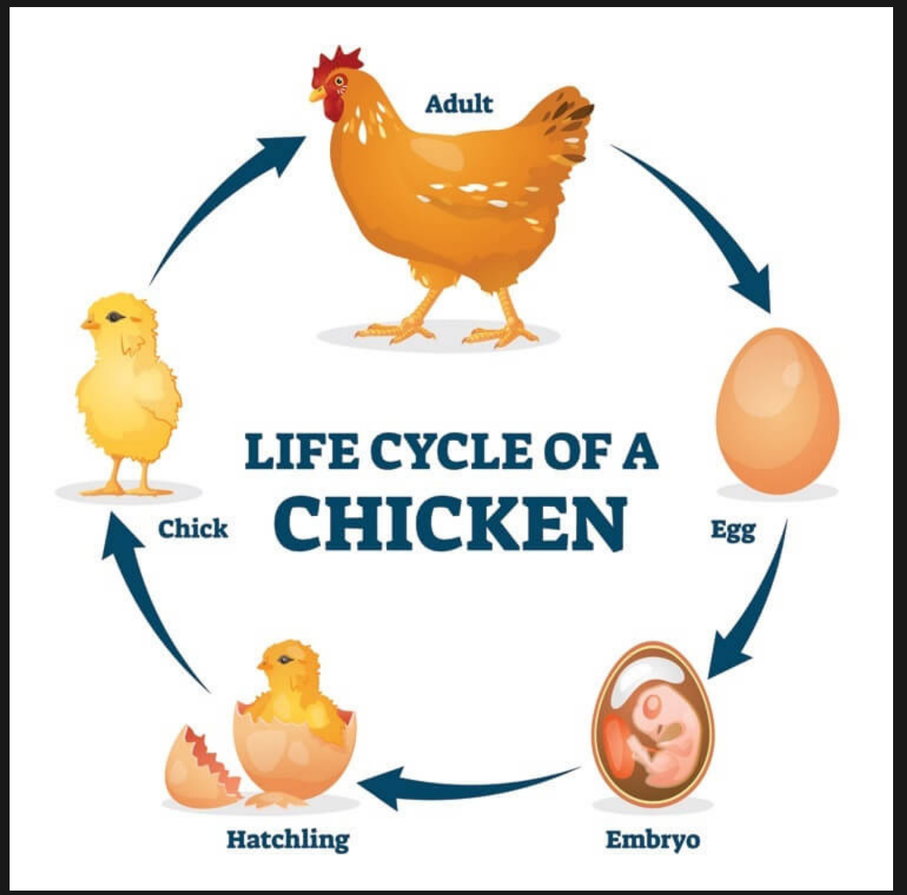
Terraform Resource Lifecycle

HashiCorp
Terraform



HashiCorp
Terraform

Resource Lifecycle

- A resource is defined in code with the resource block
  - Resource is created only if ran through terraform apply, and recorded record in state file
- Resource is recorded into state file
  - Allows Terraform to fully manage the resource
  - Creation, updating, destroying, grabbing metadata, etc.
  - Terraform consults state file when resource definition is altered, through code or manually
- Terraform makes proposed changes based on difference between code and state
- When resource is applied and created, it's attributes can be used by other Terraform configurations
  - Every resource exposes set of attributes
  - Create internal dependencies between resources by leveraging exposed attributes
  - Can use data source block to get values from AWS into Terraform

Lifecycle Arguments

- Terraform provides global lifecycle block which alters the behavior of Terraform to that resource
- The lifecycle nested block is available for every resource
  - Not Provider specific, every resource has standard lifecycle behavior
  - This is a nested block, meaning it must be defined inside a resource

```
resource "azurerm_resource_group" "example" {
  # ...

  lifecycle {
    create_before_destroy = true
  }
}
```

Lifecycle Arguments

- There are four avaialble arguments for the lifecycle block
  - create_before_destroy
  - prevent_destroy
  - ignore_changes
  - replace_triggered_by
- The most commonly used arguments are create_before_destroy, prevent_destroy, ignore_changes
- Lifecycle arguments should be used sparingly!
  - Alters the default behavior of how resource is created and updated
  - Can lead to many unintended side effects, potentially corrupting state file
  - ignore_changes is safe
    - create_before_destroy, prevent_destroy has significant impact

create_before_destroy

- Accepts a boolean value (true or false)
- By default, if Terraform detects a change and must update a resource, it will first destroy the resource if the update cannot be made in-place
- If create_before_destroy is enabled, then Terraform reverses that behavior
  - A new resource gets created, with your proposed changes, then the old resource is destroyed
- Useful if the resource must be highly available, minimize downtime, or must maintain some kind of state
- Sounds like a no-brainer to enable, but you must fully understand potential side affects
  - Difficult to fully understand all side effects
  - Can easily corrupt state file depending on resource type

prevent_destroy

- Accepts a boolean value (true or false)
- By default, if Terraform detects a change and must update a resource, it will first destroy the resource if the update cannot be made in-place
- If prevent_destroy is enabled, then Terraform will reject any plans that proposes the destruction of the resource, and provide an error
- This setting protects the resource from accidentally being deleted
- Useful for sensitive resources which under no circumstances, must be removed
  - State buckets, Cognito user pools, Highly available load balancers, etc.
- Similar to create_before_destroy, think about ramifications first

ignore_changes

- Accepts a list of attribute names
- When Terraform detects change in a resource argument, it will propose an in-place update, or a full re-creation of the resource
- If argument/attribute is supplied to ignore_changes, Terraform will ignore request to update the resource
- Commonly used when refactoring / changing existing modules
  - You don't want existing infrastructure to be affected by the changes
- Some resources get updated by CICD, Lambda is common example

```
resource "aws_instance" "example" {
  # ...

  lifecycle {
    ignore_changes = [
      # Ignore changes to tags, e.g. because a management agent
      # updates these based on some ruleset managed elsewhere.
      tags,
    ]
  }
}
```

HashiCorp Terraform

ignore_changes

- If resource has yet to be created or is destroyed, Terraform will create resource like normal
  - The ignore_changes lifecycle is ignored during the creation phase
- Only the attributes defined by the resource block are valid
  - As always, visit the Terraform documentation to get list of all available attributes

```
resource "aws_instance" "example" {
  # ...

  lifecycle {
    ignore_changes = [
      # Ignore changes to tags, e.g. because a management agent
      # updates these based on some ruleset managed elsewhere.
      tags,
    ]
  }
}
```

HashiCorp Terraform