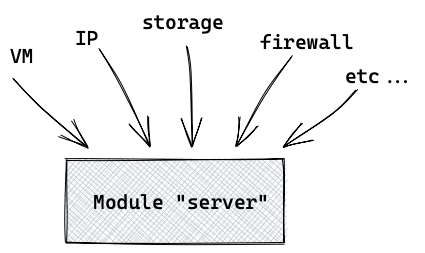
**What does a module do?**

**A Terraform module allows you to create logical abstraction on the top of some resource set. In other words, a module allows you to group resources together and reuse this group later, possibly many times.**

**Let's assume we have a virtual server with some features hosted in the cloud. What set of resources might describe that server? For example:**

* **the virtual machine itself, created from some image**
* **an attached block device of a specified size for additional storage**
* **a static public IP mapped to the server's virtual network interface**
* **a set of firewall rules to be attached to the server**
* **other things like another block device, additional network interface, and so on**

****

**Now let's assume that you need to create this server with a set of resources many times. This is where modules are really helpful – you don't want to repeat the same configuration code over and over again, do you?**

**Here is an example that illustrates how our "server" module might be called.  
"*To call a module*" means to use it in the configuration file.**

**Here we create 5 instances of the "server" using single set of configurations (in the module):**

**module "server" {**

**count = 5**

**source = "./module\_server"**

**some\_variable = some\_value**

**}**

**Terraform supports "count" for modules starting from version 0.13**

**Module organisation: child and root**

**Of course, you would probably want to create more than one module. Here are some common examples:**

* **a network like a virtual private cloud (VPC)**
* **static content hosting (i.e. buckets)**
* **a load balancer and it's related resources**
* **a logging configuration**
* **or whatever else you consider a distinct logical component of the infrastructure**

**Let's say we have two different modules: a "server" module and a "network" module. The module called "network" is where we define and configure our virtual network and place servers in it:**

**module "server" {**

**source = "./module\_server"**

**some\_variable = some\_value**

**}**

**module "network" {**

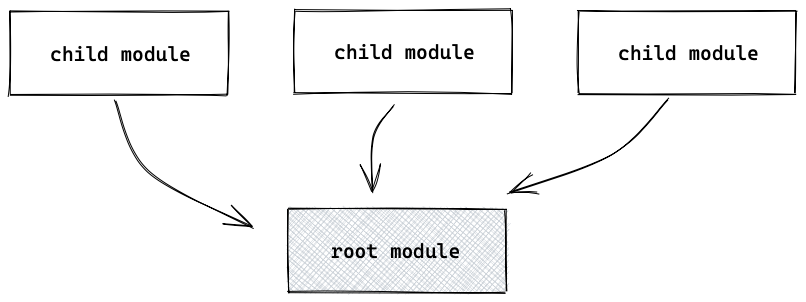
**source = "./module\_network"**

**some\_other\_variable = some\_other\_value**

**}**

**Two different child modules called in the root module**

**Once we have some custom modules, we can refer to them as "child" modules. And the configuration file where we call child modules relates to the root module.**

****

**A child module can be sourced from a number of places:**

* **local paths**
* **the official Terraform Registry – if you're familiar with other registries like the Docker Registry then you already understand the idea**
* **a Git repository (a custom one or GitHub/BitBucket)**
* **an HTTP URL to a .zip archive with the module**

**But how can you pass resources details between modules?**

**In our example, the servers should be created in a network. So how can we tell the "server" module to create VMs in a network which was created in a module called "network"?**

**This is where encapsulation comes in.**

**Module encapsulation**

**Encapsulation in Terraform consists of two basic concepts: module scope and explicit resource exposure.**

**Module Scope**

**All resource instances, names, and therefore, resource visibility, are isolated in a module's scope. For example, module "A" can't see and does not know about resources in module "B" by default.**

**Resource visibility, sometimes called resource isolation, ensures that resources will have unique names within a module's namespace. For example, with our 5 instances of the "server" module:**

**module.server[0].resource\_type.resource\_name**

**module.server[1].resource\_type.resource\_name**

**module.server[2].resource\_type.resource\_name**

**...**

**Module resource addresses created with the count meta-argument**

**On the other hand, we could create two instances of the same module with different names:**

**module "server-alpha" {**

**source = "./module\_server"**

**some\_variable = some\_value**

**}**

**module "server-beta" {**

**source = "./module\_server"**

**some\_variable = some\_value**

**}**

**Pay attention to the source argument — it remains the same, it is the same source module**

**In this case, the naming or address of resources would be as follows:**

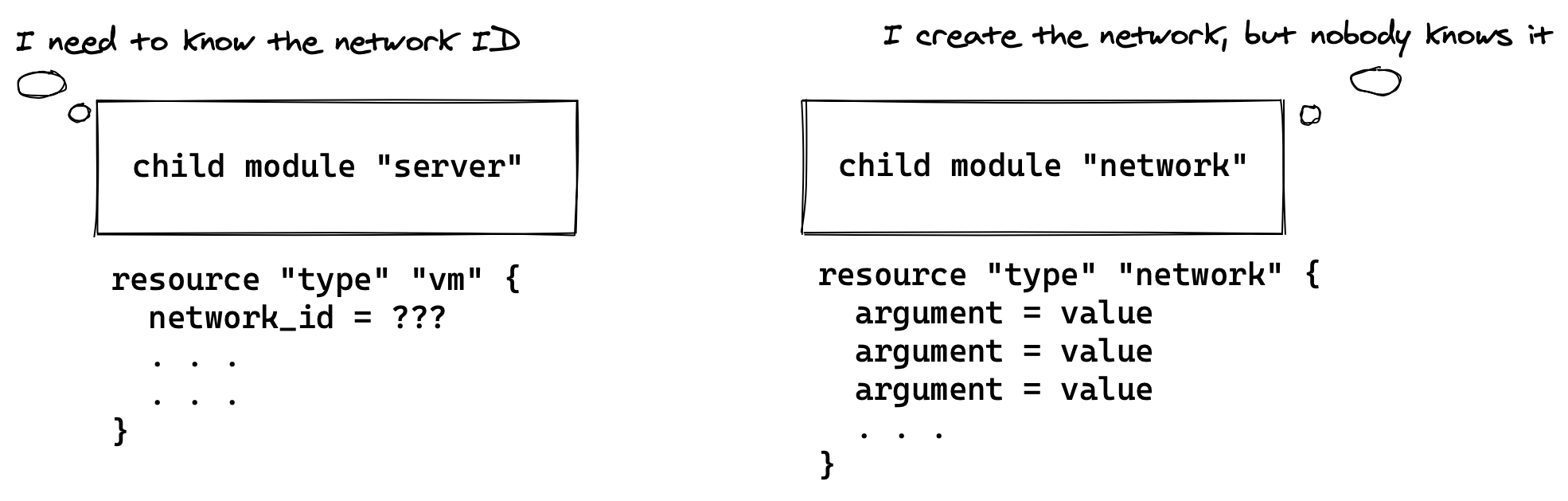
**module.server-alpha.resource\_type.resource\_name**

**module.server-beta.resource\_type.resource\_name**

**Explicit resource exposure**

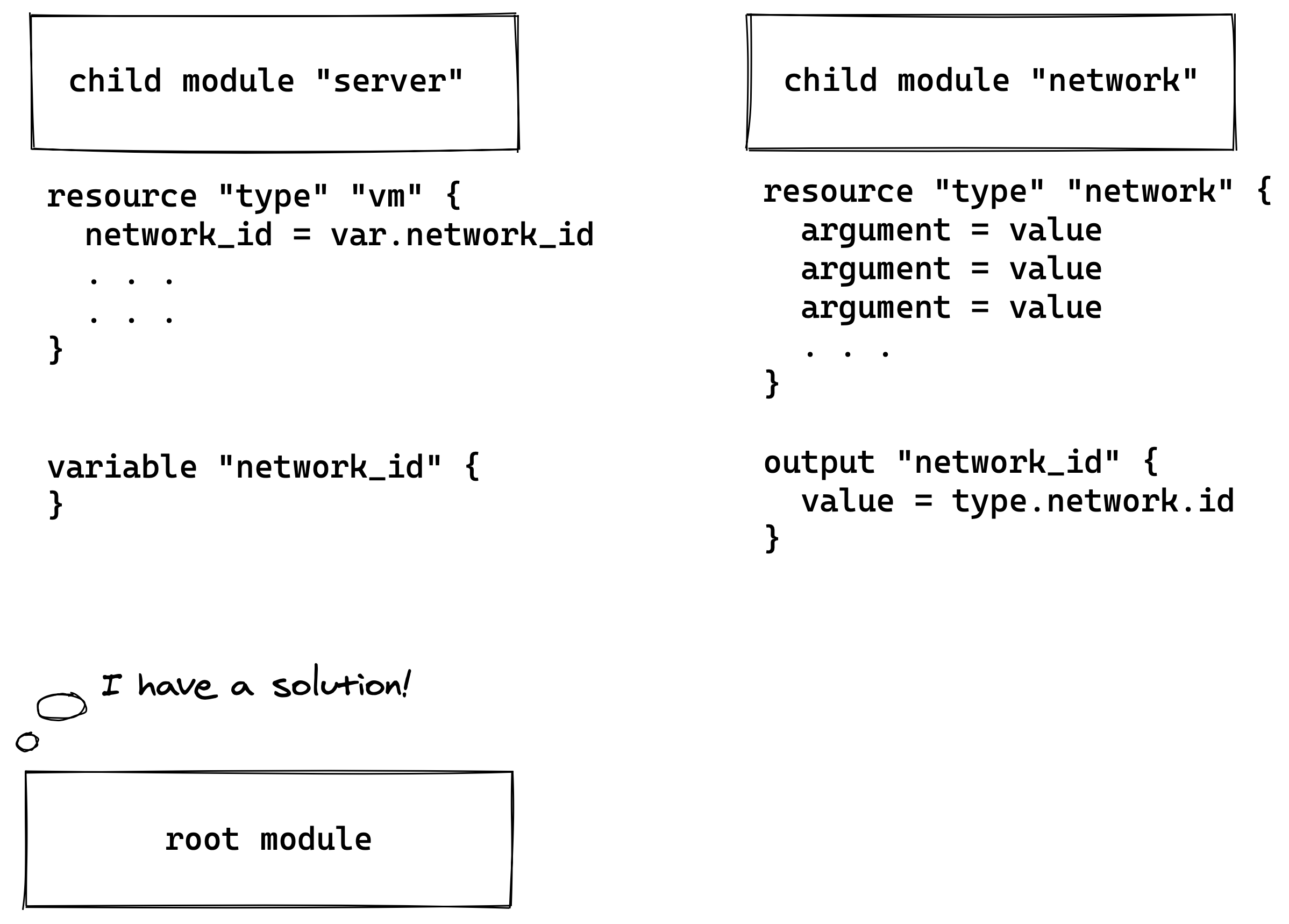
**If you want to access some details for the resources in another module, you'll need to explicitly configure that.**

**By default, our module "server" doesn't know about the network that was created in the "network" module.**

****

**So we must declare an output value in the "network" module to export its resource, or an attribute of a resource, to other modules.**

**The module "server" must declare a variable to be used later as the input:**

**The names output and variable can differ, but I suggest using the same names for clarity.**

**This explicit declaration of the output is the way to expose some resource (or information about it) outside — to the scope of the 'root' module, hence to make it available for other modules.**

**Next, when we call the child module "server" in the root module, we should assign the output from the "network" module to the variable of the "server" module:**

**network\_id = module.network.network\_id**

**Pay attention to the 'network\_id' output address here — we explicitly tell where it resides**

**Here's what the final code for calling our child modules will look like:**

**module "server" {**

**count = 5**

**source = "./module\_server"**

**some\_variable = some\_value**

**network\_id = module.network.network\_id**

**}**

**module "network" {**

**source = "./module\_network"**

**some\_other\_variable = some\_other\_value**

**}**

**This example configuration would create 5 instances of the same server, with all the necessary resources, in the network we created with as a separate module.**