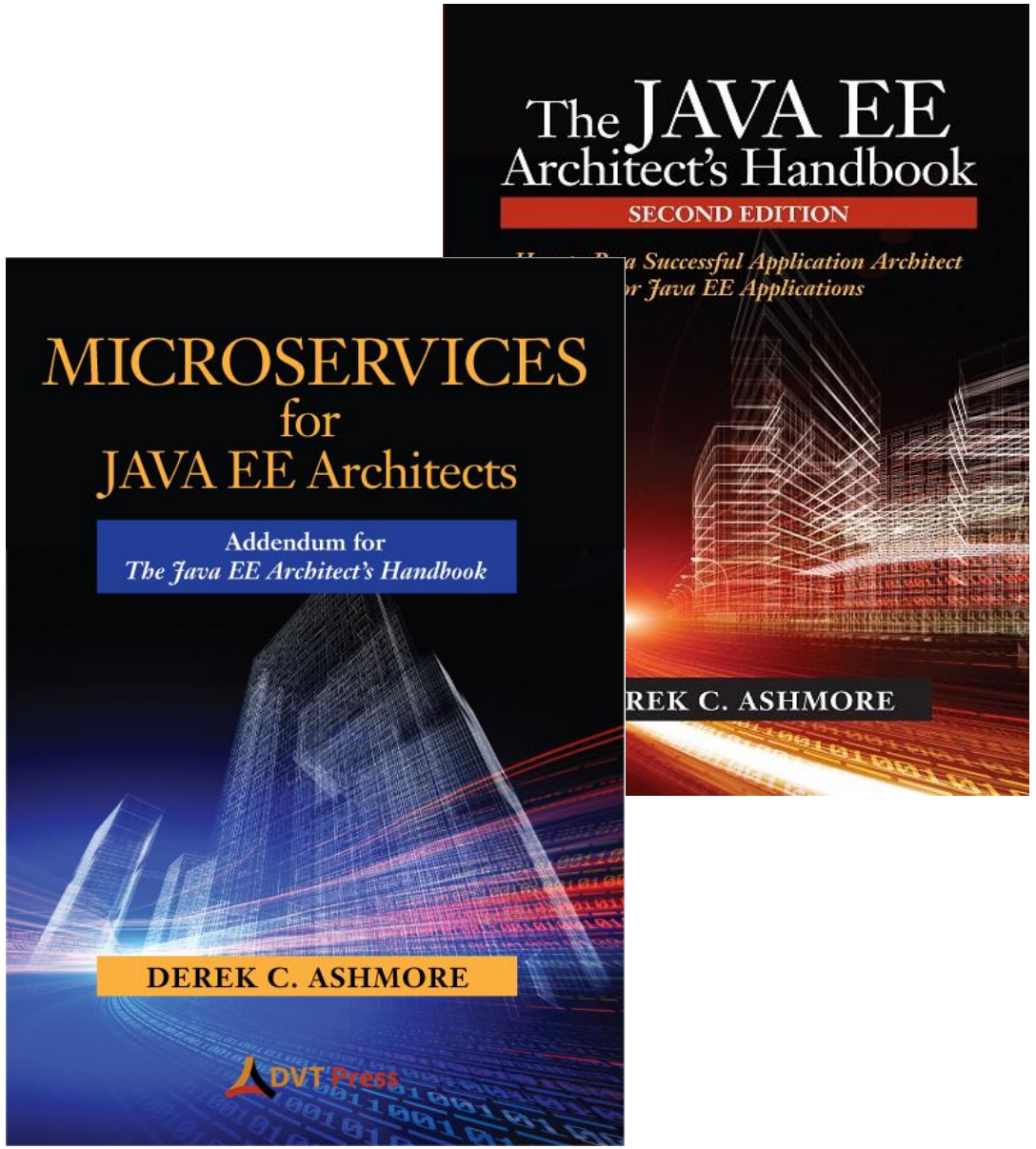
Terraform Infrastructure as Code
Best Practices and Common Mistakes

Given by Derek C. Ashmore
Code PaLOUsa 2022
August 18, 2022

Who am I?

- Professional Geek since 1987
- AWS since 2010
- Azure since 2017
- Terraform since the 0.5.x days
- Specialties
  - Application Transformation
  - Infrastructure Automation
- Yes – I still code!

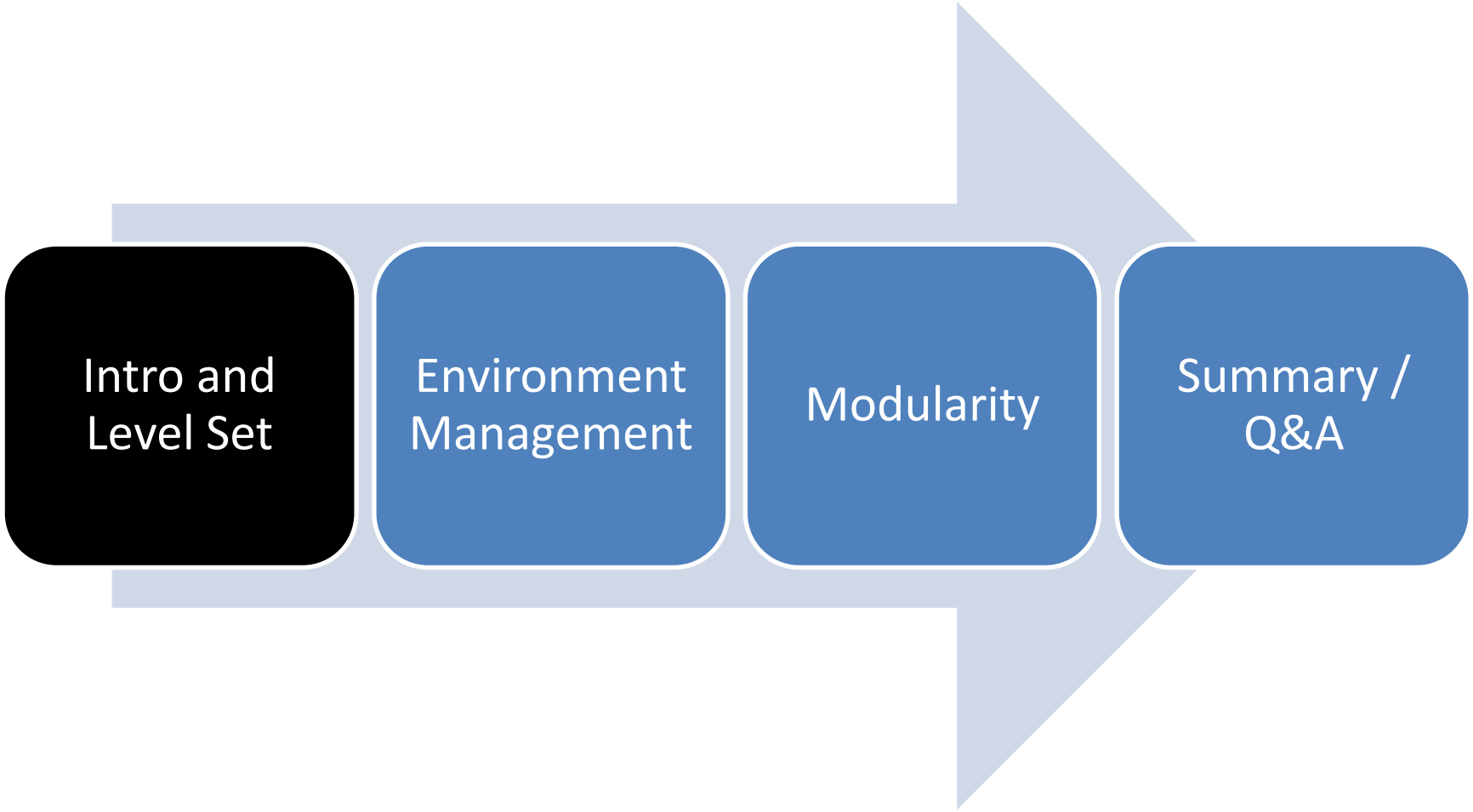


Asperitas CONSULTING

Discussion Resources

- This slide deck
  - https://www.slideshare.net/derekashmore/presentations
- Sample code on my Github
  - https://github.com/Derek-Ashmore/
- Slide deck has hyper-links!
  - Don’t bother writing down URLs
- Assumptions
  - You have used Terraform (at least played with it)
  - You know basic functionality

Agenda



Terraform Terminology

- Resources
  - Controls a cloud asset
- Data lookup
  - Searches for cloud assets
- Variables
  - Different values per context
- Function and Expressions
  - Built-ins that gather/manipulate values
- Configuration vs Module
  - Terraform Module designed for reuse
  - Terraform configuration is the outer layer
- TFVars files
  - Provides variable input for an environment
    - Like properties file
  - Used with the `-var-file` option

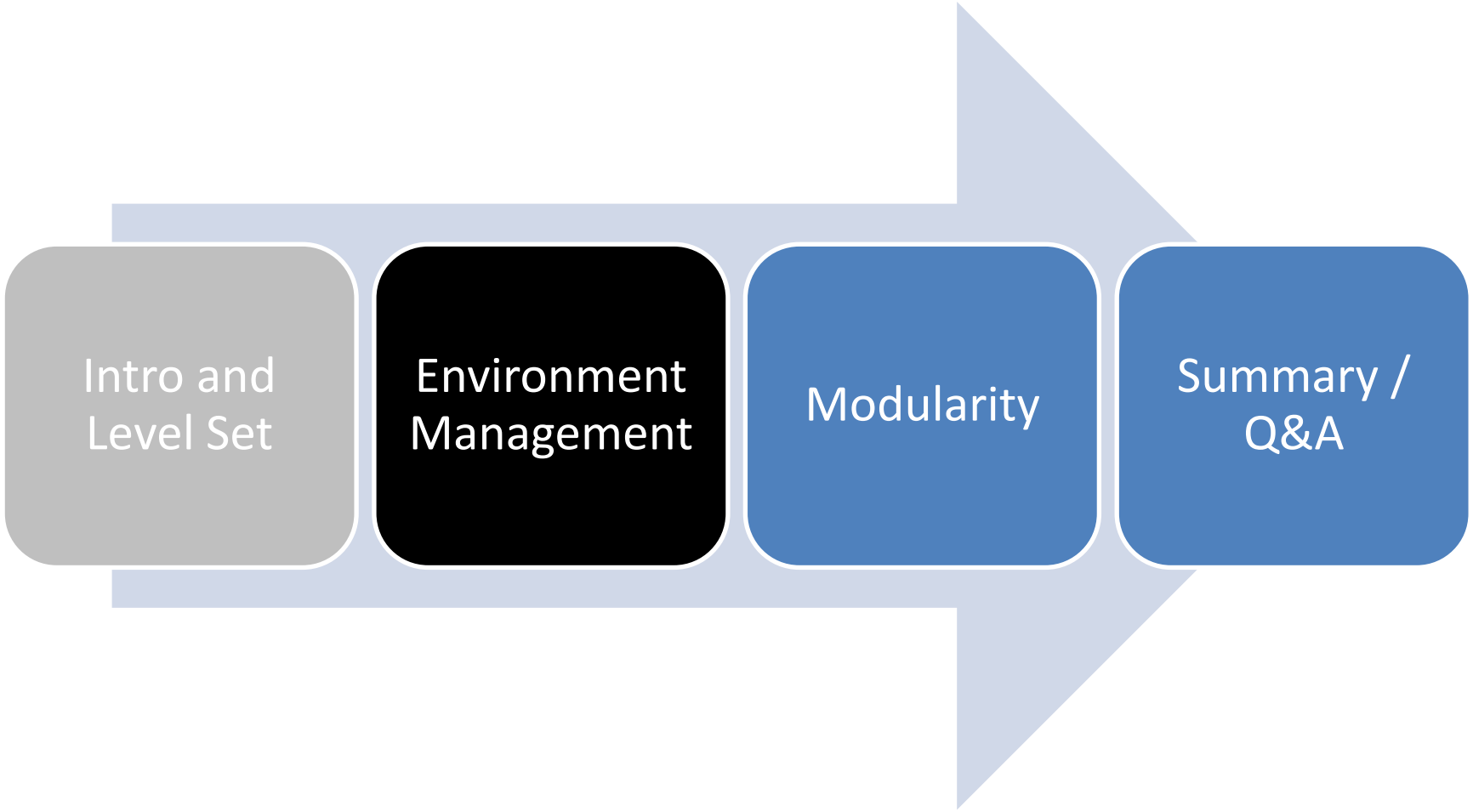
How Terraform Works

- Terraform is “Declarative”
  - Like SQL
- Reads all files with extension .tf
  - Figures out execution order
- Supports variables and functions
- Plugin architecture
  - Supports many clouds and products

```
resource "azurerm_resource_group" "test_rg" {
  name           = var.resource_group_name
  location       = var.location
  tags           = var.tags
}

resource "azurerm_virtual_network" "my_vnet" {
  name                = "my_vnet"
  address_space       = ["10.0.0.0/16"]
  location            = azurerm_resource_group.test_rg.location
  resource_group_name = azurerm_resource_group.test_rg.name
}
```

```
data "azurerm_client_config" "current" {}

output object_id {
  value = data.azurerm_client_config.current.object_id
}
```

Agenda



Project Structure

- Separate Configurations from Modules
  - Documents what's designed for reuse and what is not
- TFVars files provide environment specifics
  - All environments use the same automation
  - Easy to add environments
  - Different back-end state per environment

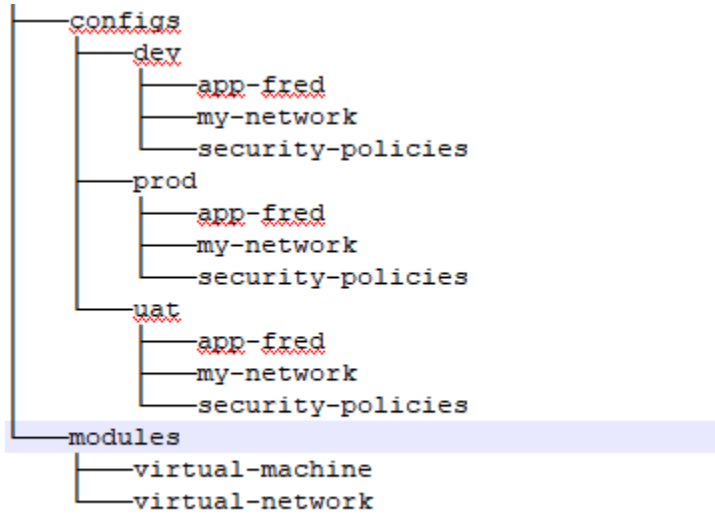
```
├──configs                          # Terraform Configurations
│   ├──app-fred
│   │   └──environments             # Environment tfvars files for different environments
│   ├──my-network
│   │   └──environments
│   └──security-policies
│       └──environments
└──modules                          # Terraform Modules
    ├──virtual-machine
    └──virtual-network
```

```
├──configs
│   ├──app-fred
│   │   └──environments
│   │           dev.tfvars
│   │           prod.tfvars
│   │           sandbox.tfvars
│   │           uat.tfvars
│   │
│   ├──my-network
```

```
terraform plan -var-file environments/dev.tfvars
terraform apply -var-file environments/dev.tfvars
terraform destroy -var-file environments/dev.tfvars
```

Project Structure Anti-Pattern

- Separate configurations per environment
- The good
  - Code is often simpler
  - Easier to add/subtract capabilities per environment
  - Separate state if using local file system default
- The bad
  - Has code duplication
  - Harder to establish new environments
  - Environments can be inconsistent
    - Works in dev, but not prod



Making Environment Differences Configurable

- Intended environment differences drive use of anti-pattern
- Use Conditionals
  - Boolean indicators
    - Resources using count
    - Dynamic blocks

```
resource "azurerm_private_endpoint" "key_vault" {
  count                     = var.private_link_ind ? 1 : 0

  name                      = var.vault_endpoint_name
```

```
locals {
  # Needed as dynamic blocks *only* accept maps for for_each
    secure_resources_map = var.secure_resources ? { one = "one"} : {}

}

resource "azurerm_container_registry" "container_registry" {
  name                      = var.container_registry_name
  resource_group_name       = var.resource_group_name
  location                  = var.location

  # Only 0 or 1 entry depending on var.secure_resources
  dynamic "network_rule_set" {
    for_each = local.secure_resources_map
    content {
      default_action = "Deny"
    }
  }
}
```

Optional configuration through ‘try’

- Use for complex inputs with optional fields
- `Try` suppresses exceptions
- Specify Terraform defaults with `null`

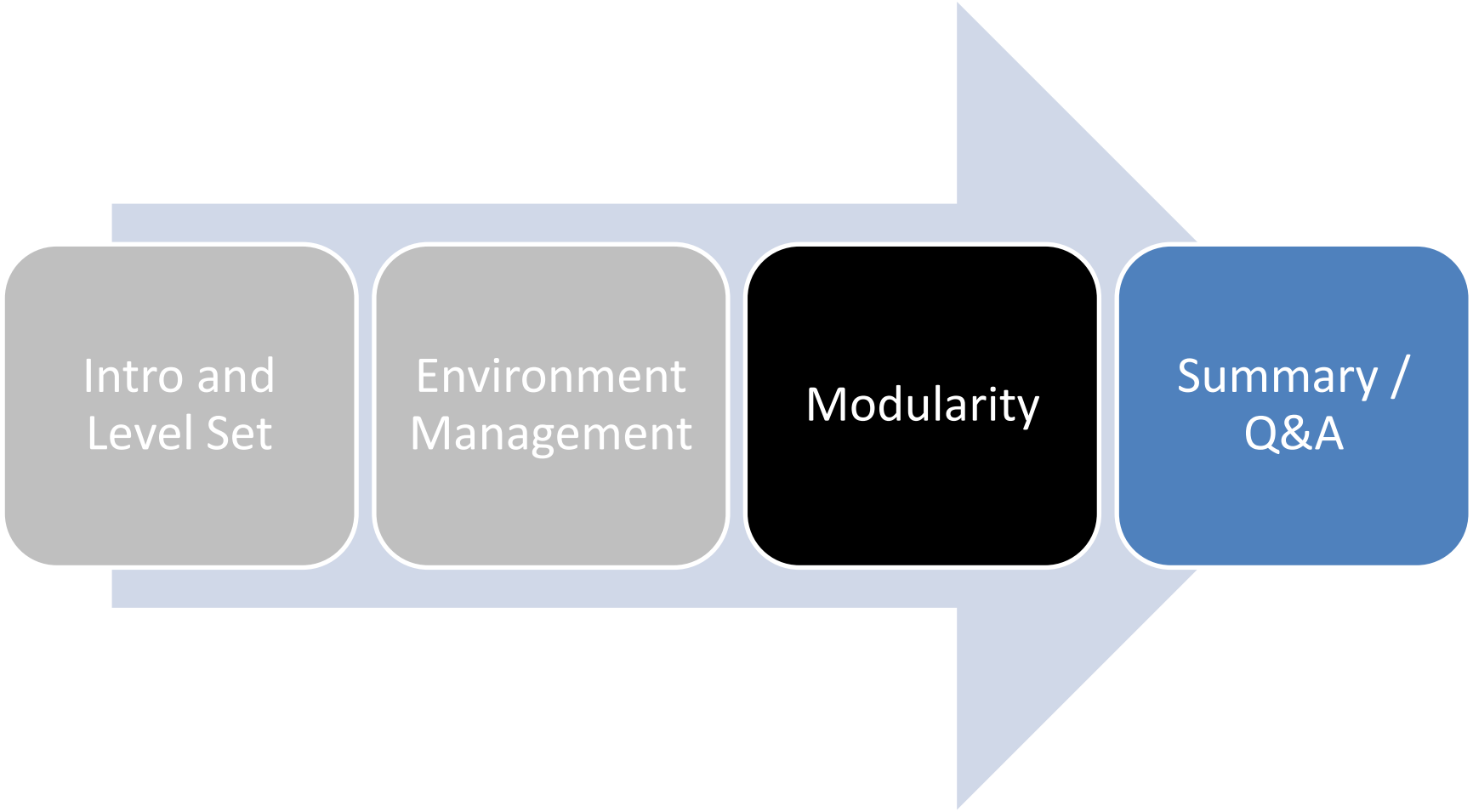
```
# Snippet from tfvars file
site_config = {
    always_on = "false"
}
```

```
locals {
  site_config_map = var.site_config != null ? {one = "one"} : {}
}

resource "azurerm_app_service" "appsvc" {
  name                        = var.app_service_name

  dynamic site_config {
    for_each = local.site_config_map
    content {
      always_on                 = try(var.site_config["always_on"], null)
      app_command_line          = try(var.site_config["app_command_line"], null)
      health_check_path         = try(var.site_config["health_check_path"], null)
    }
  }
```

Environment Management Best Practices

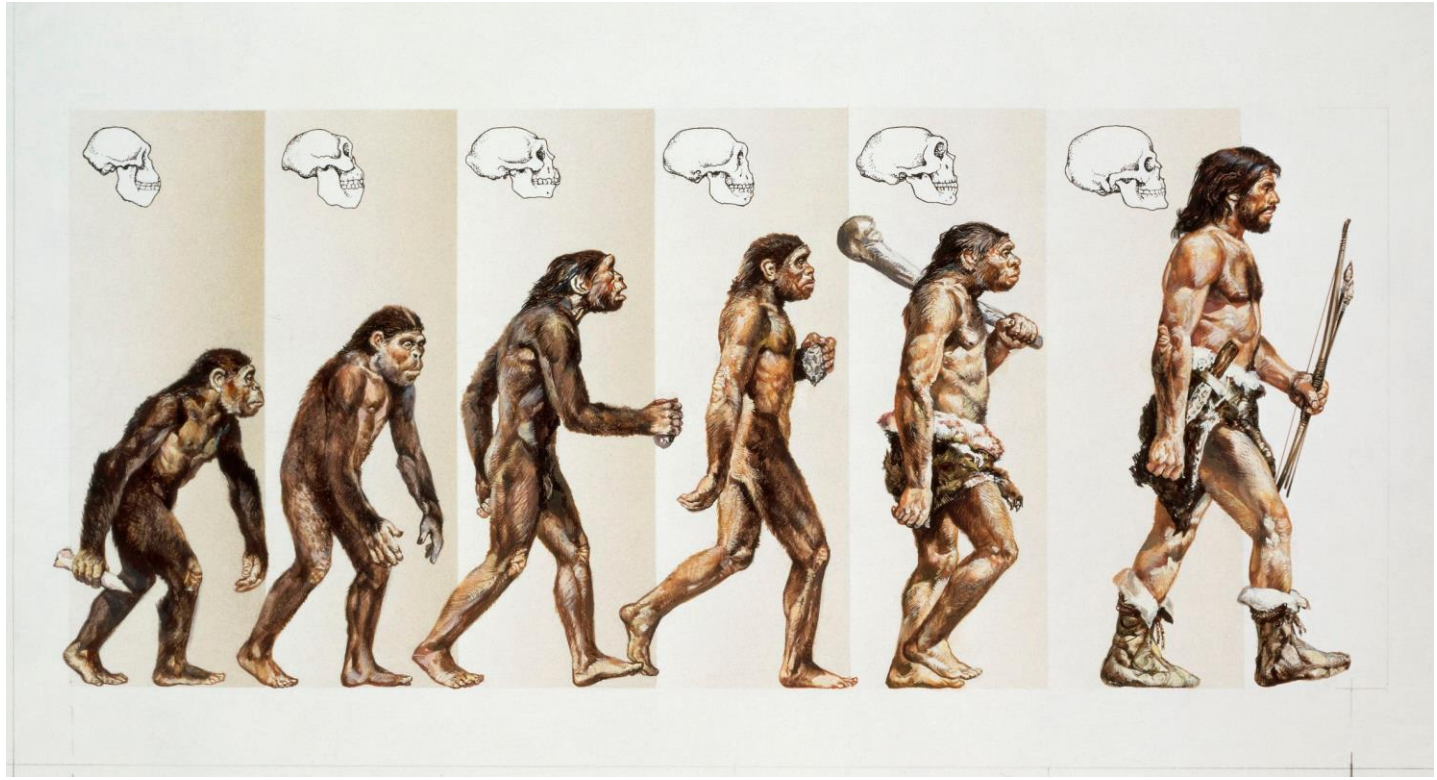
- Always run Terraform through tooling, not on your desktop
  - CI/CD Tools such as Jenkins or Terraform Cloud
  - Benefits
    - Audit history
    - Terraform and provider version control
    - Consistent runtime environment
- Always require a plan before the apply
  - Require approval step before going on to the apply
- Utilize cloud security constructs
  - AWS IAM instance roles for Jenkins agents
  - Azure Managed Identities for Jenkins or Azure DevOps agents
- Always use back-end state

Agenda



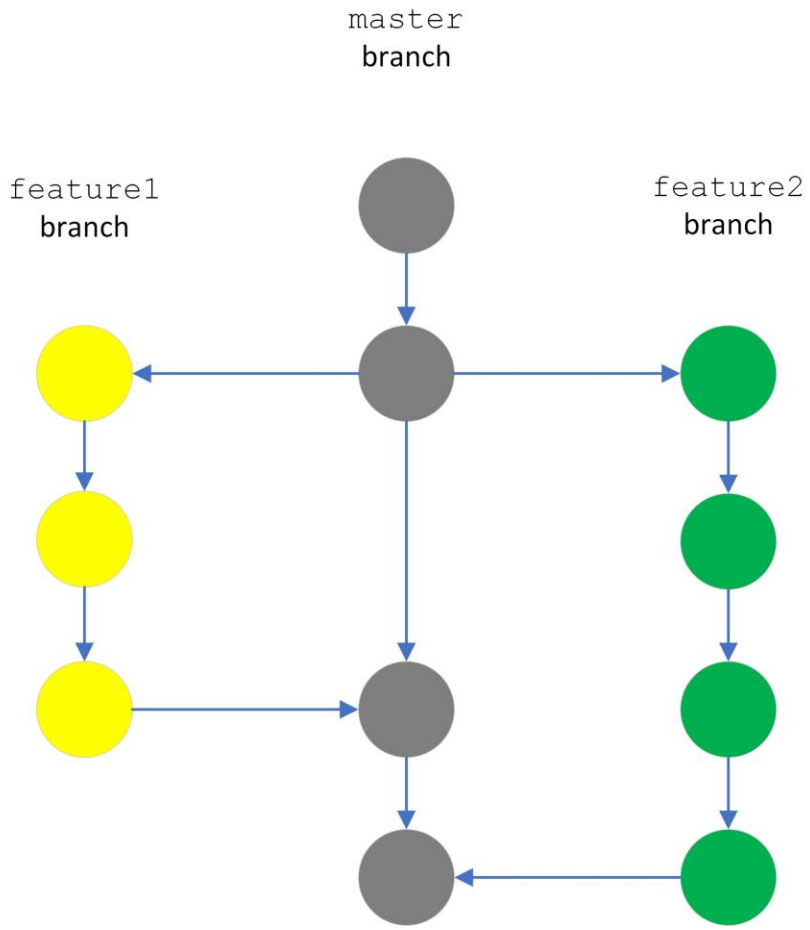
Terraform Usage Evolution

- In the beginning
  - Use Source Control
  - Use Back-end state
- As #Coders grows
  - Feature branches
  - CI/CD Pipelines
- As #Configurations grows
  - Separate repo for modules
    - Or Terraform registry
  - Implement versioning
    - Never use `main/master`!
- Further reading



Feature Branching

- DevOps Team Discipline is Key
- Feature Branches
  - Never edit main/master directly!
  - Update using Pull Requests
- Should live less than one day!
  - Single targeted enhancement
  - One developer only
  - Long-lived branches prone to merge conflicts
  - Prefer `rebase` to `merge`
- Further reading

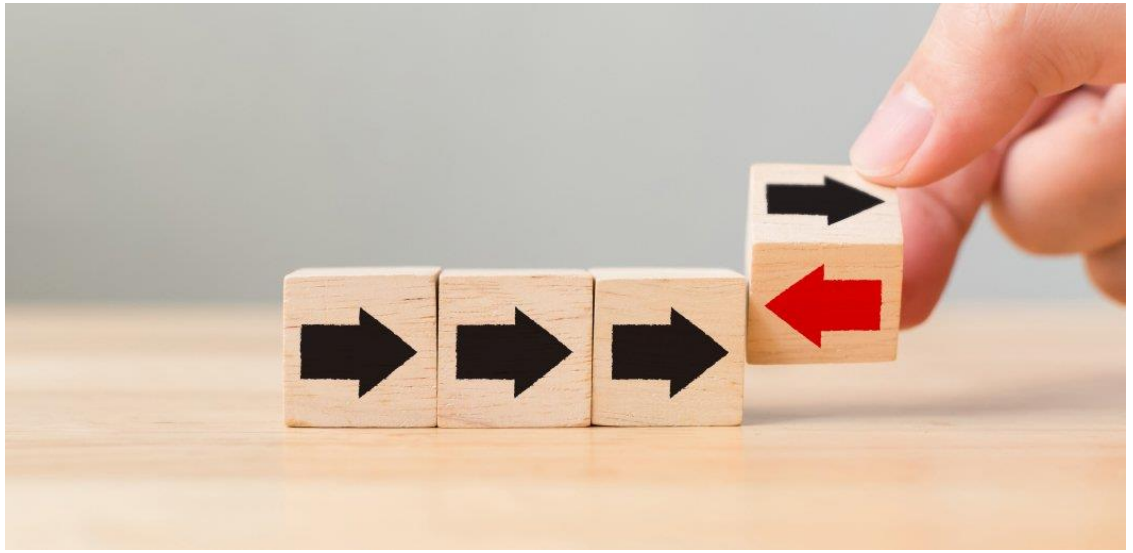


CI/CD Pipelines

- Provides consistent runtime environment
  - Terraform version
  - Cloud security policy
- Audit history / Admin security
- Pipeline approvals
  - Force Plan execution
  - Force manual approval before apply or destroy
  - Automatic “Apply” nullifies benefit of doing the plan

Modularity Anti-Patterns

- All of these examples come from the field
  - Module creation before it's needed
  - Modules that only contain one resource
  - Inappropriate Data lookups in modules
  - Undocumented modules
  - Use modules referencing `main/master`



Module creation before it’s needed

- Should have at least two consumers before module is created
- Classic YAGNI
- Hard to track down consumers after release
    - Impossible to remove unused modules

Modules that only contain one resource

- Amounts to a thin proxy
  - No value-add
- Unnecessary complexity
- Not as well documented as the underlying Terraform resource
- Every module should have at least two resources!

```
resource "azurerm_resource_group" "resource_group" {
  name     = var.name
  location = var.location
  tags     = var.tags
}

output "id" {
  value = azurerm_resource_group.resource_group.id
}
```

Inappropriate Data lookups in modules

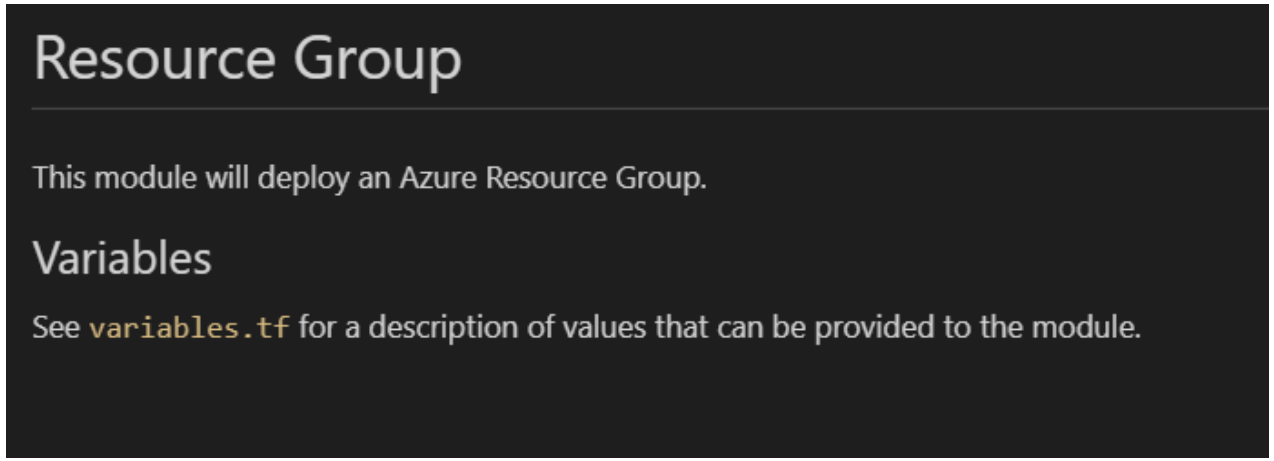
- Data lookups fail if nothing is found
  - Error if the consumer configuration creates the resource
- Makes assumptions about execution context
- Data lookups belong in configurations, not modules, as they do know context

```
data "azurerm_resource_group" "resource_group" {
  name = var.resource_group_name
}

resource "azurerm_linux_virtual_machine_scale_set" "scaled_set" {

  location                        = data.azurerm_resource_group.resource_group.location
  resource_group_name             = data.azurerm_resource_group.resource_group.name
```

```
resource "azurerm_linux_virtual_machine_scale_set" "scaled_set" {

  location                        = var.location
  resource_group_name             = var.resource_group_name
```

Undocumented Modules

- Force consumers to read/understand module code
  - Costs them time
- Makes it hard to use
- All modules should have a `README.md`:
  - Example module call
  - Release Notes
  - Variable list
  - Output list



Use modules referencing main/master

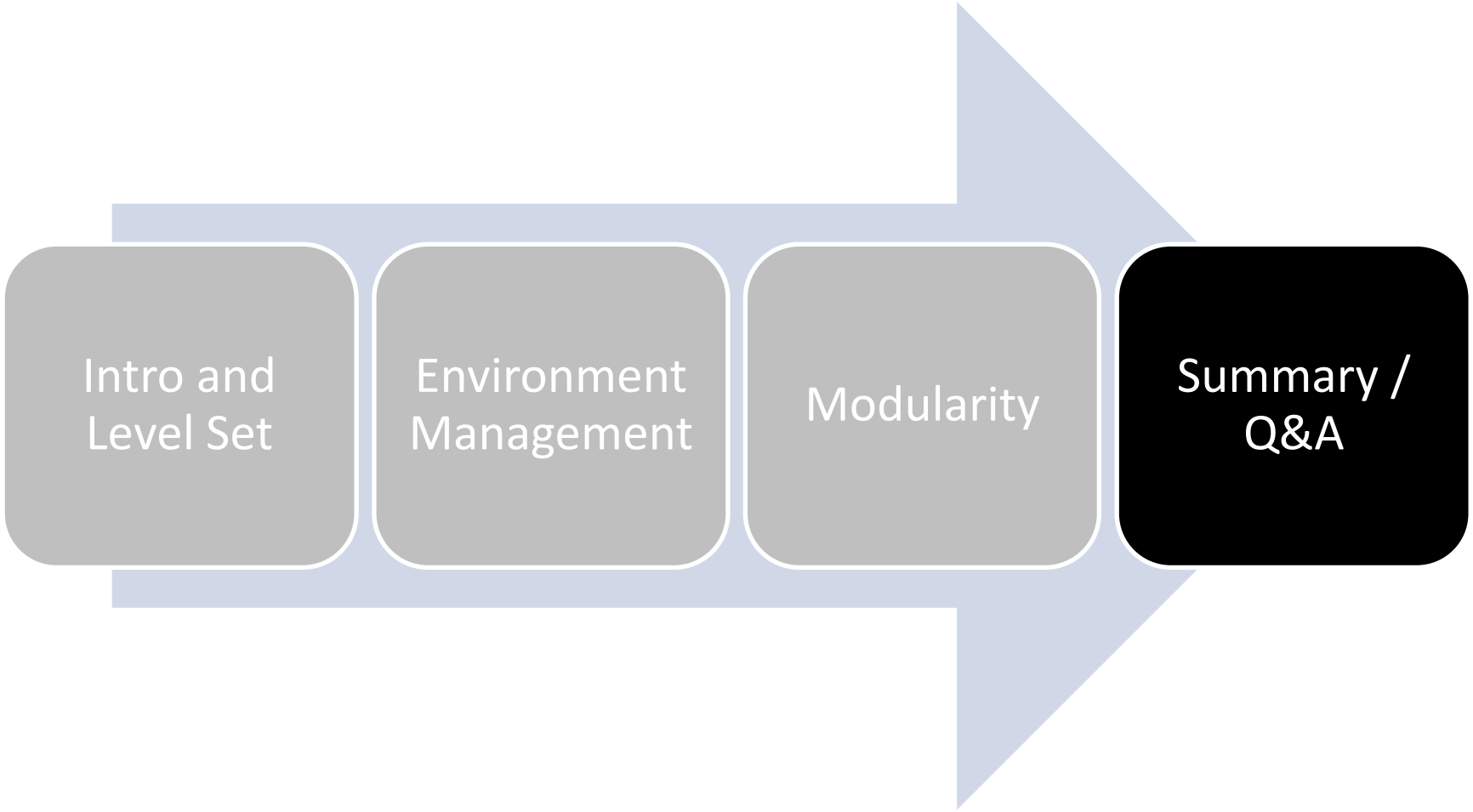
- Recipe for unplanned work
  - Consumers can break unexpectedly when modules change
- Always version modules
- Always consume referencing specific versions
  - Version upgrades are planned work
- Source code

module-with-tag.tf

terraform-function-examples > general > module-git-source > module-with-tag.tf > ...

```
1  module "test" {
2    source = "git::https://github.com/Derek-Ashmore/terraform-function-examples.git//general/collection-ops?ref=0.0.1"
3  }
4
5  output "last_node" {
6      value = module.test.last_node
7  }
```

Agenda



Secrets Handling

- Secrets include
  - Credentials (account/password)
  - SSL Certificates
  - SSH Keys
- Manage secrets separately
  - Digital Vault
    - Terraform looks the secret up
  - CI/CD Pipeline “Secret” variable
- Anti-pattern: Terraform generating password
  - Easy to get out of sync with reality
  - Secrets have different life-cycle

Simplicity is Key

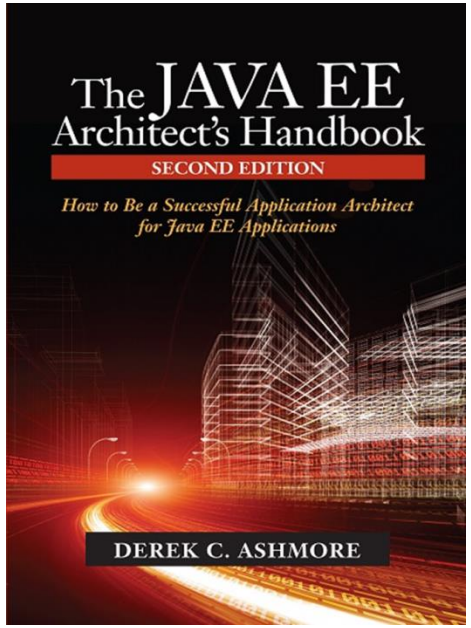
- Eliminate unused variables
- Don’t replicate derived values
  - Derive once in locals and use
- Variable defaults
  - Inappropriate defaults common
    - Environment-specific names
    - Globally unique names

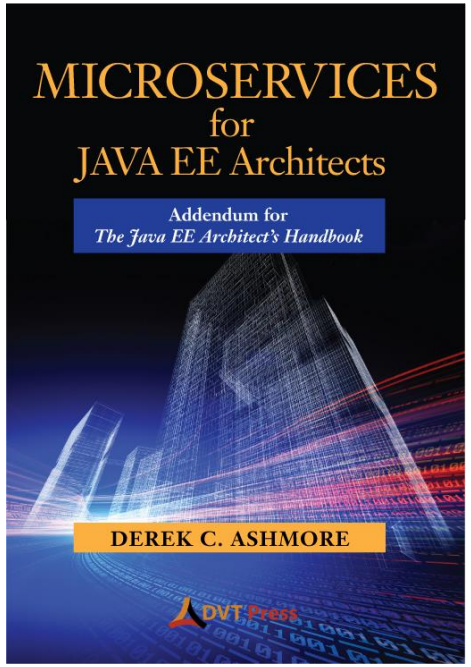
Avoid the Hammer and Nail Problem

- Terraform is good for:
  - Creating cloud assets
  - Changing attributes on cloud assets
- Terraform is not good for:
  - Maintaining content on cloud assets
    - VM configuration management
      - Use Ansible, Chef, etc.
    - Image pipelines
      - Use Packer
  - Don’t “remote control”
    - Use Terraform to execute Ansible or Packer

Thank you!

- Derek Ashmore:
  - Blog: www.derekashmore.com
  - LinkedIn: www.linkedin.com/in/derekashmore
    - Connect Invites from attendees welcome
  - Twitter: https://twitter.com/Derek_Ashmore
  - GitHub: https://github.com/Derek-Ashmore
  - Book: http://dvtpress.com/
- Please fill out the evaluation form!





Asperitas CONSULTING