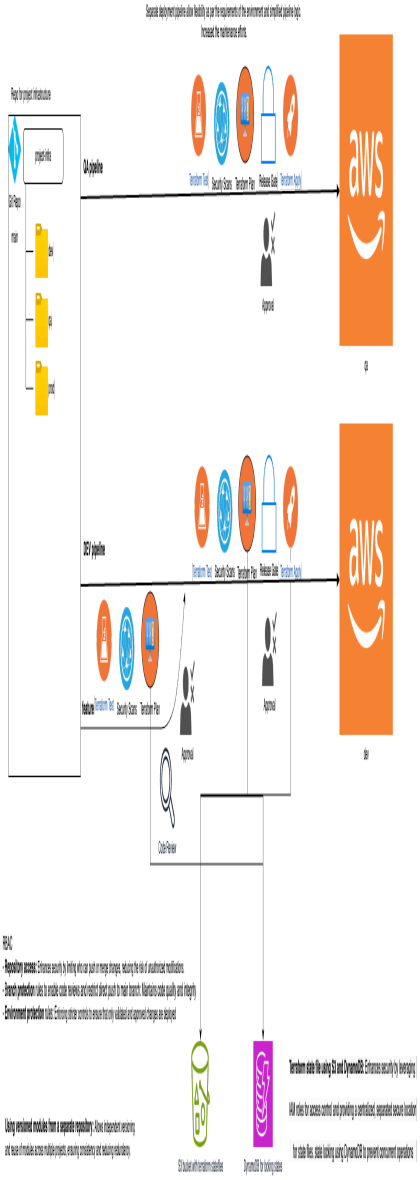
Terraform state file using S3 and DynamoDB: Enhances security by leveraging

IAM roles for access control and providing a centralized, separated secure location

for state files. state locking using DynamoDB to prevent concurrent operations

Approach 2: Separate Folder for each environment - Separate Deployment Pipeline

Trunk based approach



RBAC

- **Repository access:** Enhances security by limiting who can push or merge changes, reducing the risk of unauthorized modifications.
- **Branch protection** rules to enable code reviews and restrict direct push to main branch. Maintains code quality and integrity
- **Environment protection** rules: Enforcing stricter controls to ensure that only validated and approved changes are deployed

Using versioned modules from a separate repository: Allow independent versioning and reuse of modules across multiple projects, ensuring consistency and reducing redundancy.

Terraform state file using S3 and DynamoDB: Enhances security by leveraging

IAM roles for access control and providing a centralized, separated secure location

for state files. state locking using DynamoDB to prevent concurrent operations

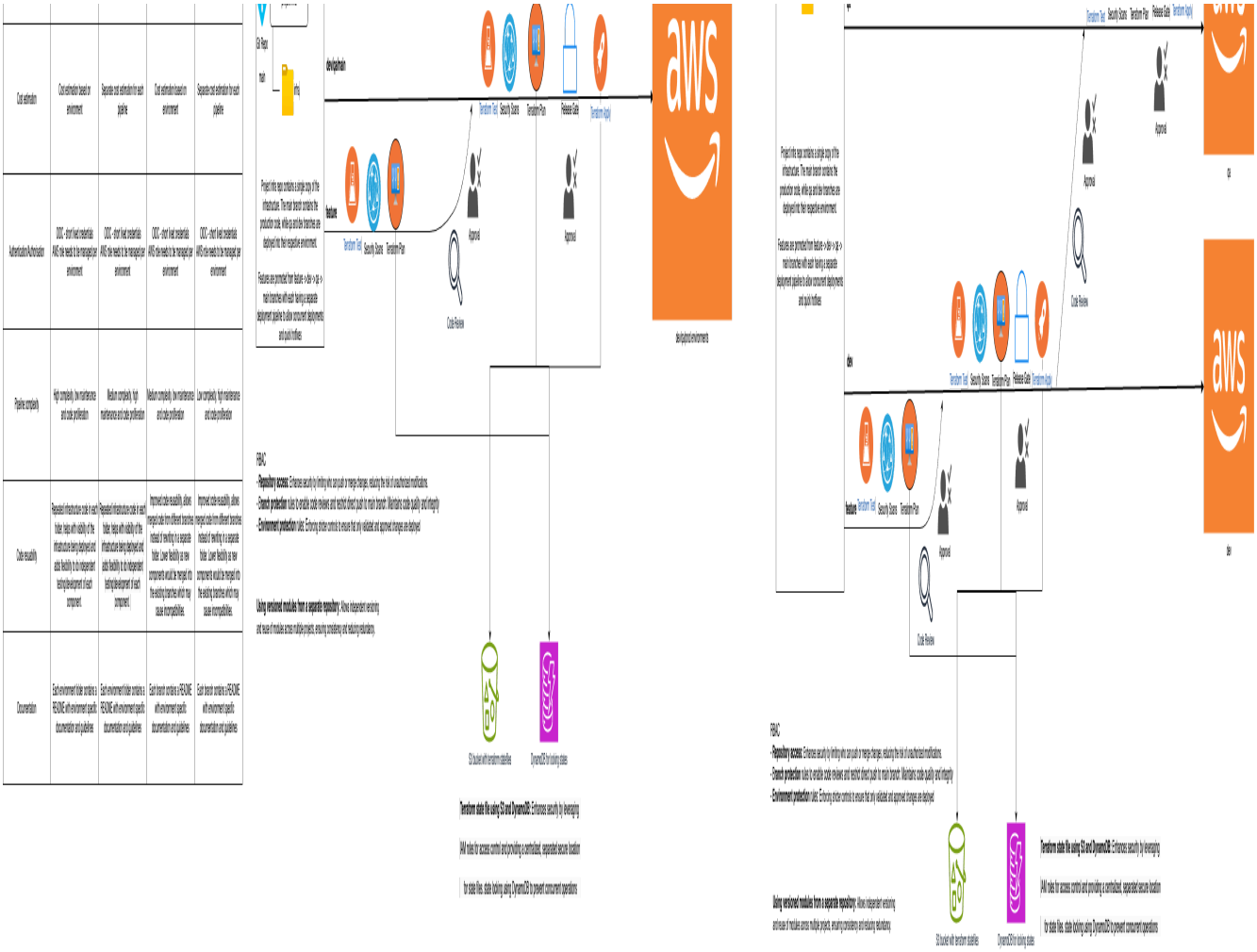
Comparison				
	1- Environments separated by folder - single deployment pipeline	2- Environments separated by folder - multiple deployment pipelines	3- Environments separated by branches - single deployment pipeline	4- Environments separated by branches - multiple pipelines
RBAC	RBAC implemented on multiple levels: - repository - branch - environment	RBAC implemented on multiple levels: - repository - branch - environment	RBAC implemented on multiple levels: - repository - branch - environment	RBAC implemented on multiple levels: - repository - branch - environment
Module versioning	Supports version pinning when using separate repo for modules using tag/version, branch, commit	Supports version pinning when using separate repo for modules using tag/version, branch, commit	Supports version pinning when using separate repo for modules using tag/version, branch, commit	Supports version pinning when using separate repo for modules using tag/version, branch, commit
Policy as code	Each environment folder consists of policies based on the environment requirements Policies executed as a part of the pipeline	Each environment folder consists of policies based on the environment requirements Policies executed as a part of the pipeline	Each branch consists of policies based on the environment requirements Policies executed as a part of the pipeline	Each branch consists of policies based on the environment requirements Policies executed as a part of the pipeline
Security Logging	Audit logs streamed to Splunk - separated by environment	Audit logs streamed to Splunk	Audit logs streamed to Splunk - separated by environment	Audit logs streamed to Splunk

Approach 3: Long Running Branches for each environment - Single Deployment Pipeline

GitFlow: Long running branches

Approach 4: Long Running Branches for each environment - Separate Deployment Pipeline

GitFlow: Long running branches



What are we actually deploying (and how)?

We will be deploying both AWS resources (using AWS terraform provider) and Databricks resources (using Databricks terraform provider). If you are not familiar with terraform, please go through this terraform tutorial AWS | Terraform | HashiCorp Developer before reading the rest of this page, as very little will make sense if you don't know have foundational knowledge of terraform.

AWS Components

Before deploying any databricks resources, we need to deploy the underlying AWS resources. These will be done in separate repos based on which AWS account the resources live in. For all AWS components (i.e. whenever we use AWS terraform provider), we will conform to Jemena's standards for repo and deployment structure as well as naming.

Repo	Resources	Link

`core-network-databricks-vpc-components`	Private Link ENI, Security Groups	Jemena / Projects / Future Networks Datahub / Databricks / core-network-databricks-vpc-components · GitLab
`app-datahub-nonprod-databricks-aws-infra`	S3, Instance Profiles, IAM Roles	Jemena / Projects / Future Networks Datahub / Databricks / app-datahub-nonprod-databricks-aws-infra · GitLab
`app-datahub-prod-databricks-aws-infra`	S3, Instance Profiles, IAM Roles	Jemena / Projects / Future Networks Datahub / Databricks / app-datahub-prod-databricks-aws-infra · GitLab

Databricks Components

After AWS components are deployed, they either need to be registered in databricks or otherwise used by databricks to create subsequent resources, in order to pass AWS ARNs etc. onto databricks resources, we will need to use GitLab's terraform remote state as a data source and provision the corresponding permission for CI runners.

Repo	Resource Level	Link
`databricks-unity-catalog`	Account Only + Workspace-agnostic resources	Jemena / Projects / Future Networks Datahub / Databricks / databricks-unity-catalog · GitLab

`databricks-workspaces`	Account + Workspace	Jemena / Projects / Future Networks Datahub / Databricks / databricks-workspaces · GitLab

Within these 2 repos, the first layer of directories distinguish `modules` and `stacks`.

Modules

Modules sub-directory contain the definition for how to deploy a particular combination of resources. e.g. `RBAC` module would define how the RBAC hierarchy is defined for any arbitrary Business Unit. Nothing is created by modules.

Stacks

Stacks call `modules`, they pass in configuration to modules and create the resources based on the patterns defined within modules. e.g. `RBAC` stack would call `RBAC` module (via source=) and pass in configuration such as BU's name.

Directories within Stacks

Every directory within stacks directory has its own terraform statefile, this is how we ensure `isolation` across BUs. Environment injection happens dynamically via `x.auto.tfvars` within those sub directories to distinguish `prod` and `noprod` deployments; the injection of actual values into `x.auto.tfvars` will be set based on the CI/CD design (typically split by branch names).

```
├── databricks-unity-catalog
    │   ├── modules
    │   │   ├── rbac
    │   │   ├── schemas
    │   │   └── service-principals
    │   └── stacks
    │       ├── digital-bu
    │       │   ├── catalogs.tf
    │       │   ├── rbac.tf
    │       │   └── schemas
    │       ├── elec-network-bu
    │       │   ├── catalogs.tf
    │       │   ├── rbac.tf
    │       │   └── schemas
    │       └── org (no CI, manual deploy)
    │           └── metastore.tf
    └── databricks-workspaces
```

```
        ├── modules
        │   ├── cluster-policy
        │   ├── long-lived-clusters
        │   ├── managed-workspace
        │   └── sql-warehouses
        └── stacks
            ├── digital-lab-workspace-infra
            ├── digital-field-workspace-infra
            ├── digital-lab-workspace-resources
            ├── digital-field-workspace-resources
            ├── elec-network-lab-workspace-infra
            ├── elec-network-field-workspace-infra
            ├── elec-network-lab-workspace-resources
            └── elec-network-field-workspace-resources
```

Rules & Workflow

rules

- main direct push to `main` denied, can only be modified via `MR`
- `MR` s require minimum 1 approval and pipeline success to merge
  - subsequent commits revoke previous approvals