Managing AWS Infrastructure with Terraform v1

Agenda

Theory

- Iterative / incremental Infrastructure-as-Code
- Basics
- Templating
- Workspaces

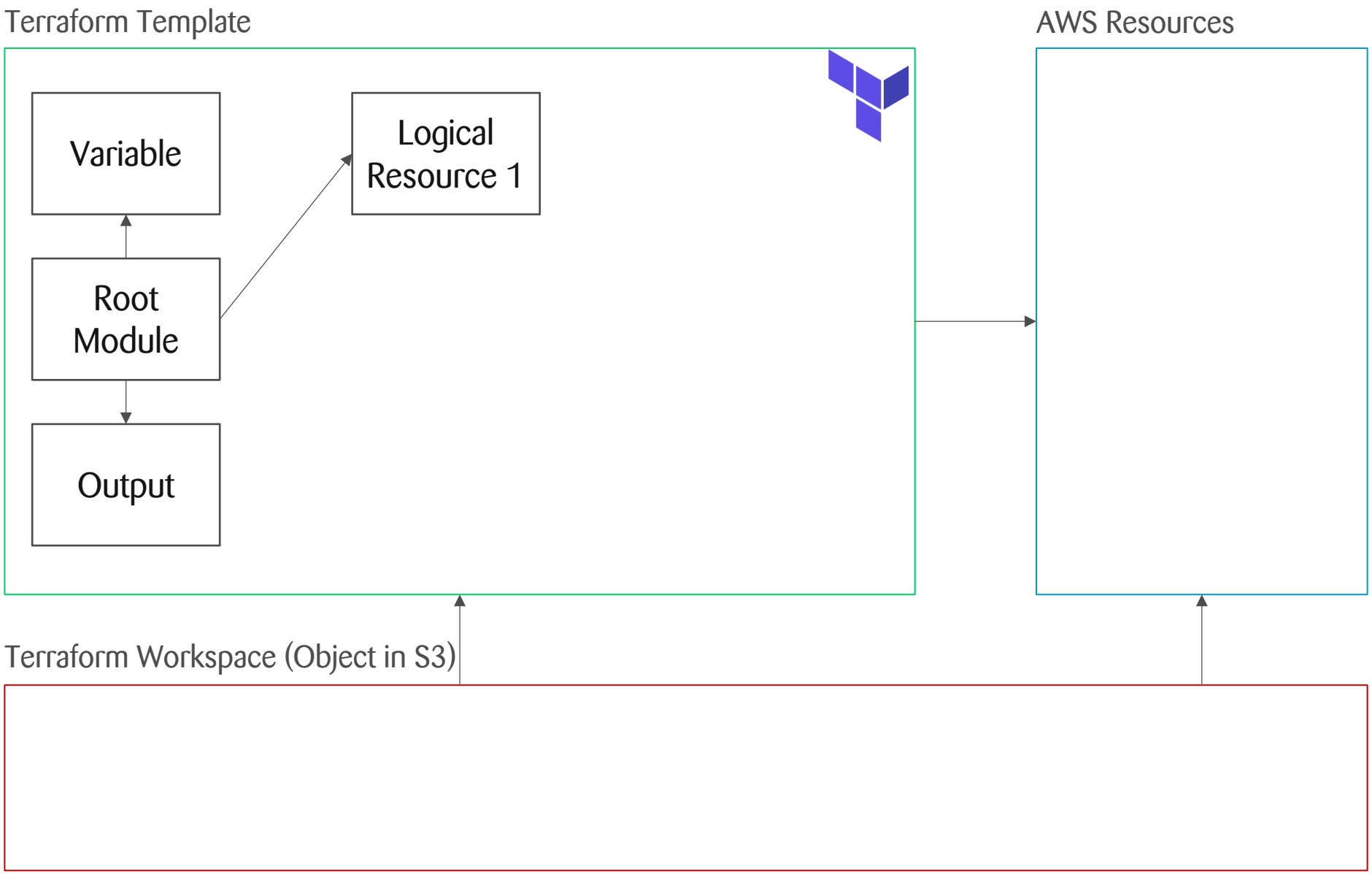
Practice

- Samples

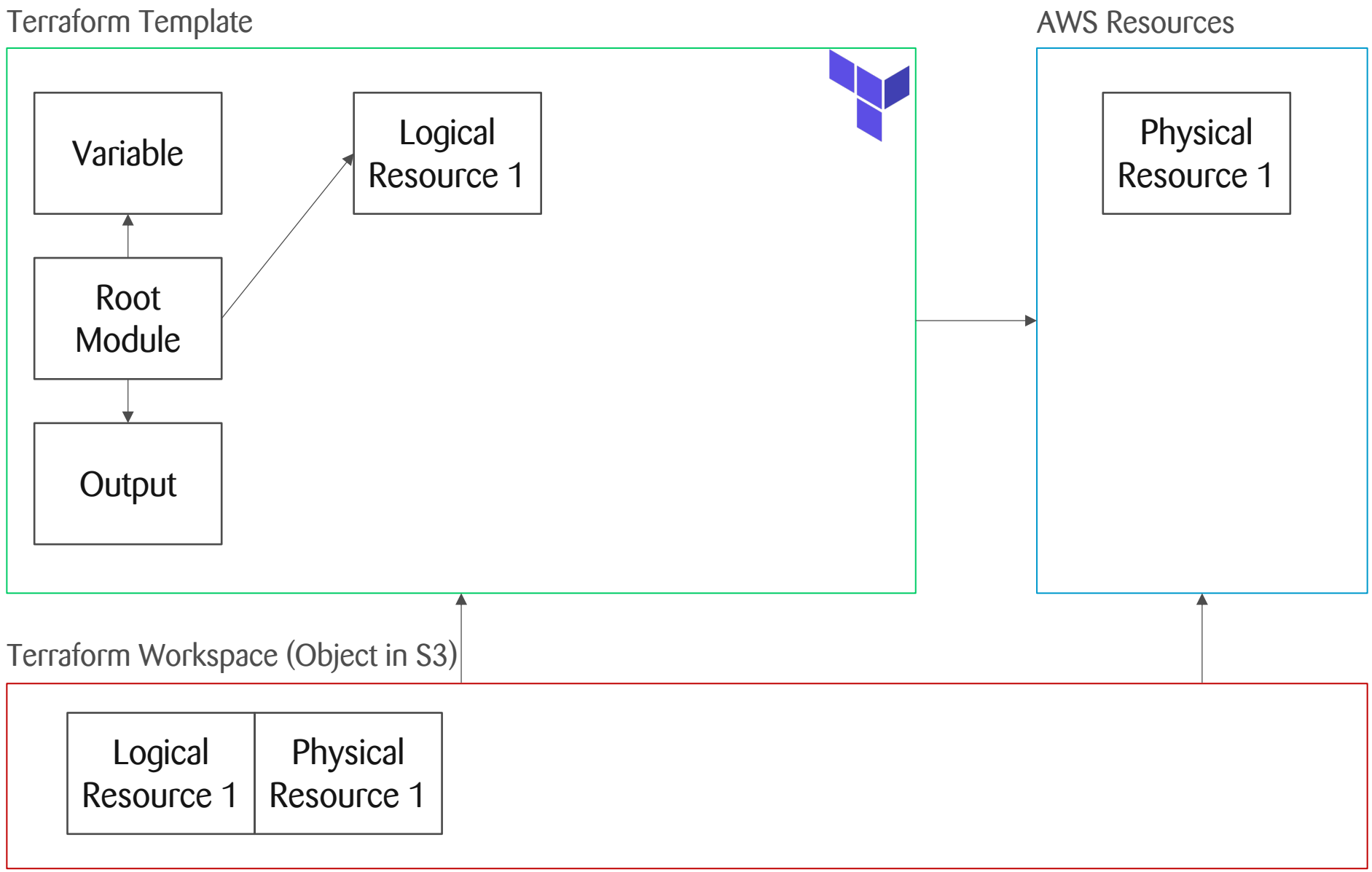
Q & A

Iterative / incremental Infrastructure-as-Code

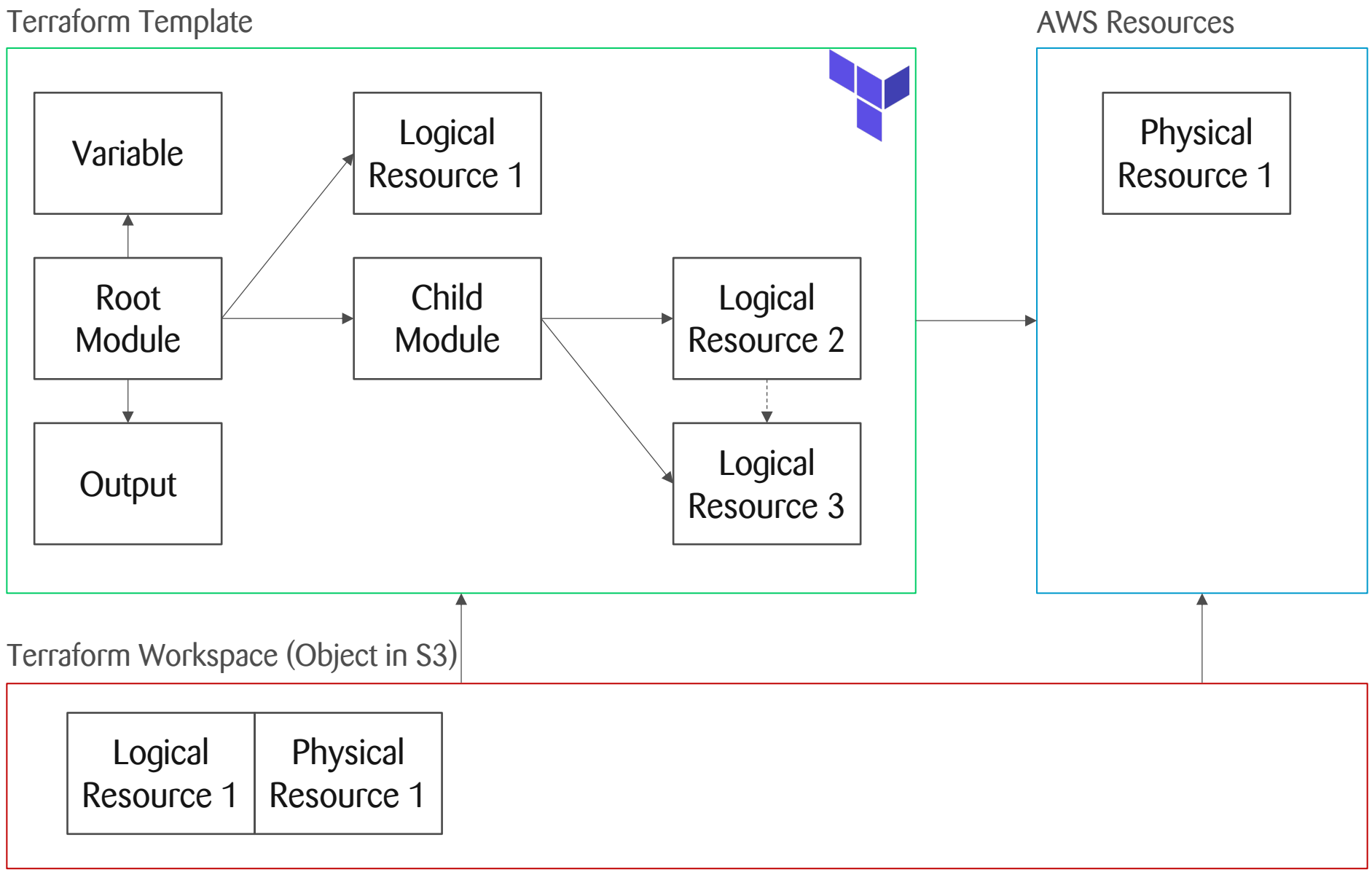
t=0: template version 1 created



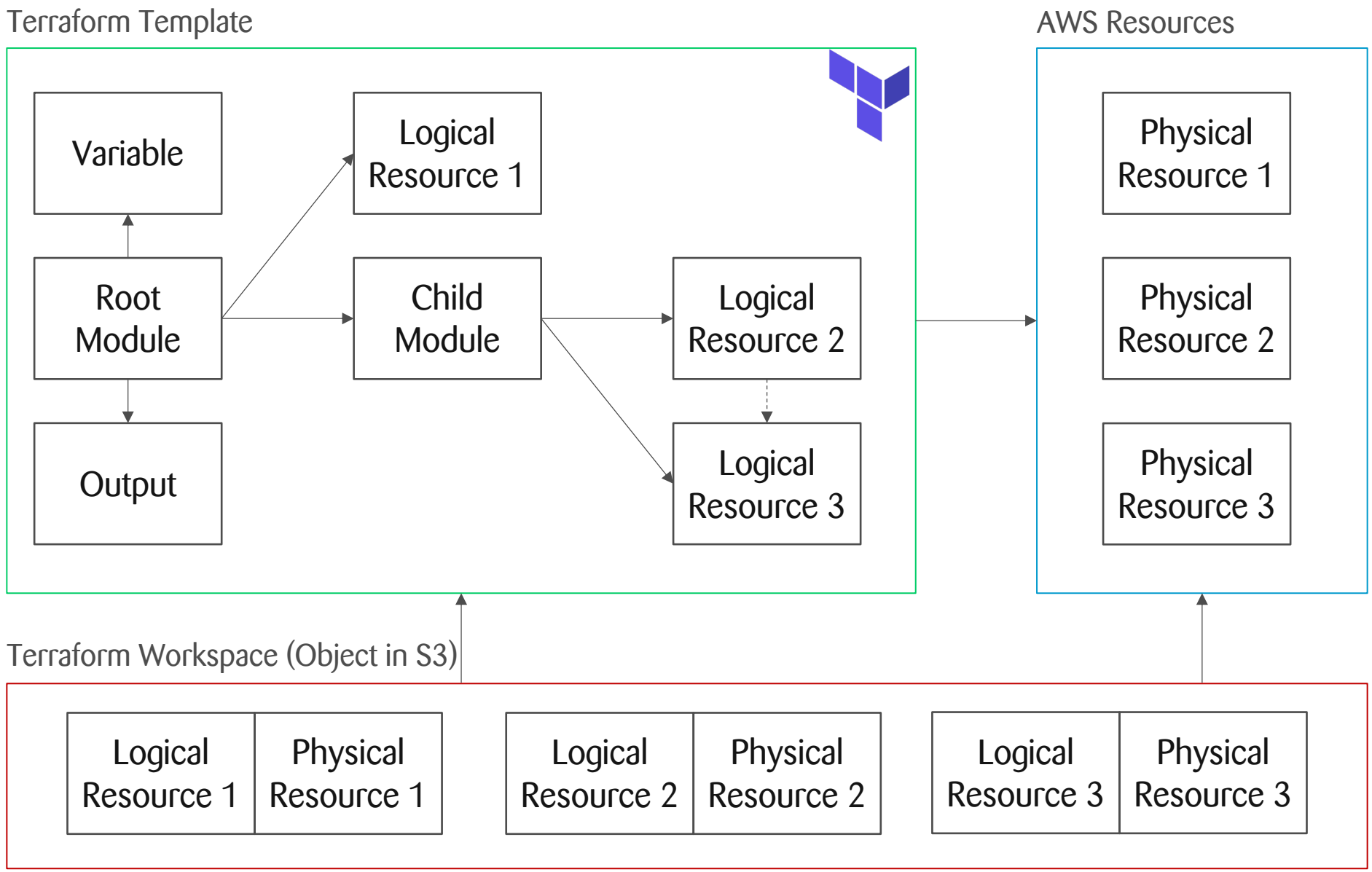
t=1: template version 1 applied (physical resources created)



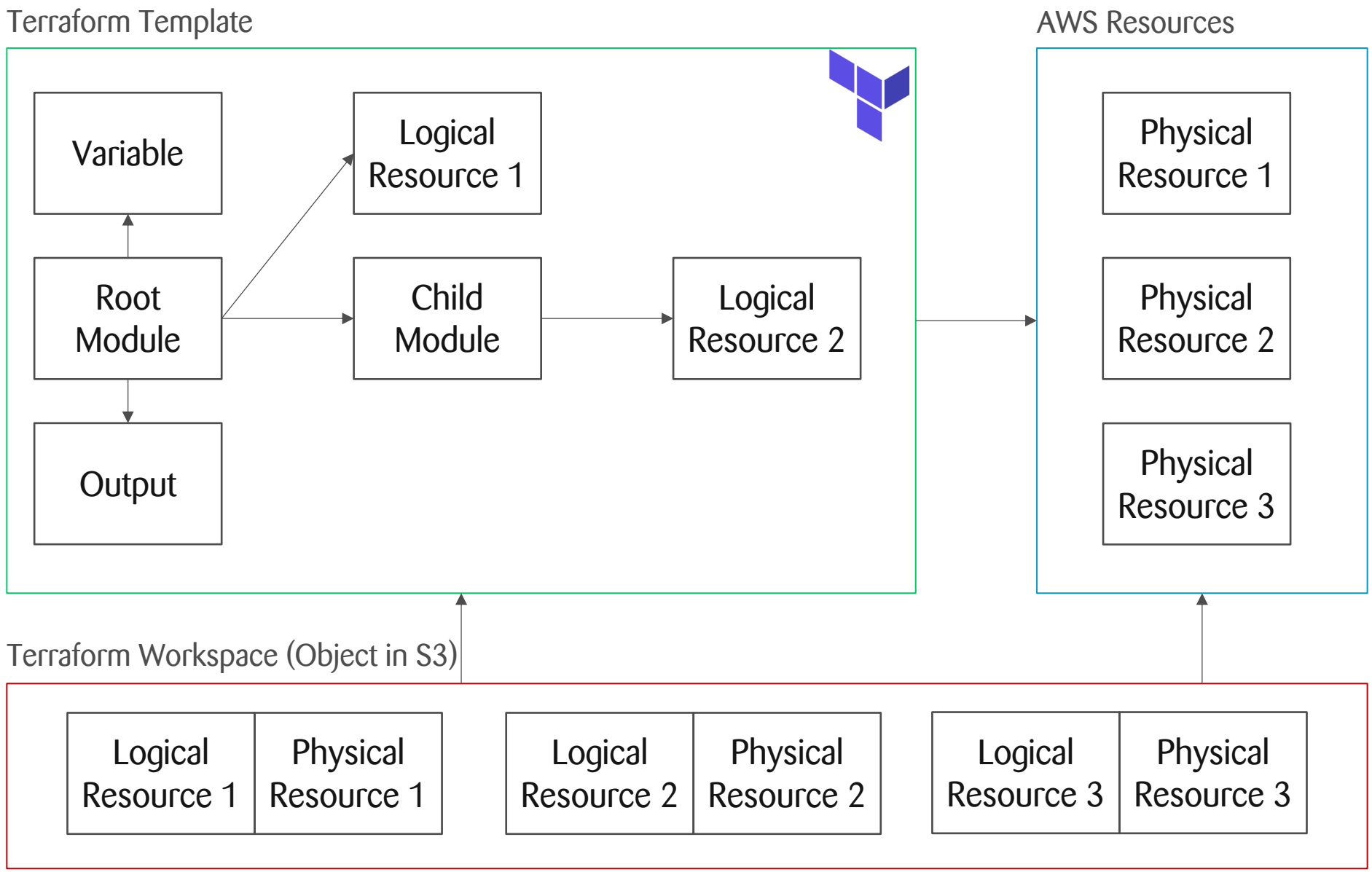
t=2: template version 2 created (logical resources added)



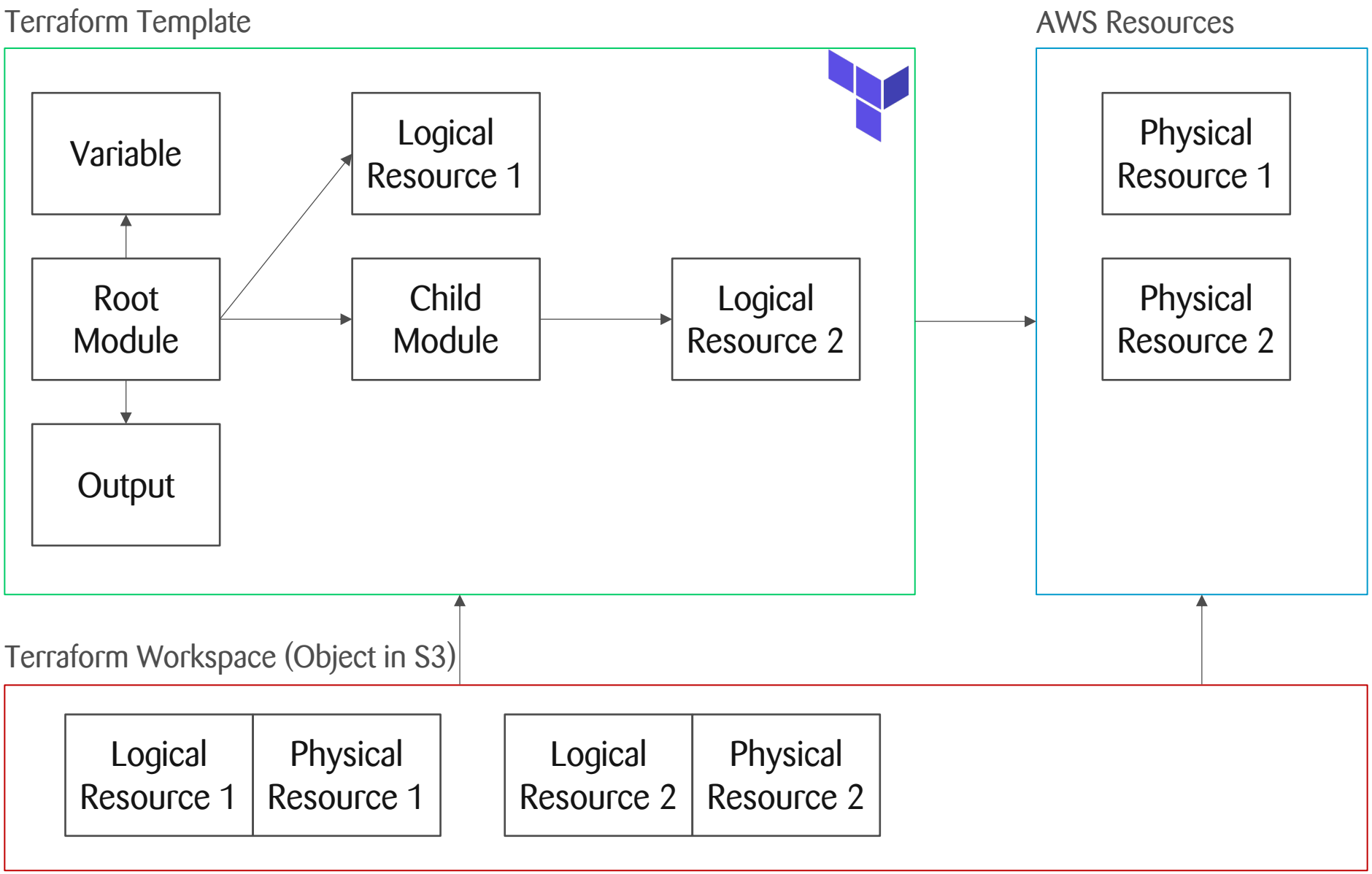
t=3: template version 2 applied (physical resources added)



t=4: template version 3 created (logical resource removed)



t=5: template version 3 applied (physical resource removed)



Basics

Terraform

Module Layout (Filesystem)

Sample (recommended structure and filenames for a minimal module)

```
Directory: RootModuleA
        File: variables.tf
        File: main.tf
        File: outputs.tf

        Directory: ./modules/ChildModule1
                File: variables.tf
                File: main.tf
                File: outputs.tf

        Directory: ./modules/ChildModule<n>
                …

Directory: RootModule<n>
        …
```

Terraform

Provider

- The Amazon Web Services (AWS) provider is used to interact with the many resources supported by AWS
- The provider needs to be configured with the proper credentials before it can be used (static credentials, environment variables, shared credentials file, EC2 Role)

```
provider "aws" {
    version = "~> 2.0"
    region  = "eu-central-1"
}
```

Terraform

Backend

- Terraform must store state about your managed infrastructure and configuration. This state is used by Terraform to map real world resources to your configuration
- A "backend" in Terraform determines how state is loaded
- By default, Terraform uses the "local" backend
- The S3 backend stores the state as a given key in a given bucket on Amazon S3

```
terraform {
    backend "s3" {
        bucket = "terraformchtzbucket"
        key    = "bucketsample/terraform.tfstate"
        region = "eu-central-1"
    }
}
```

Terraform

Module

- A module is a container for multiple resources that are used together
- A module consists of the resources defined in the .tf files in the module directory
- Every Terraform configuration has at least one module, known as its root module
- A module can call other modules, which lets you include the child module's resources into the configuration.
- Modules can also be called multiple times, either within the same configuration or in separate configurations

```
module "bucket_b" {
    source = "./mybucket"
    bucket_suffix = "chtz-testbucket-b"
}
```

Terraform

Resources

- Each resource block describes one or more infrastructure objects, such as virtual networks, compute instances, or higher-level components such as DNS records

```
resource "aws_s3_bucket" "samplebucket" {
    bucket = "${terraform.workspace}-${var.bucket_suffix}"
}

resource "aws_s3_bucket_public_access_block" "samplebucket_nonpublic" {
  count = var.block_public_acls ? 1 : 0
  bucket = aws_s3_bucket.samplebucket.id
  block_public_acls   = true
}
```

Terraform

Data Sources

- Data sources allow data to be fetched or computed for use elsewhere in Terraform configuration
- Each provider may offer data sources alongside its set of resource types

```
data "aws_s3_bucket" "existing_bucket" {
    bucket = "dev-chtz-testbucket-c"
}

resource "aws_s3_bucket_public_access_block" "bucket_nonpublic" {
  bucket = data.aws_s3_bucket.existing_bucket.id
  block_public_acls   = true
}
```


Terraform

Input Variables

- Input variables serve as parameters for a Terraform module, allowing aspects of the module to be customized
- You can set the root module variable using CLI options, environment variables or .tfvars files (which are either loaded automatically or must be provided via CLI options)

```
variable "block_public_acls" {
    type = bool
    default = true
}

resource "aws_s3_bucket_public_access_block" "samplebucket_nonpublic" {
  count = var.block_public_acls ? 1 : 0
  bucket = aws_s3_bucket.samplebucket.id
  block_public_acls   = true
}
```

Terraform

Local Values

- A local value assigns a name to an expression, allowing it to be used multiple times within a module without repeating it

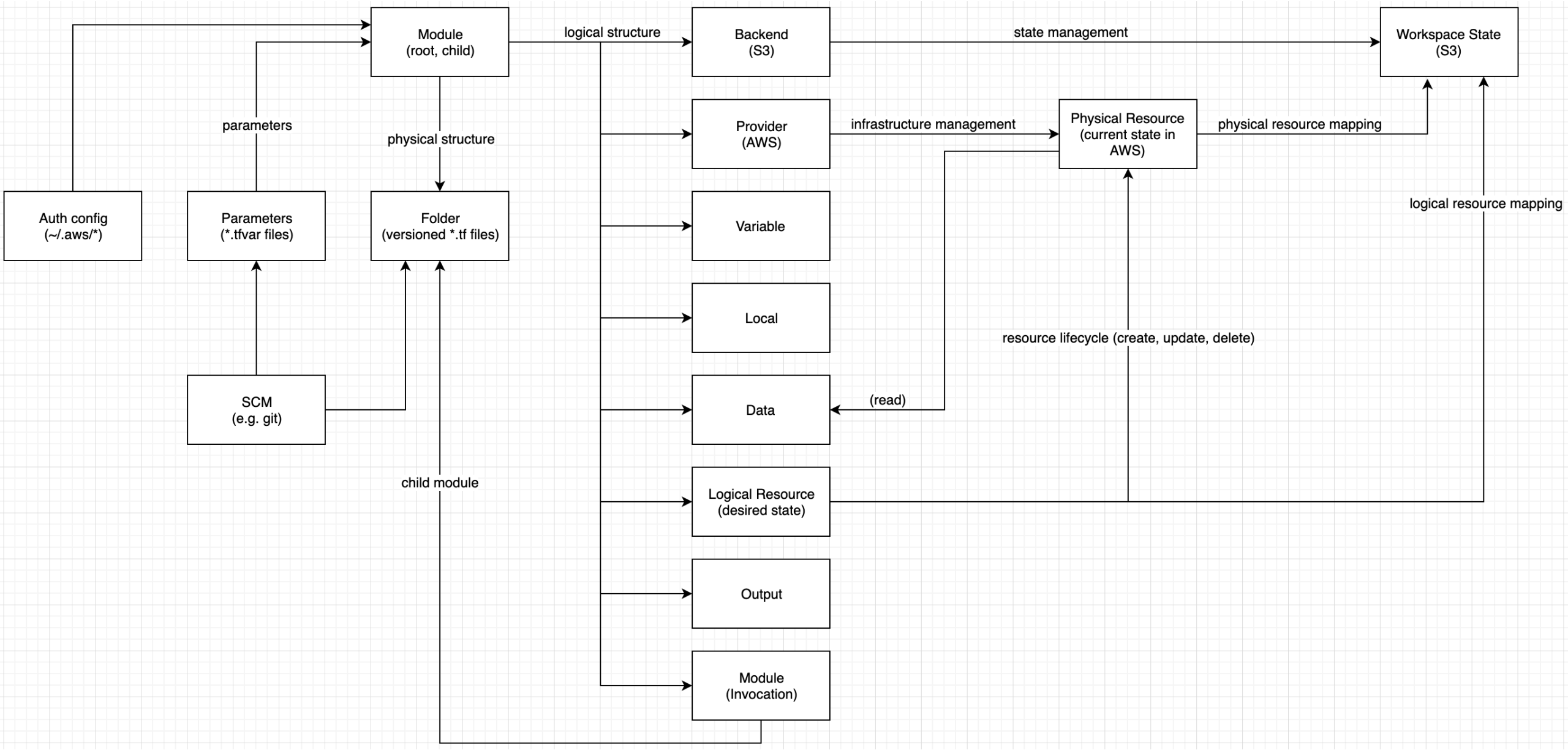
```
locals {
  common_tags = {
      TFWorkspace = terraform.workspace
  }
}

resource "aws_s3_bucket" "samplebucket" {
    bucket = "${terraform.workspace}-${var.bucket_suffix}"

    tags = merge(local.common_tags, {
        Suffix : var.bucket_suffix
    })
}
```

Terraform

Output Values

- Output values are like the return values of a Terraform module
- A child module can use outputs to expose a subset of its resource attributes to a parent module
- A root module can use outputs to print certain values in the CLI output after running terraform apply
- When using remote state, root module outputs can be accessed by other configurations via a terraform_remote_state data source

```
output "bucket" {
    value = aws_s3_bucket.samplebucket.id
}
```


Module
(root, child)
logical structure
Backend
(S3)
state management
Workspace State
(S3)
parameters
physical structure
Provider
(AWS)
infrastructure management
Physical Resource
(current state in
AWS)
physical resource mapping
Auth config
(~/.aws/*)
Parameters
(*.tfvar files)
Folder
(versioned *.tf files)
Variable
logical resource mapping
Local
resource lifecycle (create, update, delete)
SCM
(e.g. git)
Data
(read)
child module
Logical Resource
(desired state)
Output
Module
(Invocation)

Templating

Terraform

Templating

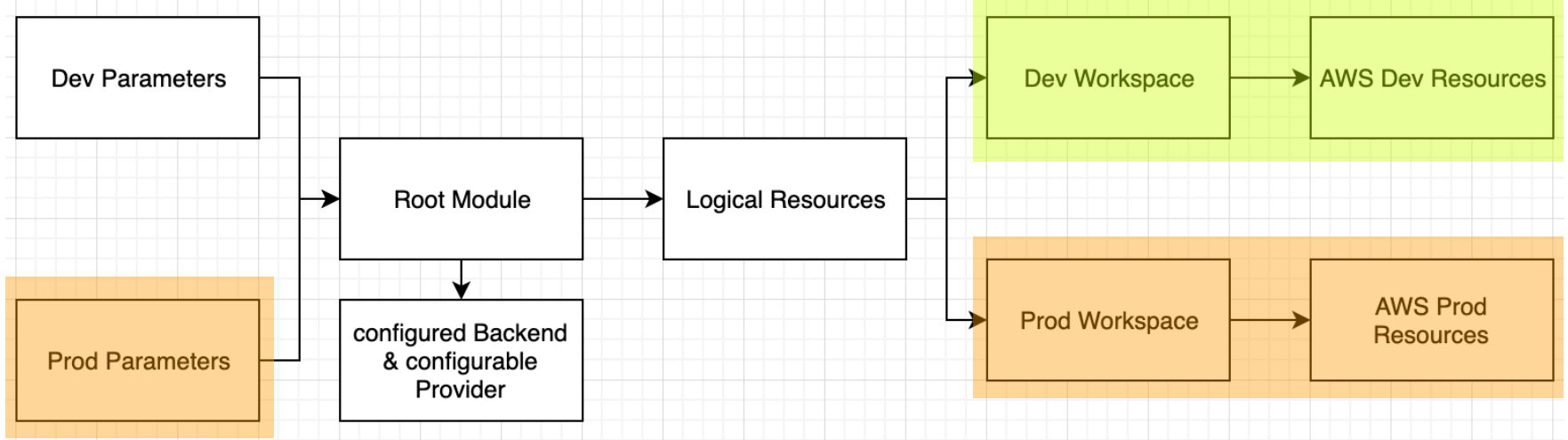

```
variable "alist" {
  type = "list"
  default = [ {name = "Hallo", value="Welt"}, {name = "hello", value="world"} ]
}

data "template_file" "example" {
  template = <<EOT
{
  "environment":
    [ %{ for i, item in var.alist ~}
      { "name": ${jsonencode(item.name)}, "value": ${jsonencode(item.value)} }
      %{ if i < length(var.alist)-1 ~} , %{endif ~}
    %{ endfor ~} ],
  "better_environment" : ${jsonencode(var.alist)}
}
EOT
}
```

Workspaces

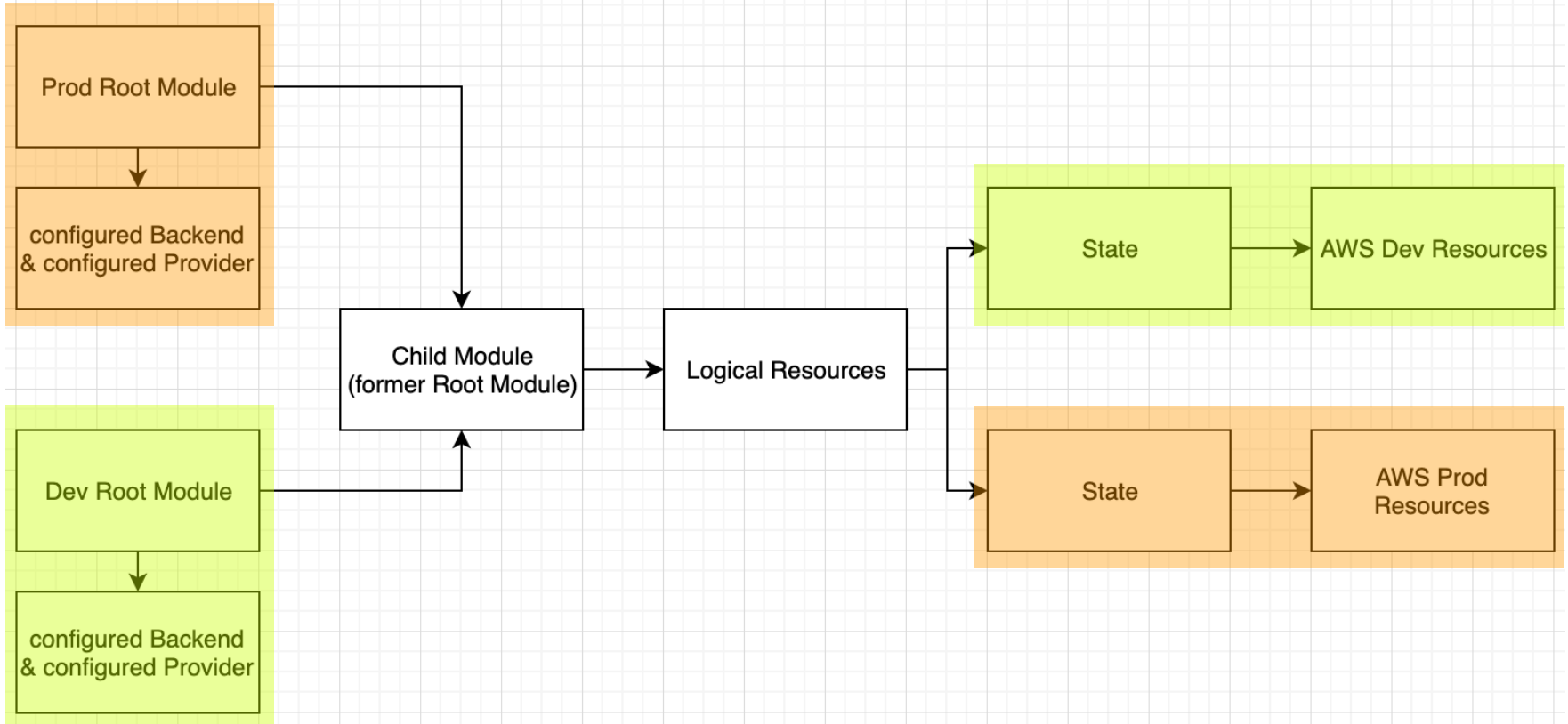
Terraform

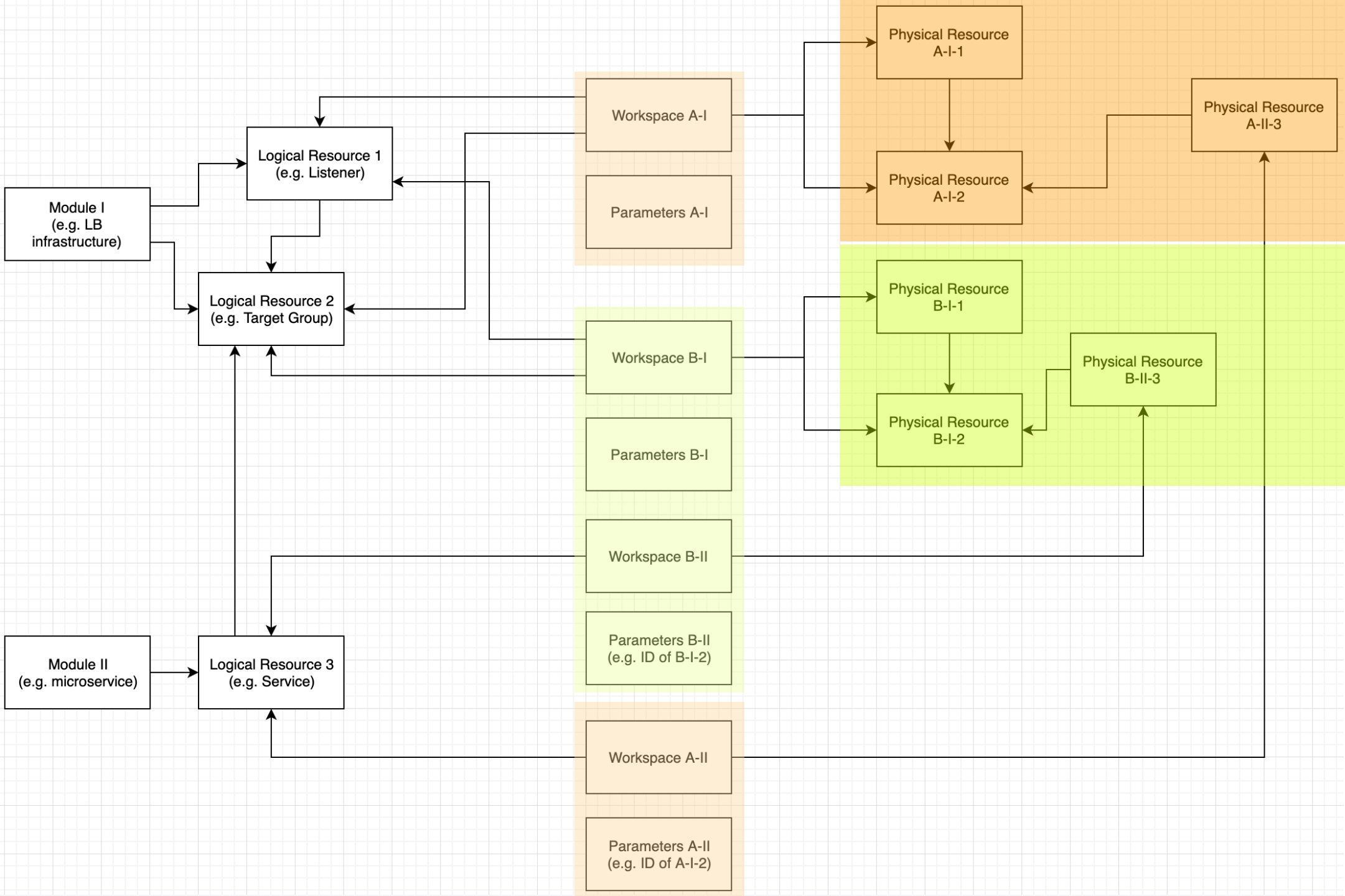
Workspaces

- The S3 backend supports multiple named workspaces, allowing multiple states to be associated with a single configuration
- Within your Terraform configuration, you may include the name of the current workspace using the ${terraform.workspace} interpolation sequence
- When Terraform is used to manage larger systems, teams should use multiple separate Terraform configurations that correspond with suitable architectural boundaries within the system. Workspaces alone are not a suitable tool for system decomposition.
- The input variable values passed to root modules are not stored in the workspace state.
- Name your workspaces with both their component and their environment (e.g. vpc-dev)



XOR




Physical Resource A-I-1
Physical Resource A-II-3
Workspace A-I
Logical Resource 1 (e.g. Listener)
Physical Resource A-I-2
Module I (e.g. LB infrastructure)
Parameters A-I
Logical Resource 2 (e.g. Target Group)
Physical Resource B-I-1
Workspace B-I
Physical Resource B-II-3
Parameters B-I
Physical Resource B-I-2
Workspace B-II
Parameters B-II (e.g. ID of B-I-2)
Module II (e.g. microservice)
Logical Resource 3 (e.g. Service)
Workspace A-II
Parameters A-II (e.g. ID of A-I-2)

Samples

Samples

see Github

- `bucketsample` – workspaces, variables, backend, provider, resources, tagging
- `childmodules` – modules, breaking and fixing terraform state
- `datasources` – datasources
- `multiaccount` – multiple root modules instead of workspaces
- `strings` – JSON and strings

Q & A