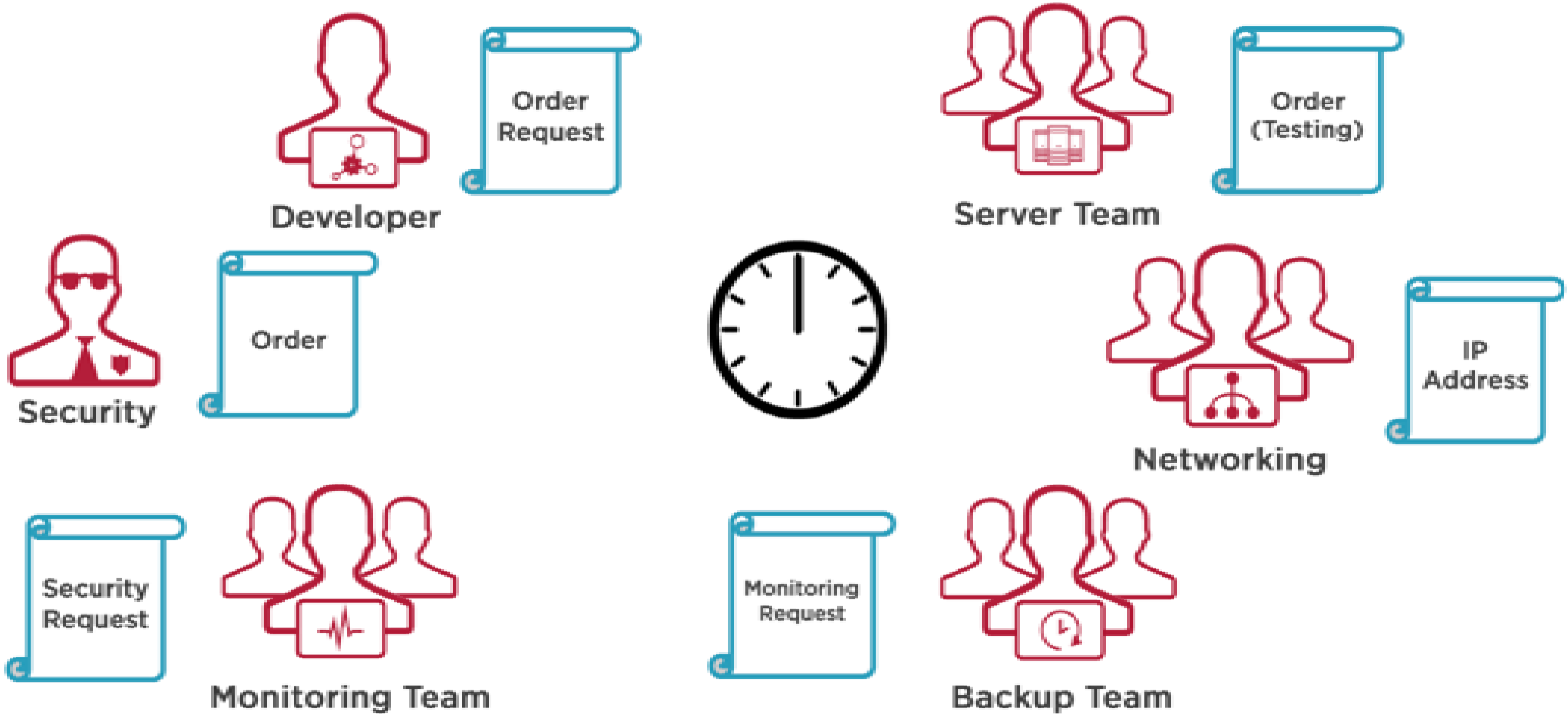
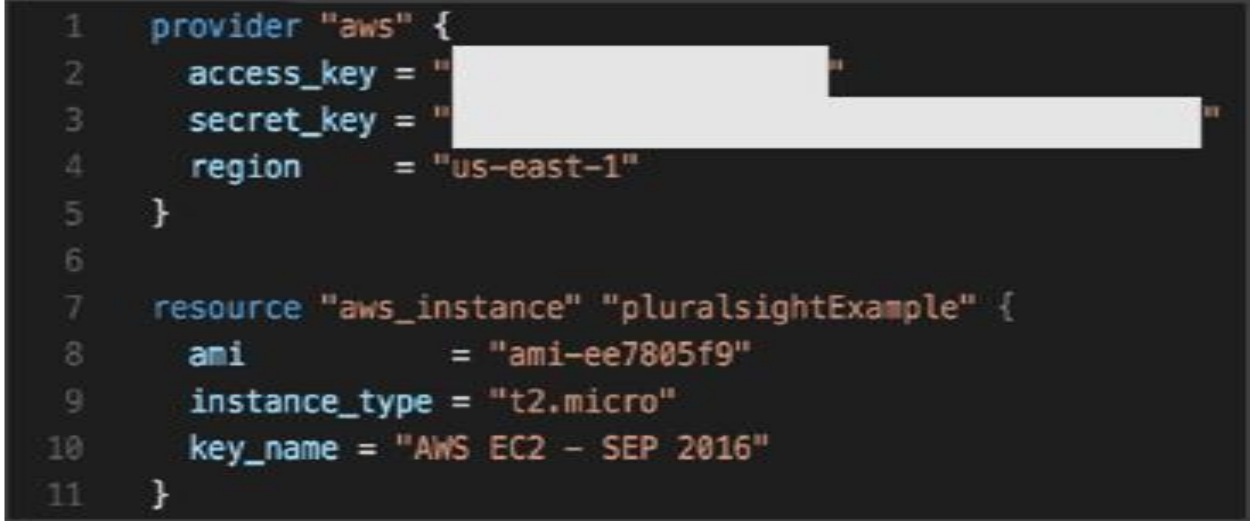
TERRAFORM

THE OLD WAY TO GET AN IT SERVICE



The Infrastructure as Code (IaC) Way



```
provider "aws" {
  access_key = "[redacted]"
  secret_key = "[redacted]"
  region     = "us-east-1"
}

resource "aws_instance" "pluralsightExample" {
  ami           = "ami-ee7805f9"
  instance_type = "t2.micro"
  key_name = "AWS EC2 - SEP 2016"
}
```

Benefits of IaC

Improved Quality from IT to Business

Speed

Innovation

Terraform and Configuration Management

CHEF SALTSTACK

- OS Configuration
- Application Installation
- Declarative
- Limited Infrastructure Automation

TERRAFORM

- Infrastructure Automation
- VM and Cloud Provisioning
- Declarative like Configuration Management Tools
- Limited OS Configuration Management

Declarative vs Procedural

Procedural	Declarative
Connect to VMware vCenter	Give me a Virtual Machine with the following configuration:
Create VM	CPUs: 2
Install Windows Operating System	Memory: 2GB
Configure NIC Settings	OS: Windows Server 2012
Install Software Package A	1 NIC with IP 10.0.0.101/24
Install Software Package B	Puppet Role: SQL Server

Automating Infrastructure Deployment

Provisioning Resources

Planning Updates

Using Source Control

Reusing Templates

Terraform Components

Terraform Executable

Terraform File

```
variable "aws_access_key" {}

variable "aws_secret_key" {}



provider "aws" {

  access_key = "access_key"
  secret_key = "secret_key"
  region = "us-east-1"

}
```

◂ Variables

◂ Provider

```
resource "aws_instance" "ex"{

  ami = "ami-c58c1dd3"
  instance_type = "t2.micro"

}

output "aws_public_ip" {

  value =
  "${aws_instance.ex.public_dns}"

}
```

◂ Resource

◂ Output

Some of the resources deployed in AWS may cost **money**. You've been warned.

Terraform Constructs

Terraform Constructs

Providers

Resources

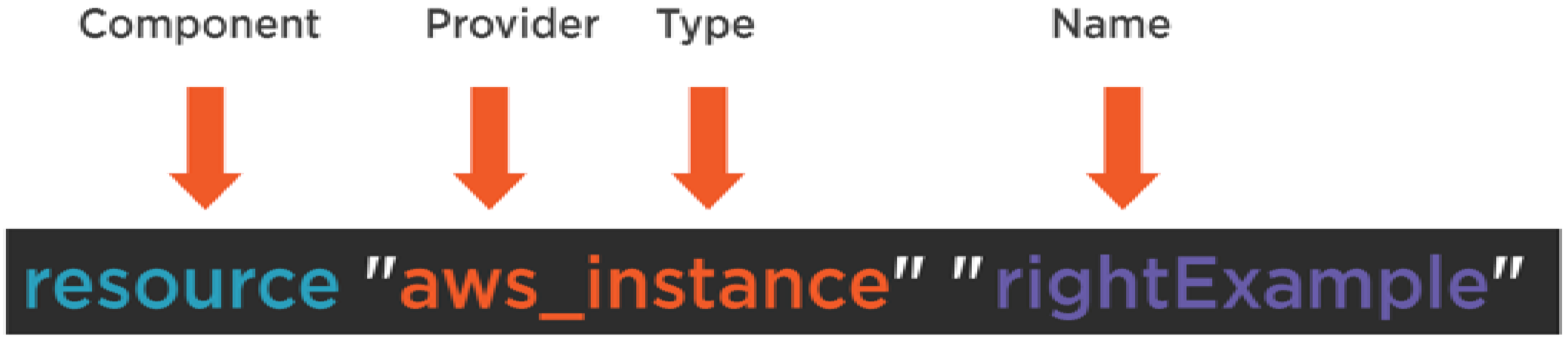
Provisioners

Providers

amazon
web services

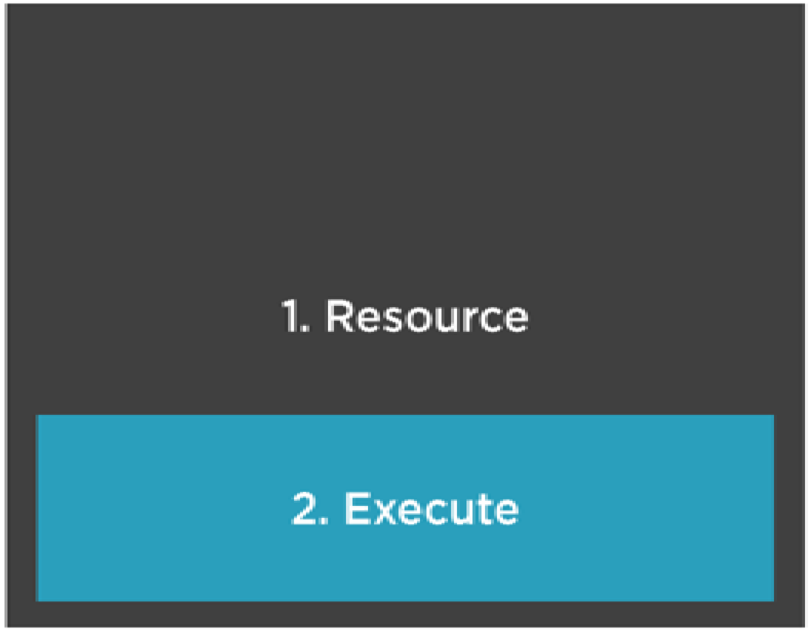
“A provider is responsible for understanding API interactions and exposing resources.”

Resources



```
resource "aws_instance" "rightExample"
```

Provisioners



“When a resource is initially created, provisioners can be executed to initialize that resource”

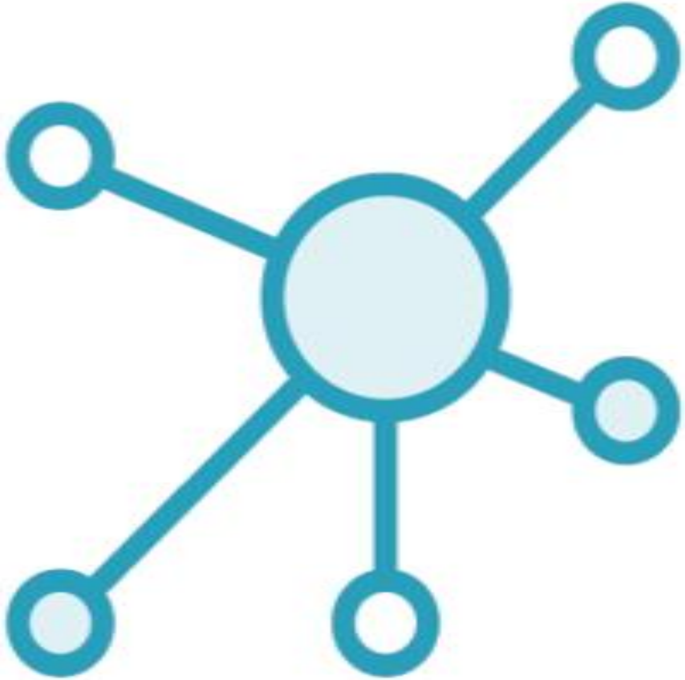
Terraform Execution

Plan	terraform plan
Execute	terraform apply
Destroy	terraform destroy

Terraform State

- JSON format (Do not touch!)
- Resources mappings and metadata
- Locking
- Local / remote
- Environments

Terraform Planning



Inspect state

Dependency graph

Additions and deletions

Walk the line

Terraform Syntax

```
#Create a variable

variable var_name {

 key = value #type, default, description

}

#Use a variable

${var.name} #get string

${var.map[“key”]} #get map element

${var.list[idx]} #get list element
```

Terraform Syntax

```
#Create provider

provider provider_name {

 key = value #depends on resource, use alias as needed

}

#Create data object

data data_type data_name {}

#Use data object

${data_type.data_name.attribute(args)}
```

Terraform Syntax

```
#Create resource

resource resource_type resource_name {

 key = value #depends on resource

}

#Reference resource

${resource_type.resource_name.attribute(args)}
```

Terraform Modules

Code reuse

Remote or local source

Terraform evaluation

Mini-Terraform configuration

Multiple instances (no count)

Module Components

Variables

Resources

Outputs

Terraform Module

```
variable “name” {}

resource “aws_s3_bucket” “bucket”{
  name = “${var.name}”
  […]
}

output “bucket_id” {
  value = “${aws_s3_bucket.bucket.id}”
}
```

Terraform Module

```
#Create module bucket
module “bucket” {
  name = “MahBucket”
  source = “.\\Modules\\s3”
}

#Use MahBucket
resource “aws_s3_bucket_object” {
  bucket = “${module.bucket.bucket_id}”
  key = “/walrus/bucket.txt”
  source = “./mahbucket.txt”
}
```

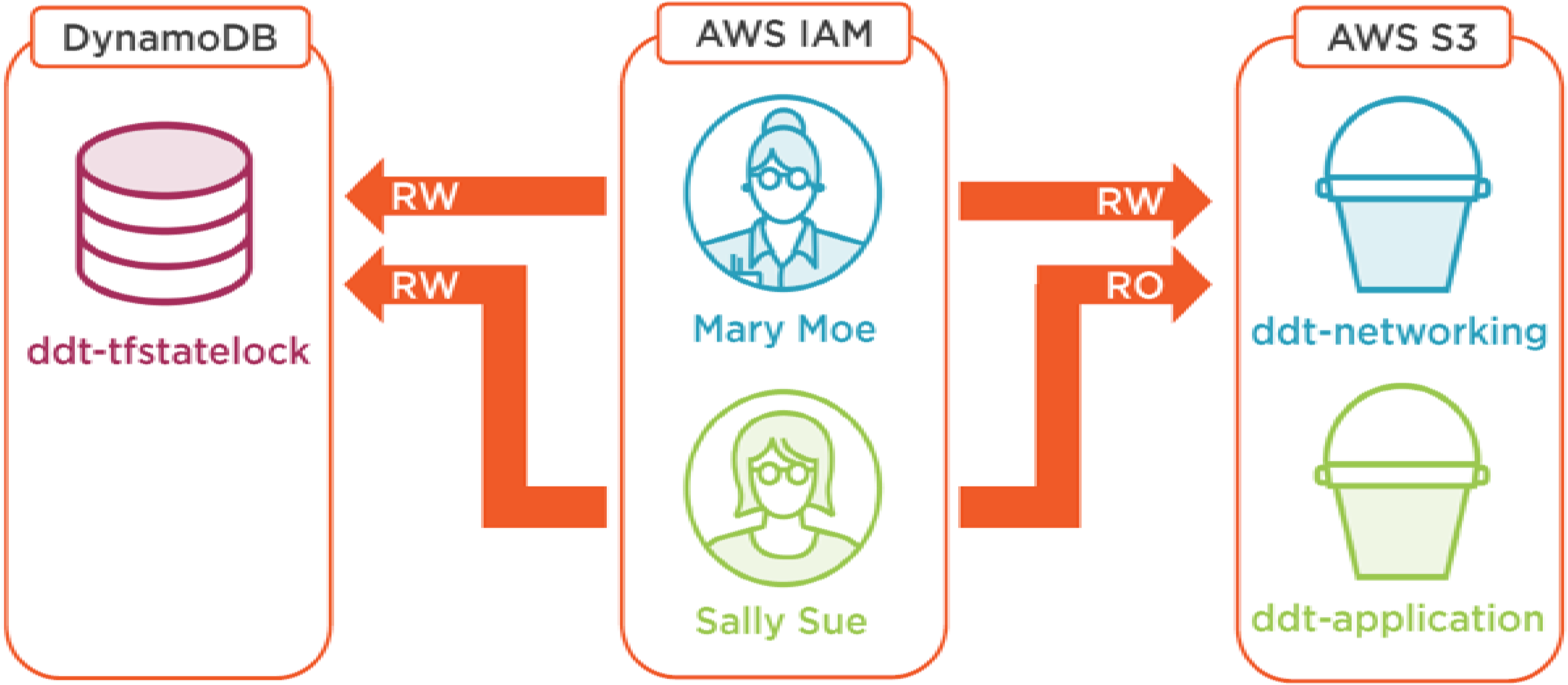
Working as part of a larger team

Configuring infrastructure for another team

Remote state enables collaboration

Need to restrict access for other teams

Remote State Setup



BACKEND

Remote state is stored in a backend

Backends must be initialized using init

Partial configurations recommended

Backends do not support interpolation

```
# Basic Backend

terraform {
      backend "type" {
            # backend configurations
            # partial configurations allowed
            # no interpolations
      }
}

# Backend Types S3, Consul, AzureRM
```

Configuring a Backend

WORKSPACES

Workspaces replace the environment command

Separate state file per workspace

Use a single configuration for multiple deployments

Supported by select backends

Configuring a Workspace

# Workspace Commands

`show` – show current workspace
`list` – list all workspaces
`select` – select which workspace to use
`new` – create a new workspace
`delete` – remove a new workspace

# Examples

```
terraform workspace new development
terraform workspace select development
```

# Using in a Configuration

```
${terraform.workspace}
```