When using interpolation to pass credentials to the Kubernetes provider from other resources, these resources SHOULD NOT be created in the same Terraform module where Kubernetes provider resources are also used. This will lead to intermittent and unpredictable errors which are hard to debug and diagnose. The root issue lies with the order in which Terraform itself evaluates the provider blocks vs. actual resources.

The example code in this book is able to work around these issues by depending on the `aws_eks_cluster_auth` data source, but that's a bit of a hack. Therefore, in production code, I always recommend deploying the EKS cluster in one module and deploying Kubernetes apps in separate modules, after the cluster has been deployed.

Conclusion

At this point, you hopefully understand how to work with multiple providers in Terraform code, and you can answer the three questions from the beginning of this chapter:

What if you need to deploy to multiple AWS regions?

Use multiple `provider` blocks, each configured with a different `region` and `alias` parameter.

What if you need to deploy to multiple AWS accounts?

Use multiple `provider` blocks, each configured with a different `assume_role` block and an `alias` parameter.

What if you need to deploy to other clouds, such as Azure or GCP or Kubernetes?

Use multiple `provider` blocks, each configured for its respective cloud.

However, you've also seen that using multiple providers in one module is typically an antipattern. So the real answer to these questions, especially in real-world, production use cases, is to use each provider in a separate module to keep different regions, accounts, and clouds isolated from one another, and to limit your blast radius.

Let's now move on to Chapter 8, where I'll go over several other patterns for how to build Terraform modules for real-world, production use cases—the kind of modules you could bet your company on.

---

1 In fact, you could even skip the `provider` block and just add any resource or data source from an official provider and Terraform will figure out which provider to use based on the prefix: for example, if you add the `aws_instance` resource, Terraform will know to use the AWS Provider based on the `aws_` prefix.

2 See "Multi-Cloud is the Worst Practice".

3 For a comparison of the different types of EKS worker nodes, see the Gruntwork blog.

4 Alternatively, you can use off-the-shelf production-grade Kubernetes modules, such as the ones in the Gruntwork Infrastructure as Code Library.