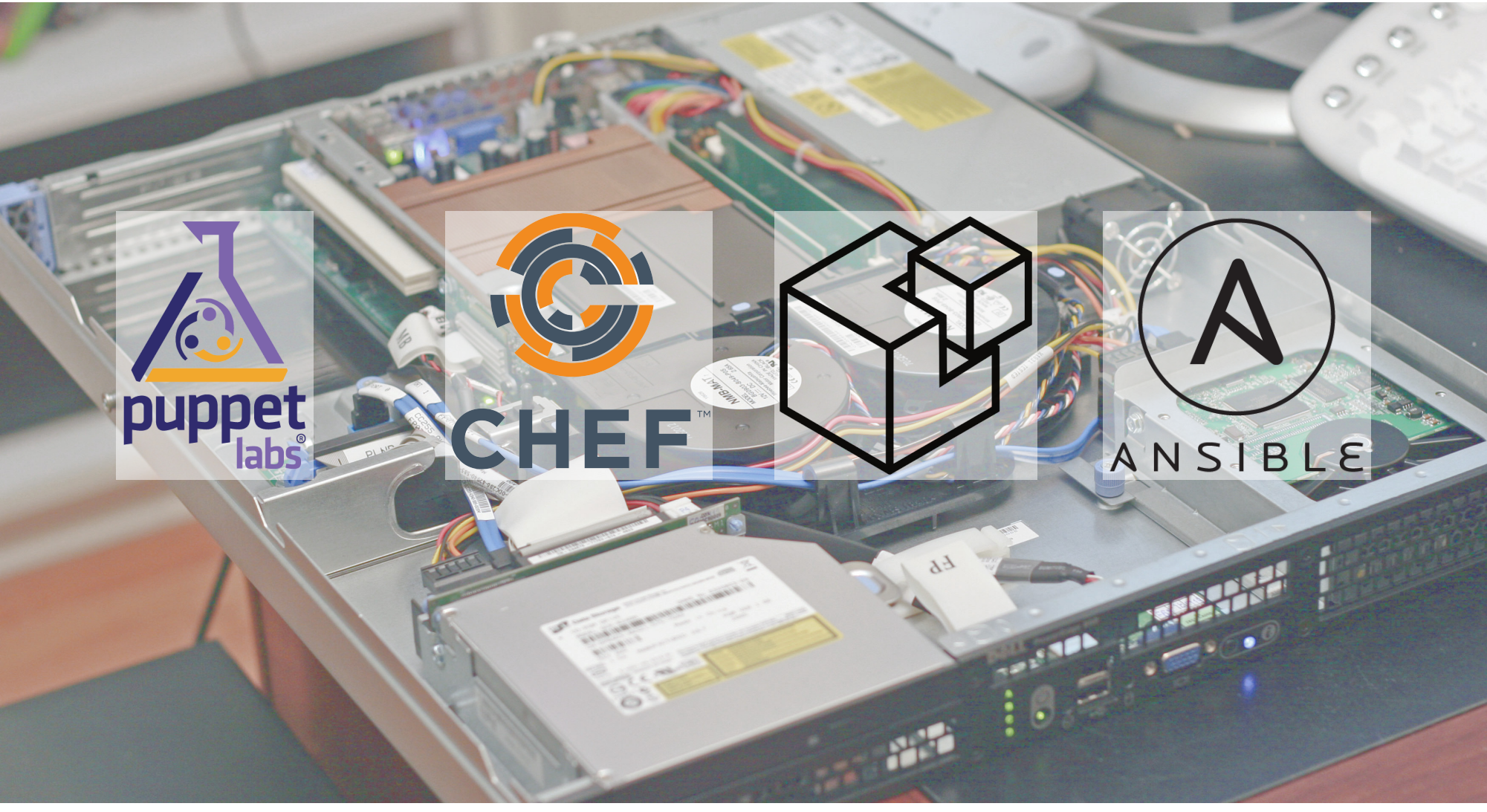
# Terraform:
# Cloud Configuration Management

Martin Schütte

18 April 2017

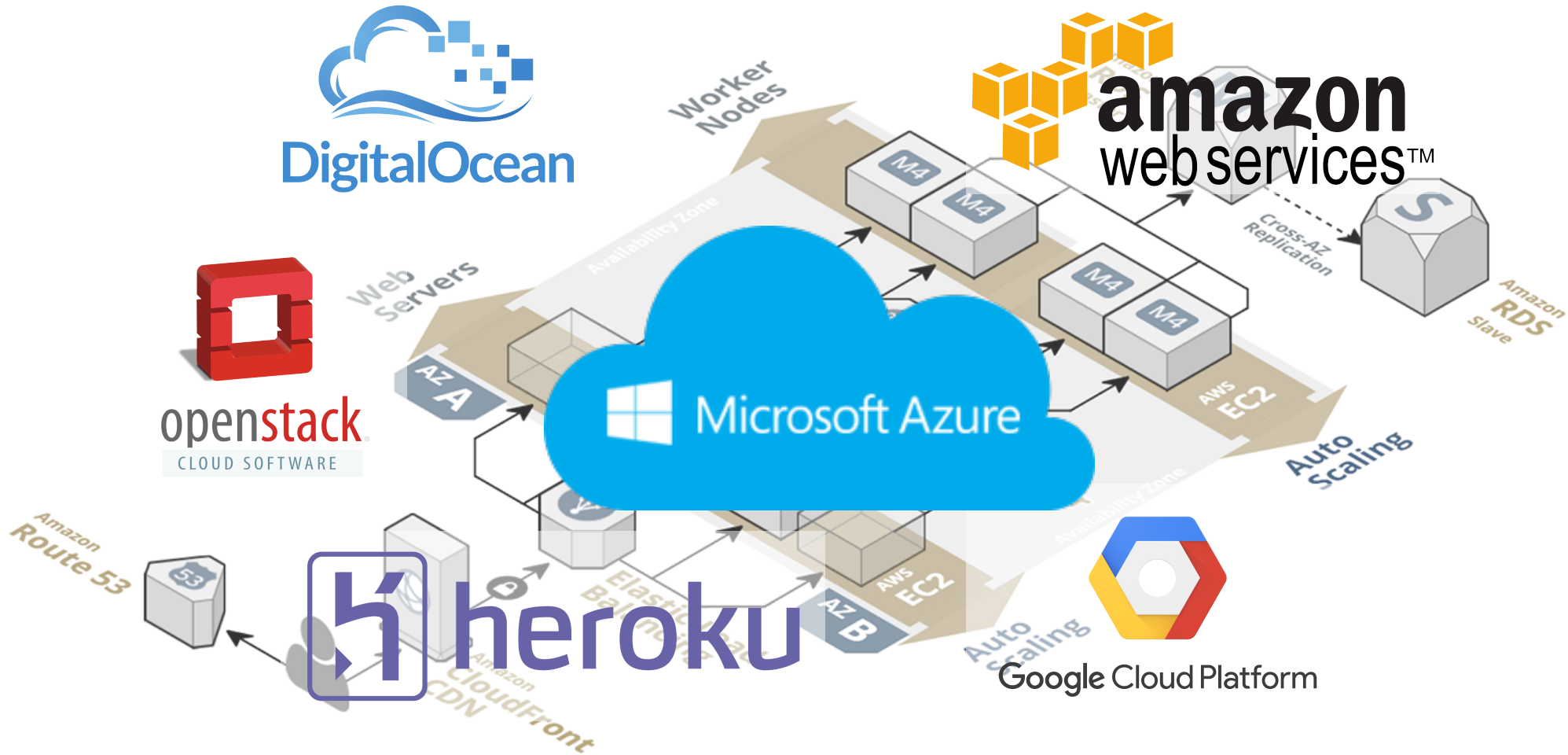
# Concepts

# From Servers ...




by Rodzilla at Wikimedia Commons (CC-BY-SA-3.0)

# ...to Services


DigitalOcean
amazon
web services™
openstack
CLOUD SOFTWARE
Microsoft Azure
heroku
Google Cloud Platform
Worker Nodes
Web Servers
Availability Zone
AZ A
AZ B
M4
AWS EC2
Auto Scaling
Cross-AZ Replication
Amazon RDS
Slave
Amazon Route 53
Amazon CloudFront CDN

# Services also need Configuration Management

- Replace “click paths” with source code in VCS
- Lifecycle awareness, not just a `setup.sh`
- Reproducible environments
- Specification, documentation, policy enforcement

$\Rightarrow$ Infrastructure as Code

# TERRAFORM

Build, Combine, and Launch Infrastructure

# Example: Simple Webservice (part 1)

```
### AWS Setup
provider "aws" {
  profile = "${var.aws_profile}"
  region  = "${var.aws_region}"
}

# Queue
resource "aws_sqs_queue" "importqueue" {
  name = "${var.app_name}-${var.aws_region}-importqueue"
}

# Storage
resource "aws_s3_bucket" "importdisk" {
  bucket = "${var.app_name}-${var.aws_region}-importdisk"
  acl    = "private"
}
```

# Example: Simple Webservice (part 2)

```
### Heroku Setup
provider "heroku" { ... }

# Importer
resource "heroku_app" "importer" {
  name   = "${var.app_name}-${var.aws_region}-import"
  region = "eu"
  config_vars {
    SQS_QUEUE_URL = "${aws_sqs_queue.importqueue.id}"
    S3_BUCKET     = "${aws_s3_bucket.importdisk.id}"
  }
}

resource "heroku_addon" "mongolab" {
  app  = "${heroku_app.importer.name}"
  plan = "mongolab:sandbox"
}
```

# Core Ideas in Terraform

- Simple model of resource entities with attributes
- Stateful lifecycle with CRUD operations
- Declarative configuration
- Dependencies by inference
- Parallel execution

# Core Concepts in Terraform

- Provider: a source of resources
  (usually with an API endpoint & authentication)
- Resource: every thing “that has a set of configurable attributes and a lifecycle (create, read, update, delete)” – implies ID and state
- Data Source: information read from provider
  (e. g. lookup own account ID or AMI-ID)
- Provisioner: initialize a resource with local or remote scripts

# Design Choices in Terraform

- Order: directed acyclic graph of all resources
- Plan: generate an execution plan for review before applying a configuration
- State: execution result is kept in state file (local or remote)
- Lightweight: little provider knowledge, no error handling

# Available services

Providers:

- **AWS**
- **Azure**
- **Google Cloud**
- Heroku
- DNSMadeEasy
- OpenStack
- Docker
- ...

Resources:

- azurerm_lb
- azurerm_subnet
- azurerm_dns_zone
- azure_instance
- aws_iam_user
- heroku_app
- postgresql_schema
- ...

Provisioners:

- chef
- file
- local-exec
- remote-exec

# DSL Syntax

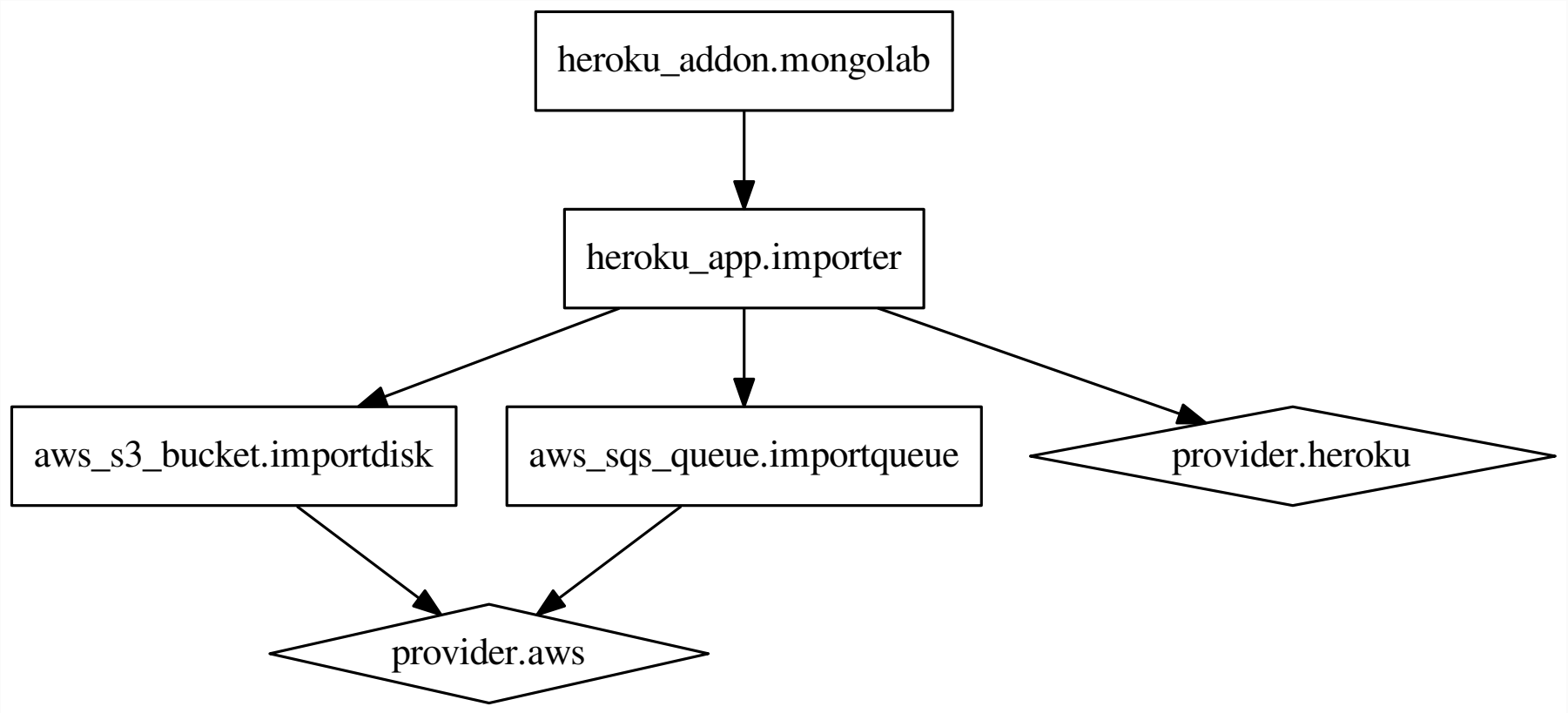
- Hashicorp Configuration Language (HCL),
  think “JSON-like but human-friendly”
- Variables
- Interpolation, e. g.
  `”number ${count.index + 1}”`
- Attribute access with `resource_type.resource_name`
- Few build-in functions, e. g.
  `base64encode(string)`, `format(format, args…)`

# HCL vs. JSON

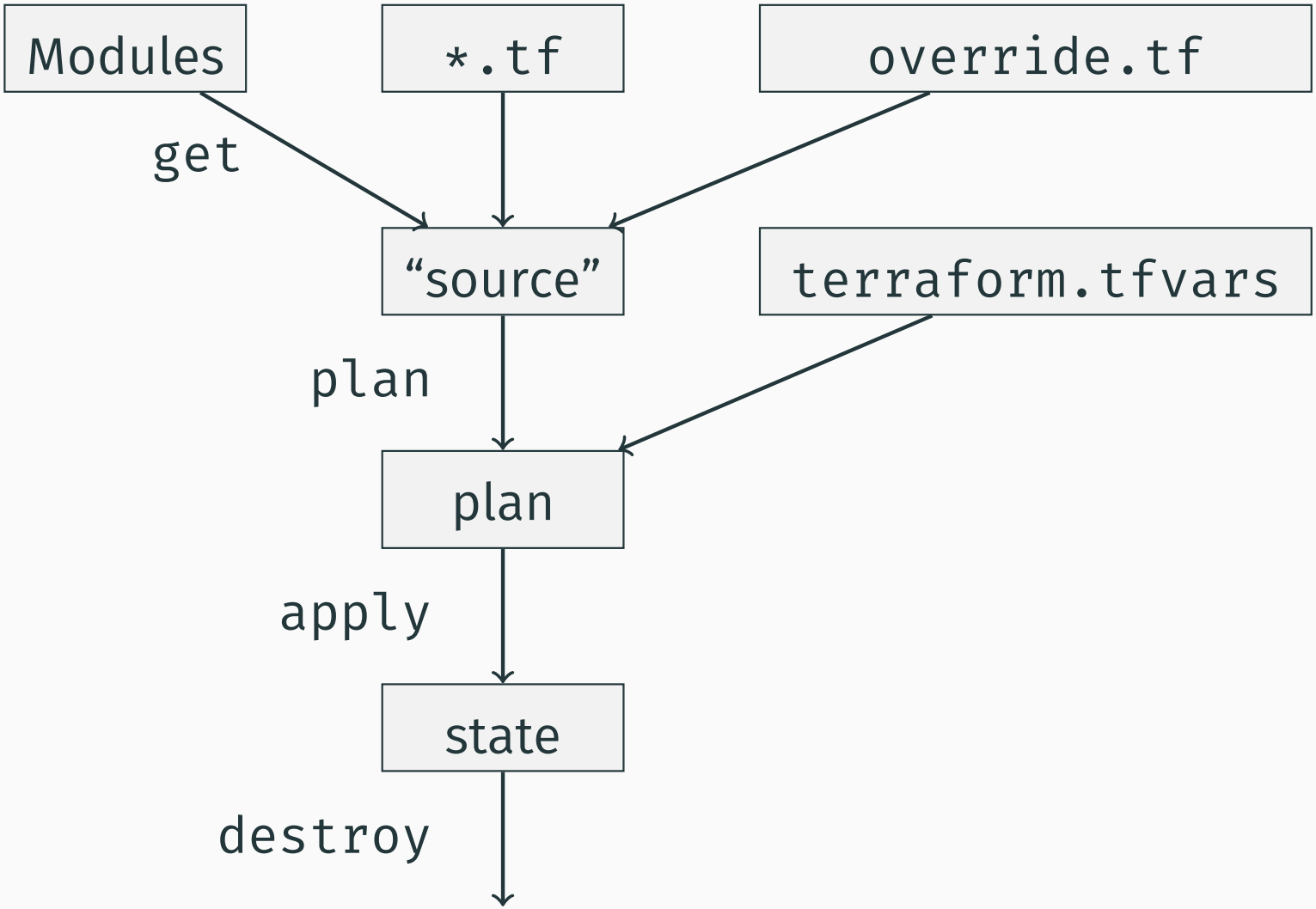
```
# An AMI
variable "ami" {
  description = "custom AMI"
}

/* A multi
   line comment. */
resource "aws_instance" "web" {
  ami = "${var.ami}"
  count = 2
  source_dest_check = false

  connection {
    user = "root"
  }
}
```

```
{
  "variable": {
    "ami": {
      "description": "custom AMI"
    }
  },
  "resource": {
    "aws_instance": {
      "web": {
        "ami": "${var.ami}",
        "count": 2,
        "source_dest_check": false,

        "connection": {
          "user": "root"
        }
      }
    }
  }
}
```

# terraform graph | dot -Tpdf



# Terraform Process



# Example: Add Provisioning

```
# Importer
resource "heroku_app" "importer" {
  name   = "${var.app_name}-${var.aws_region}-import"
  region = "eu"

  config_vars { ... }

  provisioner "local-exec" {
    command = <<EOT
cd ~/projects/go-testserver &&
git remote add heroku ${heroku_app.importer.git_url} &&
git push heroku master
EOT
  }
}
```

# Example: Add Outputs

```
# Storage
resource "aws_s3_bucket" "importdisk" { ... }

# Importer
resource "heroku_app" "importer" { ... }

# Outputs
output "importer_bucket_arn" {
  value = "${aws_s3_bucket.importdisk.arn}"
}

output "importer_url" {
  value = "${heroku_app.importer.web_url}"
}

output "importer_gitrepo" {
  value = "${heroku_app.importer.git_url}"
}
```

# Example: Add Lifecycle Meta-Parameter

```
# Storage
resource "aws_s3_bucket" "importdisk" {
  bucket = "${var.app_name}-${var.aws_region}-importdisk"
  acl    = "private"

  lifecycle {
    prevent_destroy = true
  }
}
```

# Demo

```
$ terraform validate
$ terraform plan -out=my.plan
$ terraform show my.plan
$ terraform apply my.plan

$ terraform output
$ terraform output -json
$ terraform output importer_url
$ curl -s $(terraform output importer_url)

$ terraform graph | dot -Tpdf > graph.pdf && evince graph.pdf

$ terraform plan -destroy
$ terraform destroy
```

# Features

# Modules

“Plain terraform code” lacks structure and reusability

Modules

- are subdirectories with self-contained terraform code
- may be sourced from Git, Mercurial, HTTPS locations
- use variables and outputs to pass data

# Example Module

```
module "vm" {
  source =
    "github.com/chiradeep/terraform-azure-modules/ubuntu_public_vm"

  resource_group_name = "alpha-4ca23e"
  location            = "West US"
  name                = "alpha"

  vhd_uri_base        =
    "https://alphastacctf62da8.blob.core.windows.net/alphacntrn22/"
  user_data_file     = "userdata.sh"
  subnet_id          = "subnet-public-98a1e34"
  ssh_public_keyfile = "key.pub"

  vm_count = 2
}
```

using chiradeep/terraform-azure-modules

# `terraform.tfstate`

- Terraform keeps known state of resources
- Defaults to local state in `terraform.tfstate`
- Optional remote state with different backends (Azure Storage, S3, Consul, Atlas, ...)
  - Useful to sync multiple team members
  - May need additional mutex mechanism (v0.9 added state locking for Local, S3, and Consul)
  - Remote state is a data source

# Example: Using State Import

```
$ terraform import azurerm_storage_account.my_storage_account \
  /subscriptions/e9b2ec19-ab6e-4547-a3ec-5a58e234ce5e/resourceGroups/
  demo-res-group/providers/Microsoft.Storage/storageAccounts/demostorage20170418

azurerm_storage_account.my_storage_account: Importing from ID ...
azurerm_storage_account.my_storage_account: Import complete!
  Imported azurerm_storage_account (ID: ...)
azurerm_storage_account.my_storage_account: Refreshing state... (ID: ...)

Import success! The resources imported are shown above. These are
now in your Terraform state. Import does not currently generate
configuration, so you must do this next. If you do not create configuration
for the above resources, then the next ‘terraform plan‘ will mark
them for destruction.

$ terraform state list
azurerm_storage_account.my_storage_account

$ terraform state show azurerm_storage_account.my_storage_account
id                     = /subscriptions/e9b2ec19...
account_kind           = Storage
account_type           = Standard_LRS
location               = westeurope
name                   = demostorage20170418
...
```

# Example: Use Remote State

```
$ terraform init -backend=true  \
  -backend-config="backend=azure" \
  -backend-config="storage_account_name=demostorage20170418" \
  -backend-config="container_name=demo-storage-container" \
  -backend-config="key=network.terraform.tfstate"
```

# Example: Use Remote State to Chain Projects

```
data "terraform_remote_state" "net" {
  backend = "azure"
  config {
    storage_account_name = "demostorage20170418"
    container_name       = "demo-storage-container"
    key                  = "network.terraform.tfstate"
  }
}

resource "azurerm_public_ip" "rhelvm" {
  name                = "demo-rhelvm-ip"
  location            =
    "${data.terraform_remote_state.net.location}"
  resource_group_name =
    "${data.terraform_remote_state.net.resource_group_name}"
  public_ip_address_allocation = "static"
}
```

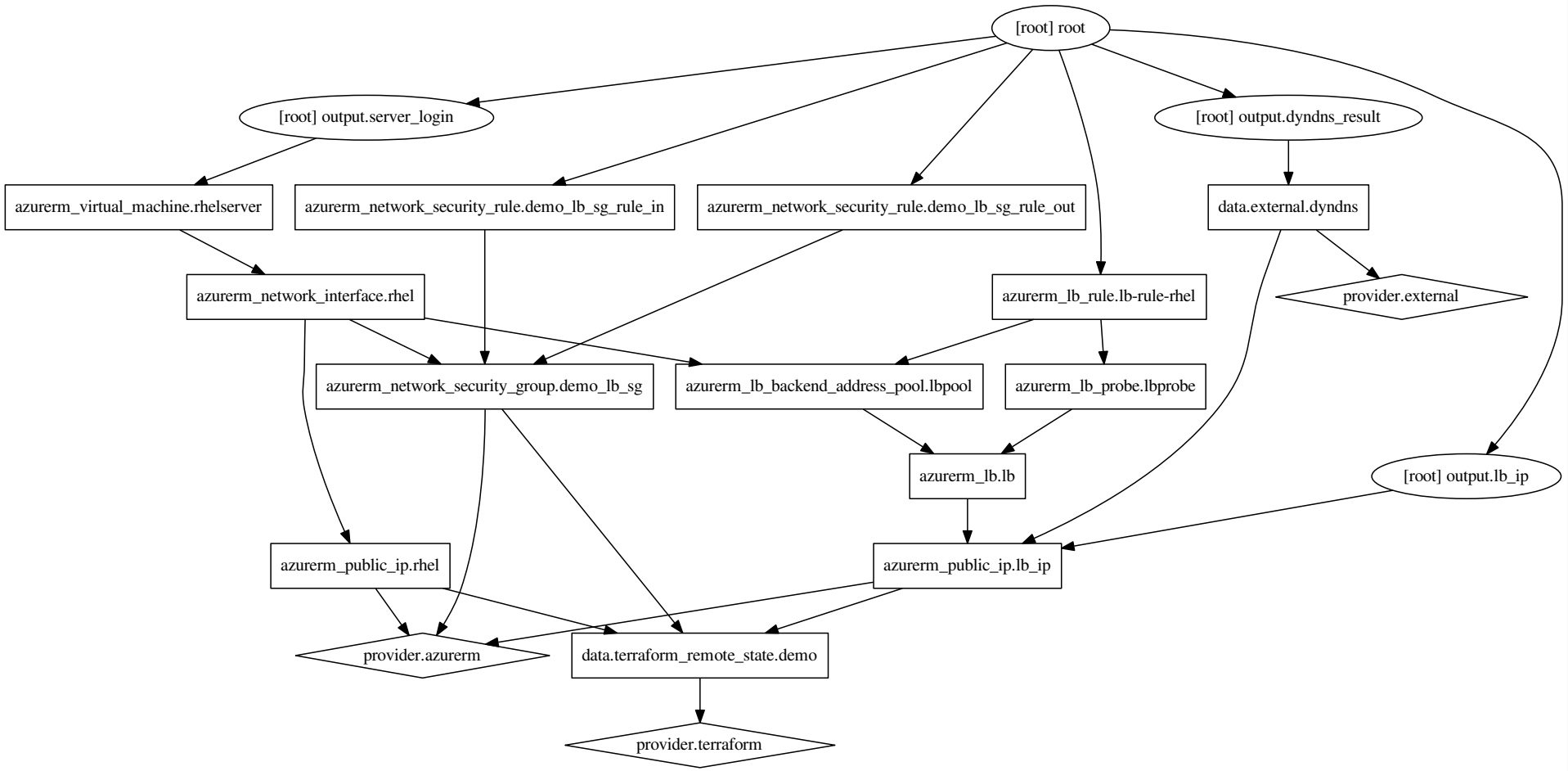
# Example: Using Data Source to lookup data

```
# searches for most recent tagged AMI in own account
data "aws_ami" "webami" {
  most_recent = true
  owners      = ["self"]

  filter {
    name   = "tag:my_key"
    values = ["my_value"]
  }
}

# use AMI
resource "aws_instance" "web" {
  instance_type = "t2.micro"
  ami           = "${data.aws_ami.webami.id}"
}
```

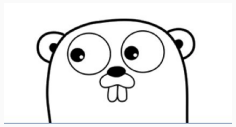
# Example: “External” Data Source

```
data "external" "dyndns" {
  program = ["bash", "${path.module}/variomedia_dyndns.sh"]

  query = {
    hostname  = "azure-demo.martin-schuette.de"
    ipaddress = "${azurerm_public_ip.lb_ip.ip_address}"
  }
}
```

# Azure Demo


[root] root
[root] output.server_login
[root] output.dyndns_result
azurerm_virtual_machine.rhelserver
azurerm_network_security_rule.demo_lb_sg_rule_in
azurerm_network_security_rule.demo_lb_sg_rule_out
data.external.dyndns
azurerm_network_interface.rhel
azurerm_lb_rule.lb-rule-rhel
provider.external
azurerm_network_security_group.demo_lb_sg
azurerm_lb_backend_address_pool.lbpool
azurerm_lb_probe.lbprobe
azurerm_lb.lb
[root] output.lb_ip
azurerm_public_ip.rhel
azurerm_public_ip.lb_ip
provider.azurerm
data.terraform_remote_state.demo
provider.terraform

# How to Write Own Plugins

- Learn you some Golang 
- Use the `schema` helper lib
- Adapt to model of
  Provider (setup steps, authentication) and
  Resources (arguments/attributes and CRUD methods)
- Start reading of simple plugins like
  `builtin/providers/mysql`

# Usage

# Issues

Under active development,
current version 0.9.3 (April 12), expecting 1.0 soon

- Modules are very simple
- Lacking syntactic sugar
  (e. g. aggregations, common repetitions)
- Big improvements in state management
- Large variation in provider support

# General Problemes for all Tools

- Testing is inherently difficult
- Provider coverage largely depends on community
- Resource model mismatches, e. g. with Heroku apps
- Ignorant of API rate limits, account ressource limits, etc.

# Recent Features in 0.7–0.9

- Lists and Maps may be passed to modules
- State Import
- Data Sources
- State Environments

# Comparable Tools

Configuration Management Tools:

- SaltStack Salt Cloud
- Ansible modules
- Puppet modules

Vendor Tools:

- Azure Resource Manager Templates
- AWS CloudFormation
- OpenStack Heat

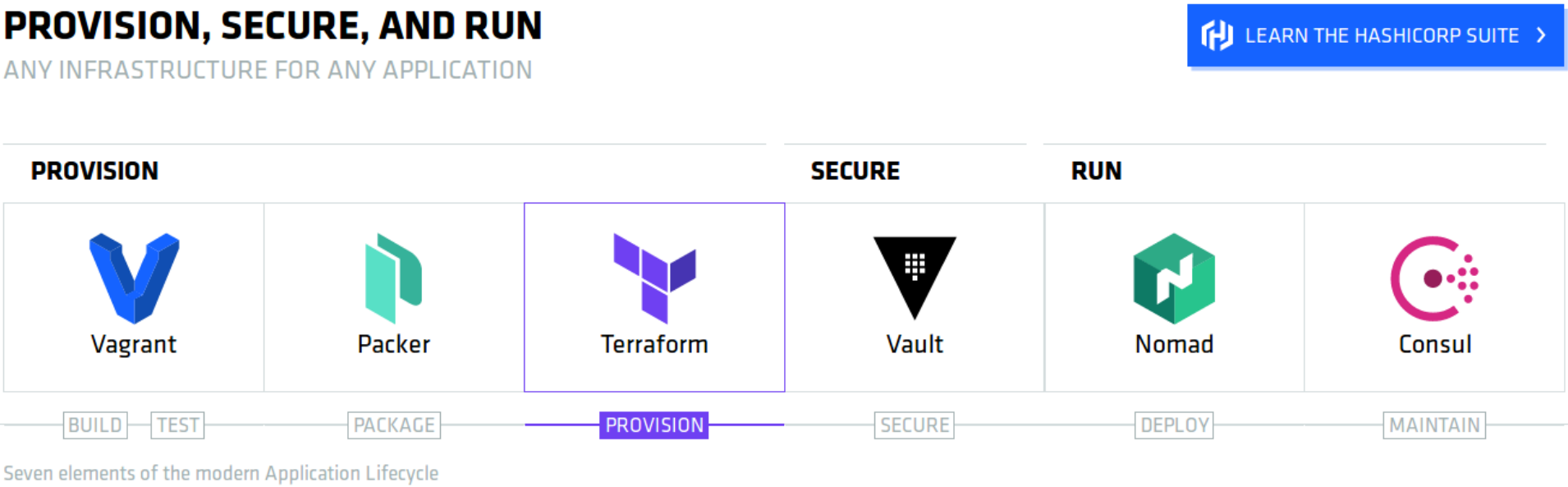
# Workflow

- Avoid user credentials in Terraform code,
  use e. g. profiles and *assume-role* wrapper scripts
- At least use separate user credentials,
  know how to revoke them
- To hold credentials in VCS use PGP encryption,
  e. g. with Blackbox

# Workflow (contd.)

- Use a VCS, i. e. git
- Always add some "`${var.shortname}`" to namespace
- Use remote state and consider access locking,
  e. g. with a single build server
- Take a look at Hashicorp Atlas and its workflow

# Hashicorp Toolset



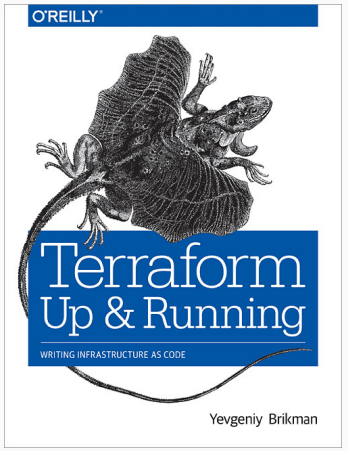

Seven elements of the modern Application Lifecycle

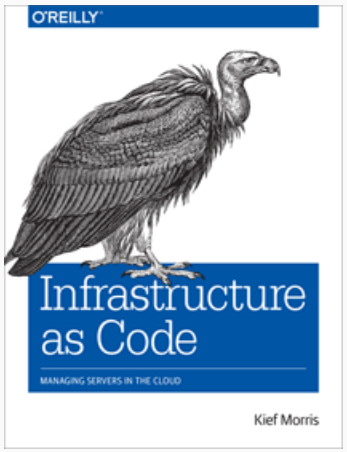
# Links and Resources

- Terraform.io and hashicorp/terraform
- terraform-community-modules
- Creating Azure Resources with Terraform
- A Comprehensive Guide to Terraform
- Terraform, VPC, and why you want a tfstate file per env
- Terraform: Beyond the Basics with AWS

# Books

*Hopefully, deployments will become routine and boring–and in the world of operations, boring is a very good thing.*
— *Terraform: Up & Running* by Yevgeniy Brikman




*Defining system infrastructure as code and building it with tools doesn't make the quality any better. At worst, it can complicate things.*
— *Infrastructure as Code* by Kief Morris

# The End


Martin Schütte
@m_schuett
info@martin-schuette.de

slideshare.net/mschuett/