Introduction to Terraform and Terraform Modules

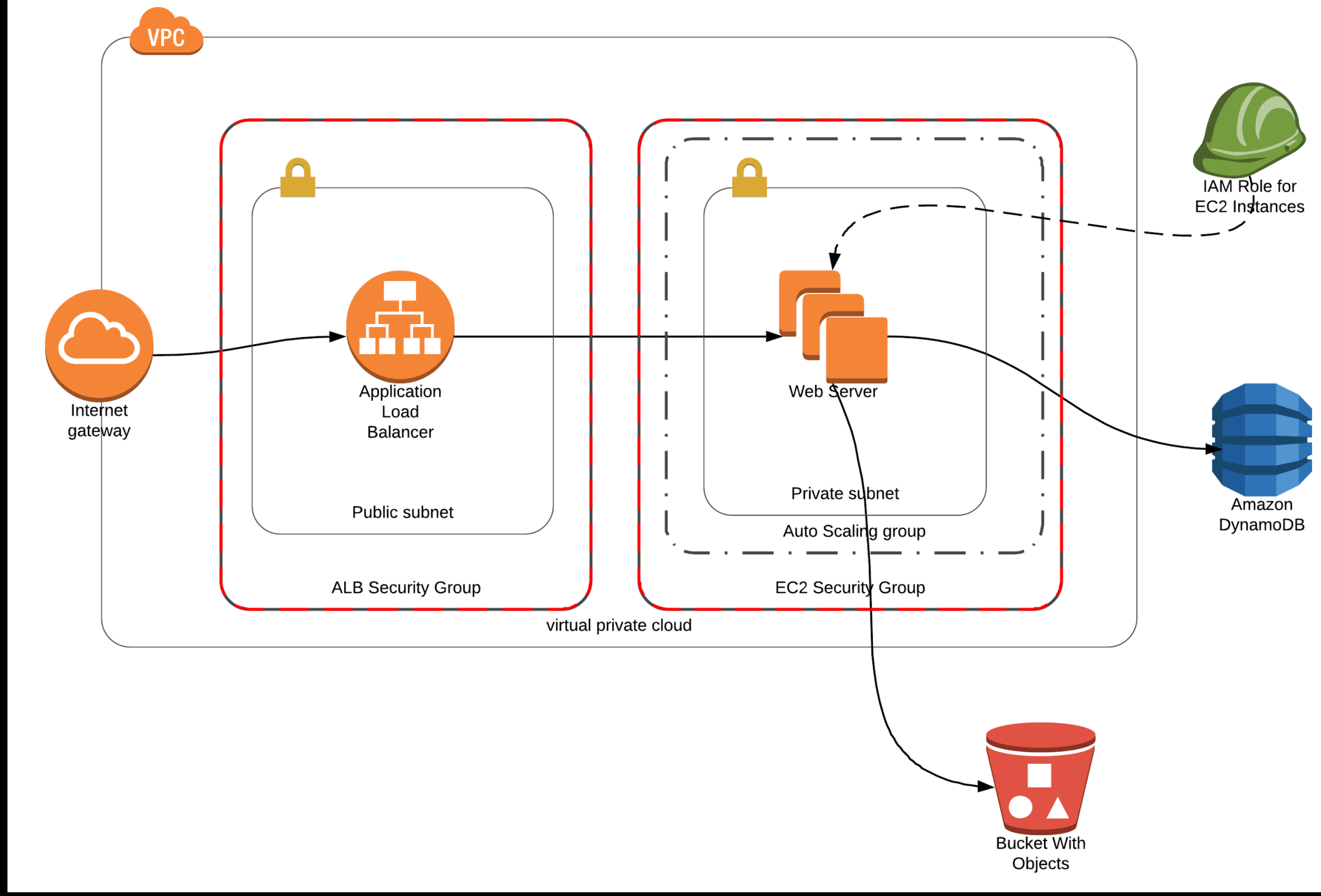
Alberto Castro

What is Terraform?

- “Terraform is a tool for building, changing, and *versioning* infrastructure safely and efficiently.”

- Building, changing, and *versioning* – *Infrastructure as Code that can be managed with Git.*

- Safely and efficiently – Terraform maintains state of infrastructure and applies the changes in a (mostly) non-conflicting manner.



Goal: Build this infrastructure with Terraform

Terraform Configuration for AWS

- Install Terraform downloading it from https://www.terraform.io/downloads.html

- Install `awscli` in your computer.

- To configure terraform create a user in AWS using IAM and name it `terraform` and programatic access (no console login)

- Assign full administration permissions to the user.

- Download the Access Keys and upload them and initialize the awscli by running:
  `aws configure`

Terraform Basic Use

- Create a terraform file in a directory in your computer.

- If it is the first time you are running terraform for this script you should run `terraform init` so that terraform can download the corresponding providers (aws in our case)

- Run `terraform plan` and check the changes that will be applied.

- Run `terraform apply` and after accepting the changes terraform will provision your infrastructure.

- If you desire to destroy the infrastructure run `terraform destroy` this will remove all the resources created before.

Other Useful Commands

- `terraform fmt` Formats the terraform files.
- `terraform state subcommand` Allow you to display and manage terraform state (`list`, `rm`, `show`, ...)
- Run `terraform validate` Validates the Terraform files.
- `terraform taint` Manually mark a resource for recreation
- `terraform untaint` Manually mark a resource for recreation

Example 1- Basic

Homework

- Create an AWS account, create a terraform user and generate a key pair for it.
  URL NEEDED

- Initialize terraform and deploy example1 draft1, draft2 and final. Examine the result in the console. Remember to run terraform destroy when you finish working to avoid extra charges.

- Using example1/draft1 deploy one more ec2 instance:

  1. By copying the definition and renaming it.

  2. By adding the following line to the existing definition of the instance resource:

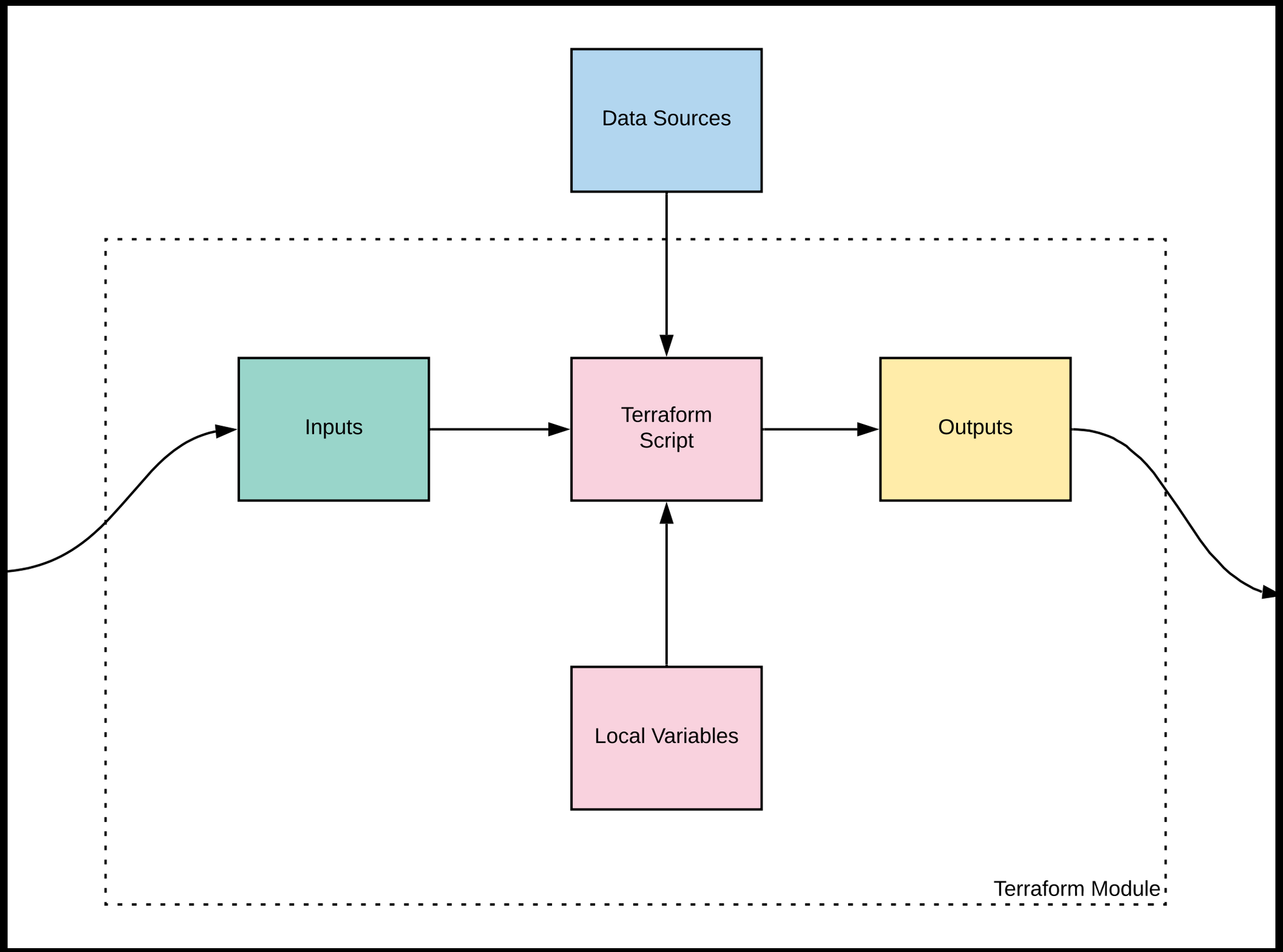
     count = 2

- Using the `aws_security_group_rule` resource rewrite the ingress rules for the aws security group **ec2_security_group** defined in main (remember to remove the ingress rules in the `aws_security_group_rule` definition).

- Re-run terraform with this configuration. The result should be:

  - If you already deployed the terraform resources terraform plan should tell you that no changes are necessary.

  - If you are deploying them anew you should get the same result as the final example.



Terraform Module Structure

Terraform Architecture

- Terraform is written in Go and it is a tool developed and maintained by Hashicorp (https://www.hashicorp.com/).

- A core engine manages the language features, so that the same language (HCL) is used for all infrastructure types.

- Providers create the resources in the different cloud services. This way depending on the provider loaded different infrastructure can be managed.

- The AWS Terraform provider leverages the Go SDK to create resources.

Language

- Terraform's language is *declarative.* A terraform script specifies what is the final state of the infrastructure after changes are applied.

- Terraform manages all dependencies (mostly), creating resources in the correct order.

- The latest version of Terraform (0.12) uses HCL 2.0 which is a major improvement over the previous version making the code more readable and easier to write.

- Every script is its own "module" which in Terraform means a script that accepts inputs and outputs (like a function) and that can be linked to other modules.

- Iteration can be achieved using a counter and (more recently) for loops.

- There is a loose type structure with:

  - Primitive Types: string, number, bool

  - Collection Types: list, map, set (all of same type elements)

  - Structural Types: tuple (a lists of multiple types) and object (a map of multiple types)

- All these types have literal expressions to declare explicit values.

- Primitive type examples: “this is a string”; 12345 (my favorite password!); true

- List: [“a”, “b”, “z”]

- Map:

```
{
    us-east-1  = "ami-0a01a5636f3c4f21c"
    us-east-2  = "ami-0a02eadc6d8770f83"
    us-west-1  = "ami-0bbeea654a35ef611"
    us-west-2  = "ami-0a1af68029fa293b6"
}
```

- Object:

```
{
    anumber = 12
    alist   = [“2”, “w”, “foo”]
    atuple  = [2, false, “baz”]
}
```

Example 2 - Modules

Terraform State

- Terraform maintains a file (terraform.tfstate) that contains the state of the infrastructure after the last applied change.

- Initially Terraform assumes that the resources that it is creating do not exist, and as new resources are created the terraform.tfstate file is updated.

- It is possible to import state, although it is cumbersome.

- It is *highly discouraged* to make changes without Terraform once it has been created with it. Doing so means having to import the new state or give up on using Terraform altogether.

- Terraform State stores information about the current resources, the parameters, and dependencies.
- API calls are made to the cloud provider to check the infrastructure state and compare it with the changes.
- The state *can* be used by terraform to plan if desired, this is useful for very large infrastructure deployments.
- The state can be saved and shared using for example:
  - Terraform Cloud
  - S3 bucket + DynamoDB table for locking

Example 3 - Modules With Remote State

Storing and Sharing Modules: 1. The Registry

- Terraform registry is a repository of modules freely available for use directly from terraform.

- All modules are versioned and fully documented.

- Modules are available for Terraform v0.11 and v0.12. Examples are for Terraform v0.12 and HCL 2.0 in this presentation.

- Some modules are “verified” which means that the modules have been written and reviewed by Hashicorp engineers.

Storing and Sharing Modules: 2. Git Repositories

- Git repositories and in particular GitHub can be used to store and share repositories. (https://www.terraform.io/docs/modules/sources.html)

- Modules defined in a git repository are loaded with:

```
source = "git::https://example.com/vpc.git"
```

- Branches and tags can be specified allowing for versioning and source control. Here v1.2.4 is the name of a branch or tag:

```
source = “git::https://example.com/vpc.git?ref=v1.2.4"
```

Example 4

Using the Registry

Terraform in AWS

- Terraform is recommended by AWS on par with their own tool CloudFormation to manage and deploy infrastructure in AWS.

- Terraform's AWS provisioner receives frequent updates and new features are deployed shortly after, or sometimes as the services are added to AWS.

- It is written with the AWS Go SDK so as features are added there they are added to Terraform.

- Terraform is used by AWS engineers.

How to learn Terraform

- Learn first some Cloud Infrastructure principles (Virtual Servers, IAM, Databases, VPCs, etc).

- Open a free account in a major cloud provider and play with the console for a while.

- Recreate the infrastructure created before using Terraform.

- After that check out code in the official Terraform registry and study it.

- Find a cool project and deploy your infrastructure with Terraform, do not worry, you can start small and add more later.

Resources

- Github Repository with the examples and slides https://github.com/albertocastro63/terraform-nerd

- Terraform Web Site https://www.terraform.io/

- Terraform Registry https://registry.terraform.io/

- *Terraform Up And Running 2nd Edition* by **Yevgeniy Brikman** O'Reilly Media, September 2019 http://shop.oreilly.com/product/063692022501O.do