- Testable modules
- Versioned modules
- Beyond Terraform modules

Small Modules

Developers who are new to Terraform, and IaC in general, often define all of their infrastructure for all of their environments (dev, stage, prod, etc.) in a single file or single module. As discussed in "State File Isolation", this is a bad idea. In fact, I'll go even further and make the following claim: large modules—modules that contain more than a few hundred lines of code or that deploy more than a few closely related pieces of infrastructure—should be considered harmful.

Here are just a few of the downsides of large modules:

Large modules are slow

> If all of your infrastructure is defined in one Terraform module, running any command will take a long time. I've seen modules grow so large that `terraform plan` takes 20 minutes to run!

Large modules are insecure

> If all your infrastructure is managed in a single large module, to change anything, you need permissions to access everything. This means that almost every user must be an admin, which goes against the *principle of least privilege*.

Large modules are risky

> If all your eggs are in one basket, a mistake anywhere could break everything. You might be making a minor change to a frontend app in staging, but due to a typo or running the wrong command, you delete the production database.

Large modules are difficult to understand

The more code you have in one place, the more difficult it is for any one person to understand it all. And when you don't understand the infrastructure you're dealing with, you end up making costly mistakes.

Large modules are difficult to review

Reviewing a module that consists of several dozen lines of code is easy; reviewing a module that consists of several thousand lines of code is nearly impossible. Moreover, `terraform plan` not only takes longer to run, but if the output of the `plan` command is several thousand lines, no one will bother to read it. And that means no one will notice that one little red line that means your database is being deleted.

Large modules are difficult to test

Testing infrastructure code is hard; testing a large amount of infrastructure code is nearly impossible. I'll come back to this point in Chapter 9.

In short, you should build your code out of small modules that each do one thing. This is not a new or controversial insight. You've probably heard it many times before, albeit in slightly different contexts, such as this version from *Clean Code*:[4]

> *The first rule of functions is that they should be small. The second rule of functions is that they should be smaller than that.*

Imagine you were using a general-purpose programming language such as Java or Python or Ruby, and you came across a single function that was *20,000 lines* long—you would immediately know this is a code smell. The better approach is to refactor this code into a number of small, standalone functions that each do one thing. You should use the same strategy with Terraform.

Imagine that you came across the architecture shown in Figure 8-1.

Kinesis
(Stream processing)
VPC Peering
Lambda
(ETL)
SQS
(Queue)
S3
(static content)
CloudFront
(CDN)
Redis
RDS Master
RDS Slave
Internal Load
Balancer
Public Load
Balancer
Backend apps
(Docker cluster)
Frontend apps
(Docker cluster)
Availability
Zone 1
Availability
Zone 2
Availability
Zone 3
Private data subnets
Private app subnets
Public subnets
VPC
Route 53
(DNS)
User

Figure 8-1. A relatively complicated AWS architecture.

If this architecture was defined in a single Terraform module that was 20,000 lines long, you should immediately think of it as a code smell. The better approach is to refactor this module into a number of small, standalone modules that each do one thing, as shown in Figure 8-2.

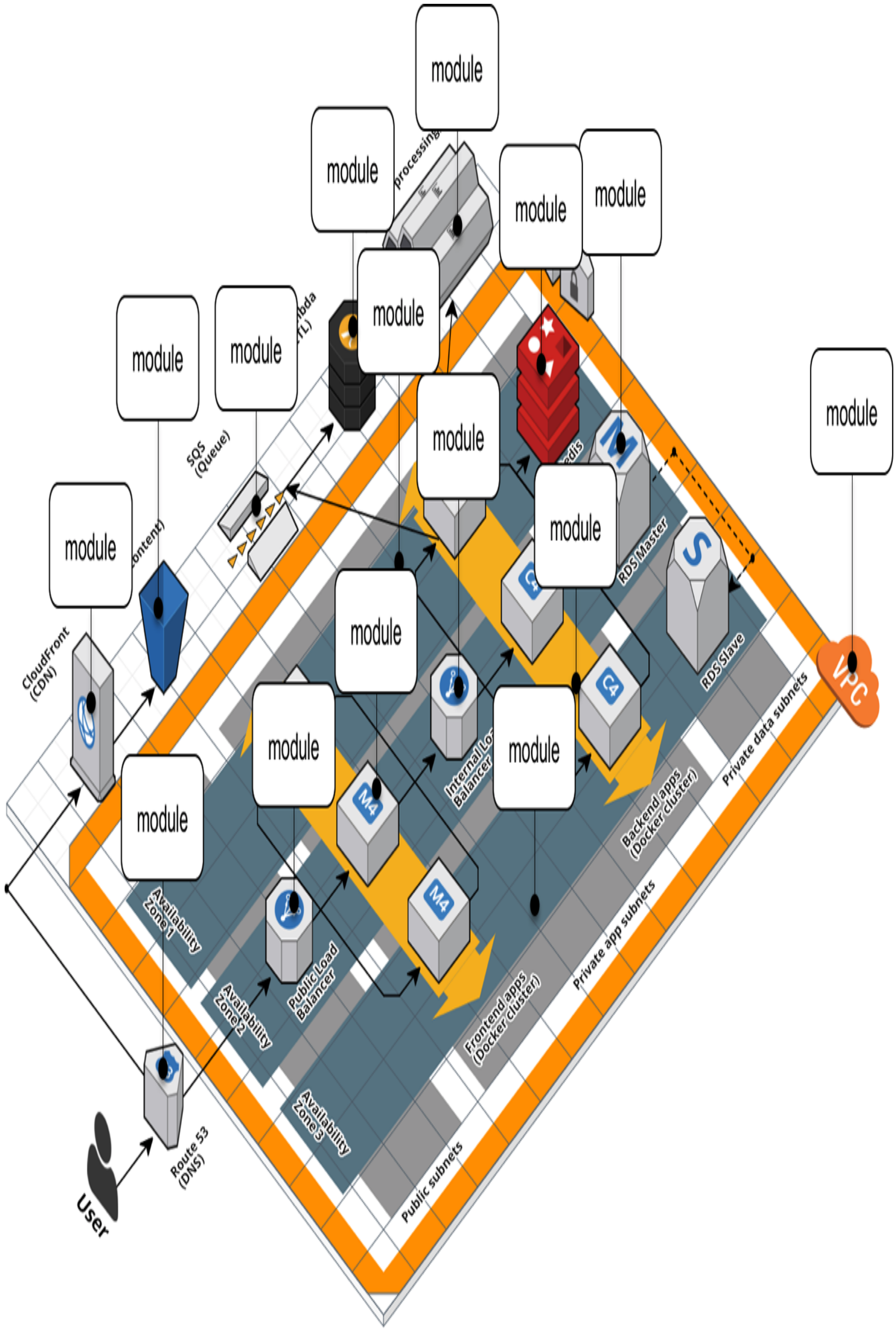
module
module
module
module
module
module
module
module
module
module
module
module
module
module
module
CloudFront (CDN)
SQS (Queue)
processing
RDS Master
RDS Slave
VPC
Internal Lo Balancer
Public Load Balancer
Availability Zone 1
Availability Zone 2
Availability Zone 3
Frontend apps (Docker cluster)
Backend apps (Docker cluster)
Public subnets
Private app subnets
Private data subnets
M4
M4
C4
C4
M
S
Route 53 (DNS)
User

Figure 8-2. A relatively complicated AWS architecture refactored into many small modules.

For example, consider the `webserver-cluster` module, which you last worked on in Chapter 5. This module has become fairly large, as it is handling three somewhat unrelated tasks:

Auto Scaling Group (ASG)

The `webserver-cluster` module deploys an ASG that can do a zero-downtime, rolling deployment.

Application Load Balancer (ALB)

The `webserver-cluster` deploys an ALB.

Hello, World app

The `webserver-cluster` module also deploys a simple "Hello, World" app.

Let's refactor the code accordingly into three smaller modules:

`modules/cluster/asg-rolling-deploy`

A generic, reusable, standalone module for deploying an ASG that can do a zero-downtime, rolling deployment.

`modules/networking/alb`

A generic, reusable, standalone module for deploying an ALB.

`modules/services/hello-world-app`

A module specifically for deploying the "Hello, World" app, which uses the `asg-rolling-deploy` and `alb` modules under the hood.

Before getting started, make sure to run `terraform destroy` on any `webserver-cluster` deployments you have from previous chapters. After you do that, you can start putting together the `asg-rolling-`

`deploy` and `alb` modules. Create a new folder at *modules/cluster/asg-rolling-deploy*, and move the following resources from *module/services/webserver-cluster/main.tf* to *modules/cluster/asg-rolling-deploy/main.tf*:

- `aws_launch_configuration`
- `aws_autoscaling_group`
- `aws_autoscaling_schedule` (both of them)
- `aws_security_group` (for the Instances but not for the ALB)
- `aws_security_group_rule` (just the one rule for the Instances but not those for the ALB)
- `aws_cloudwatch_metric_alarm` (both of them)

Next, move the following variables from *module/services/webserver-cluster/variables.tf* to *modules/cluster/asg-rolling-deploy/variables.tf*:

- `cluster_name`
- `ami`
- `instance_type`
- `min_size`
- `max_size`
- `enable_autoscaling`
- `custom_tags`
- `server_port`

Let's now move on to the ALB module. Create a new folder at *modules/networking/alb*, and move the following resources from *module/services/webserver-cluster/main.tf* to *modules/networking/alb/main.tf*: