Scaling Terraform
A “Loose” Guide

Lachlan White
Terraform.. Now What?

```hcl
output "hello_world" {
  value = "Hello, DOTC Melbourne 2022"
}

DOTC-Melbourne-2022/demo $ terraform apply --auto-approve

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

Outputs:

hello_world = "Hello, DOTC Melbourne 2022"
```
Lachlan White
(He/Him)

General Manager of Technology | LAB³

Community Ambassador | HashiCorp
What are we solving?

Technical Complexity
Different platforms have different interfaces, increases the complexity and inconsistency of operations leading to an increase of cost across these operations.

Terraform creates a layer of abstraction, in the form of a workflow-level abstraction, rather than resource-level abstraction.

Enabling a single way of working across platforms.

Organizational Complexity
As infrastructure scales, it requires more to people and teams to maintain it.

How can we delegate ownership of infrastructure, across multiple teams and empower them to work in parallel, without conflict. Whilst acknowledging the uniqueness of ownership and responsibility across those domains?
Phases of Scale

Manual

- Provisioned via UI or CLI
- No naming standard
- No traceability

Semi-Automated

- Combination of UI & CLI
- Limited traceability
- Rollbacks are hard due to differing ways of working

Infrastructure As Code

- Terraform OSS
- Provisioning & deployment automated
- Source files in vcs
- Some terraform modules linked to architectural patterns

Collaborative IaC

- TFC, TFC4B, TFE
- Self-service infrastructure
- Consumers and standardizers
- Workspace specific controls
- Remote module repository
Journey to Successful Scale

Does this do what I need it to do?

Experiment

Quick Wins

Terraform Cloud

Enterprise Controls, Centralise state management etc

Collaborative IaC

Self-Service, controlled workflows designed to accelerate outcomes

Standardisation

Defining standards of structure, ways of working and contribution

Unlock Scale

Industrialise and scale the approach, to maximise the return on your investment

Modules

Creation of reusable, building blocks.

Where can I add value quickly?
Modules

- Reusable
- Scalable
- Abstraction
- True Value
Modules

Should be created when:

▪ One or more resources are used in direct conjunction with each other
  – Such as NICs and Disks added to a VM Resource to abstract for a standard VM Module
▪ One or more modules are used in direct conjunction with each other, to form a standard pattern or architecture.
▪ Opinionated configuration adds additional value, in which repetition would otherwise ensue; Complex Validation, Security configurations etc.

Should not be created when

▪ To create a simple wrapper with no additional value:
  – Such as an Azure Load Balancer where the configuration is always different
▪ The modules' purpose is to be consumed by another higher-level module, and that is it’s only use case.
Terraform Cloud
Design for Collaboration
Deployment Structure
Module Structure
# Variables

## Standardise, Standardise, Standardise.

### Names
- Avoid the use of shorthand
- Reduce prerequisite knowledge required
- Keep it simple
- Deceivingly difficult

### Descriptions
- Effort here is important
- Specify if required or optional
- Standardise formatting for multi-line descriptions

### Default Values & Type
- Work hand in hand
- Important as you utilize more complex variable types, objects of objects etc.
- Enable abstraction

---
