# ARM ramp-up

## Reference Links

### ARM templates:

<https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/overview>

Create and deploy ARM templates:

<https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/template-tutorial-create-first-template?tabs=azure-powershell>

<https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/template-tutorial-add-resource?tabs=azure-powershell>

Define resources in ARM templates

<https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/template-tutorial-add-resource?tabs=azure-powershell>

Child Resources

<https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/child-resource-name-type>

Linked and Nested templates

<https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/linked-templates?tabs=azure-powershell>

## ARM templates

Basic ARM template - be able to explain the basic components of the structure of an ARM template, and use all of them to create a template of a basic resource.

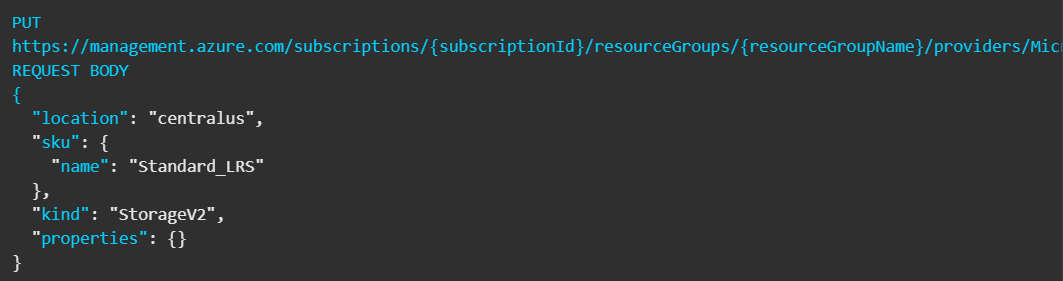
An ARM template is a JSON file where you define the resources you want to create and how to configure them. The template uses declarative syntax, which lets you state what you intend to deploy without having to write the sequence of programming commands to create it. You can also specify in which resource group those resources are deployed.

One of the main advantages of using these ARM templates is automation, since we can define our infrastructure and automate different processes.

It’s important to know that when we deploy an ARM template, the Resource Manager converts the template into REST API operations. For example, when Resource Manager receives a template with the following resource definition:



It converts the definition to the following REST API operation, which it sends to the Microsoft.Storage resource provider:

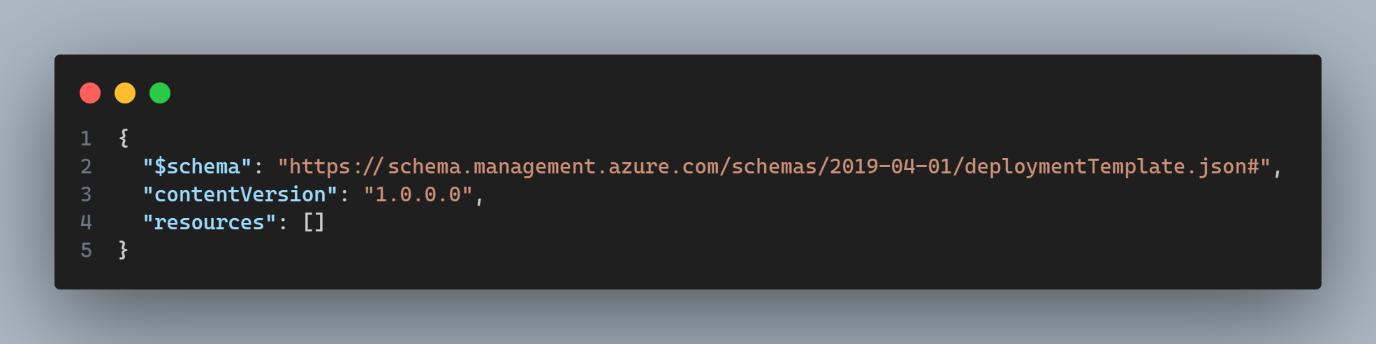


We can deploy these ARM templates using:

* Azure portal
* Azure CLI
* PowerShell
* REST API
* Button in GitHub repository
* Azure Cloud Shell

### Basic components in an ARM template

This template doesn't deploy any resources, but we can see the necessary elements an ARM template has:



* **$schema:**

The schema file describes the properties that are available within a template. The schema basically tells Azure and tools how to interpret and validate your template. For example, in this scenario the schema is telling azure “Use the ARM template schema rules from April 1, 2019”.

* **contentVersion:**

Specifies the version of the template, we can change this value as our preference.

* **resources:**

Contains the resources you want to deploy or update.

Even though those are the necessary elements in an ARM template we can add other such as:

* **parameters:**

these are the inputs you provide before the deployment starts.

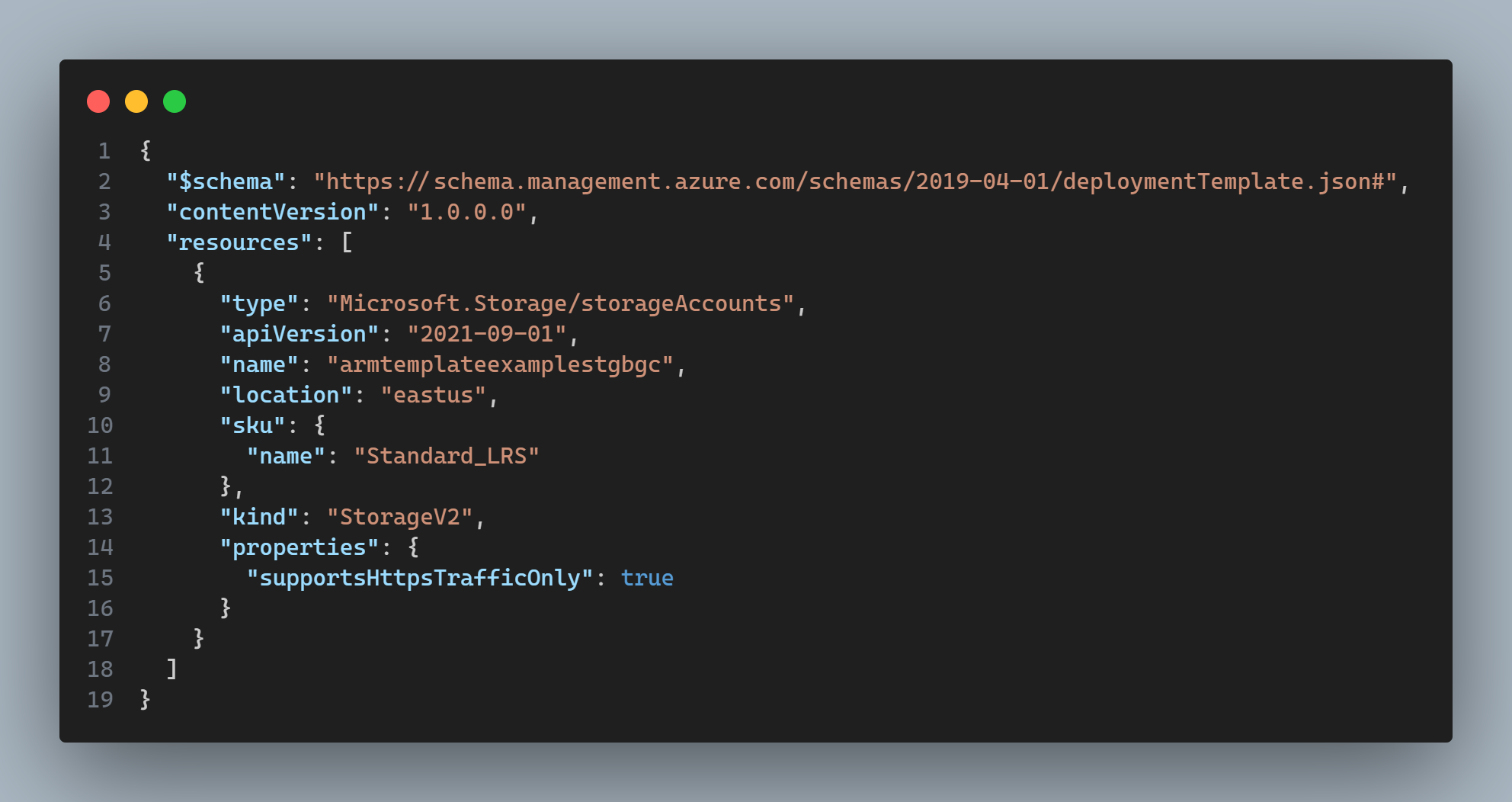
* **variables:**

These are helper values that are calculated inside of the template.

* **outputs:**

These are the results that Azure gives back to us when the deployment finishes.

Also, something that is important to know is that inside of the resources property, we can add all the resources we want to deploy, however depending of the resource you want to deploy the properties might change. As an example, we are going to deploy a simple storage account:



As I mentioned before, depending of the resource you want to deploy the properties might change, however every resource you deploy has at least the following three properties:

* **type**: Type of the resource. This value is a combination of the namespace of the resource provider and the resource type such as Microsoft.Storage/storageAccounts.
* **apiVersion**: Version of the REST API to use for creating the resource. Each resource provider publishes its own API versions, so this value is specific to the type.
* **name**: Name of the resource.

Most resources also have a **location** property, which sets the region where you deploy the resource.

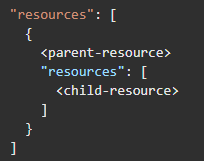
### Child resources

Child resources are resources that exist only within the context of another resource. For example, a virtual machine extension can't exist without a virtual machine. The extension resource is a child of the virtual machine.

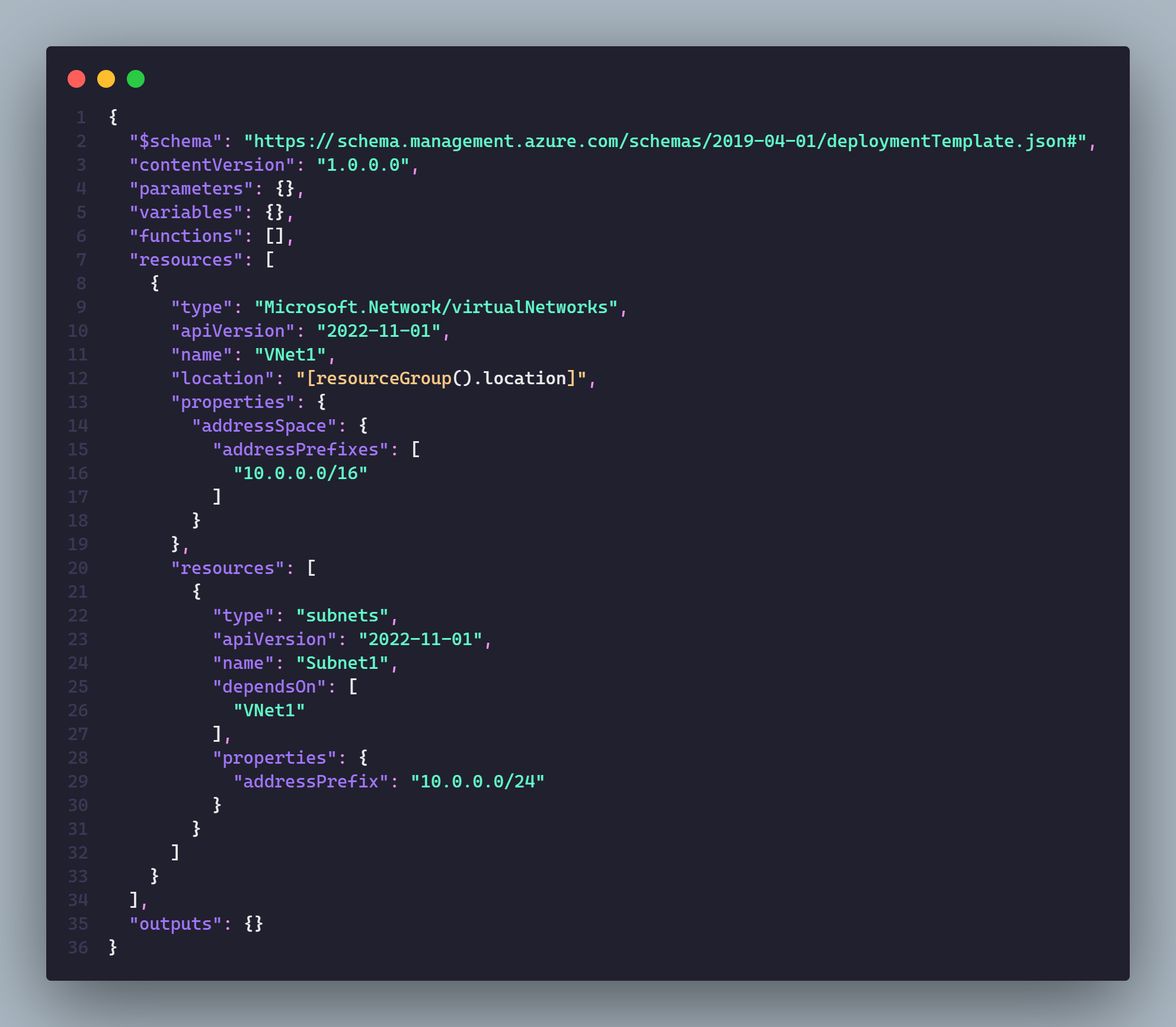
Now, we have two options when it comes to child resources.

#### Define the child resource within the parent resource

Here, the child resources are going to be inside of the resources property from the parent resource, for example:



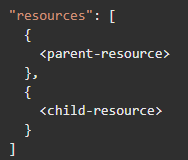
In real life we can see this child resources for example with the subnets, that can be inside of a Virtual Network. For example:



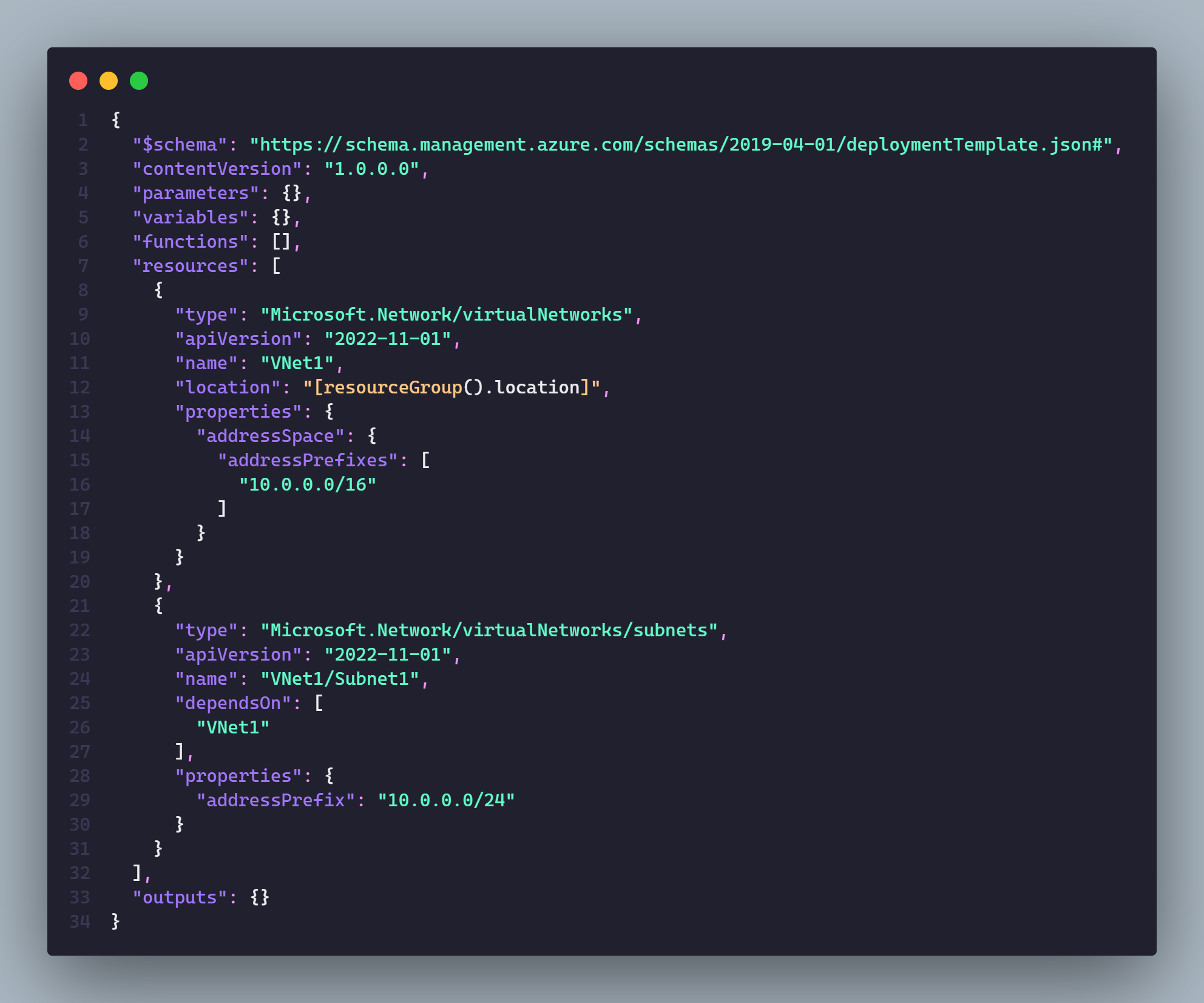
Notice that the full resource type is still Microsoft.Network/virtualNetworks/subnets. You don't provide Microsoft.Network/virtualNetworks/ because it's assumed from the parent resource type.

#### Define the child the Outside parent resource

To define a child resource outside of the parent resource we can do it in the following way:



Continuing with the last example of the VNet and the subnet we could define it like this:



Notice that we have the subnet in the root level of the resources property and in this case we actually add the complete name on the **type** property.

### Nested templates

### Multiple instances

### Conditional Templates

### Deployment Scripts

### Bicep Templates