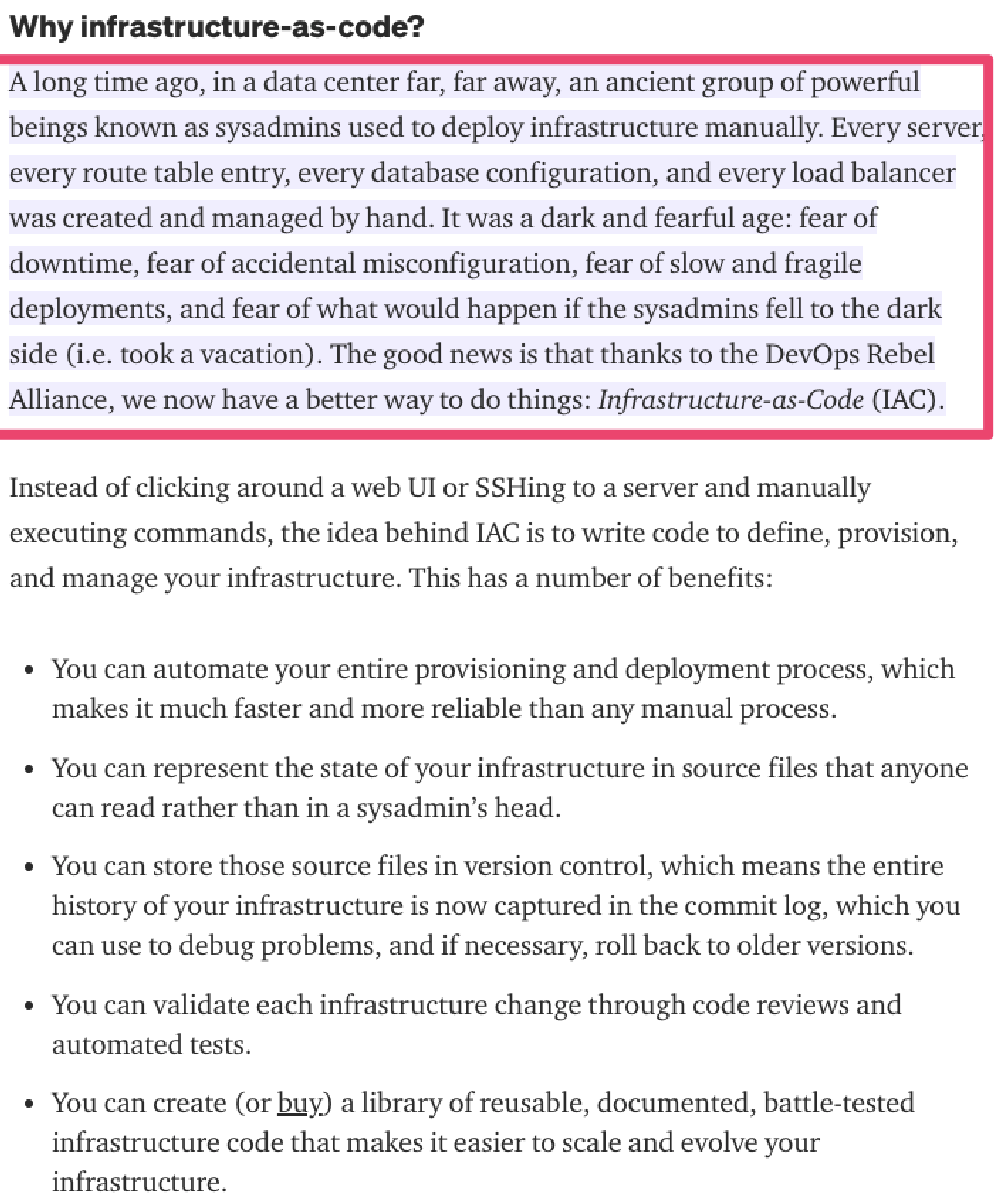
Day002

Why Infrastructure-as-code (IaC)



Why infrastructure-as-code?

A long time ago, in a data center far, far away, an ancient group of powerful beings known as sysadmins used to deploy infrastructure manually. Every server, every route table entry, every database configuration, and every load balancer was created and managed by hand. It was a dark and fearful age: fear of downtime, fear of accidental misconfiguration, fear of slow and fragile deployments, and fear of what would happen if the sysadmins fell to the dark side (i.e. took a vacation). The good news is that thanks to the DevOps Rebel Alliance, we now have a better way to do things: *Infrastructure-as-Code* (IAC).

Instead of clicking around a web UI or SSHing to a server and manually executing commands, the idea behind IAC is to write code to define, provision, and manage your infrastructure. This has a number of benefits:

- You can automate your entire provisioning and deployment process, which makes it much faster and more reliable than any manual process.
- You can represent the state of your infrastructure in source files that anyone can read rather than in a sysadmin's head.
- You can store those source files in version control, which means the entire history of your infrastructure is now captured in the commit log, which you can use to debug problems, and if necessary, roll back to older versions.
- You can validate each infrastructure change through code reviews and automated tests.
- You can create (or buy) a library of reusable, documented, battle-tested infrastructure code that makes it easier to scale and evolve your infrastructure.

Credit: @Gruntwork

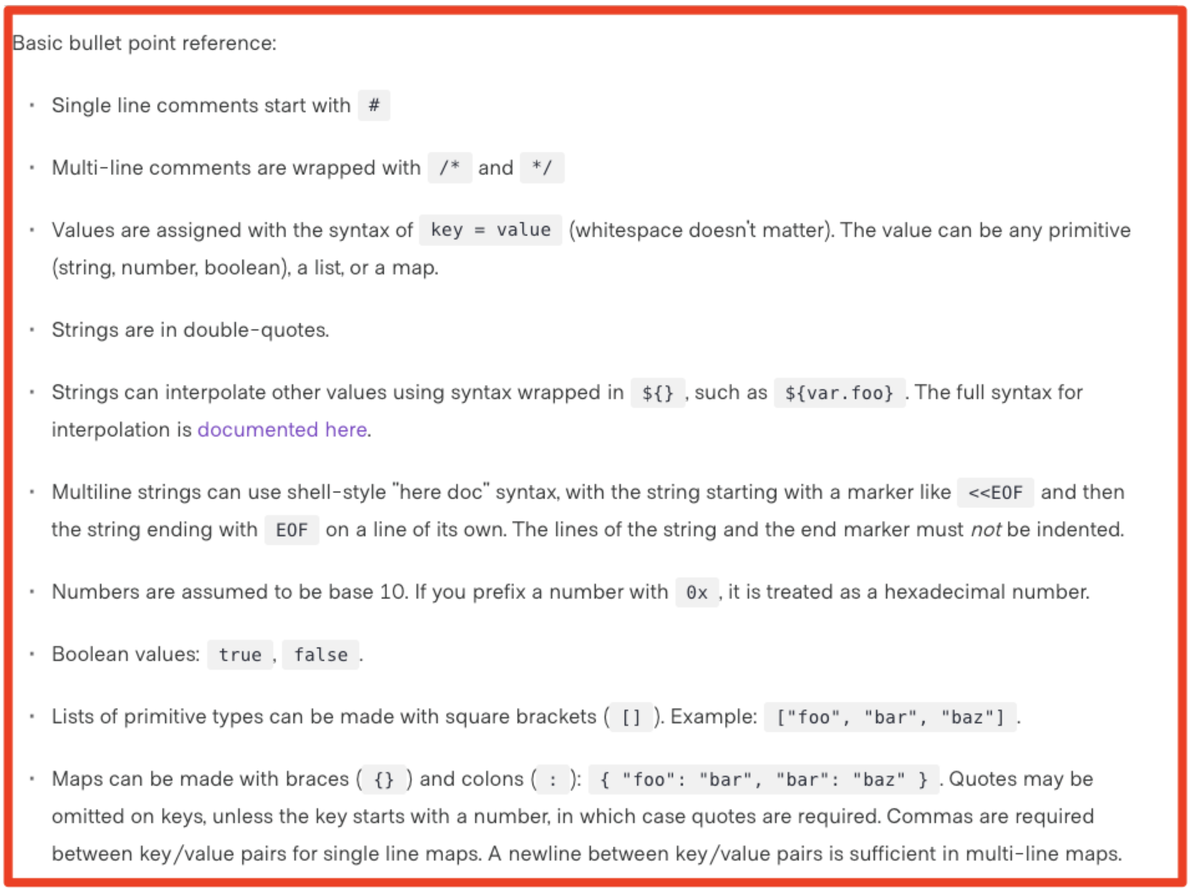
What is HCL

The syntax of Terraform configurations is called HashiCorp Configuration Language (HCL).

```
# An AMI
variable "ami" {
  description = "the AMI to use"
}

/* A multi
   line comment. */
resource "aws_instance" "web" {
  ami               = "${var.ami}"
  count             = 2
  source_dest_check = false

  connection {
    user = "root"
  }
}
```

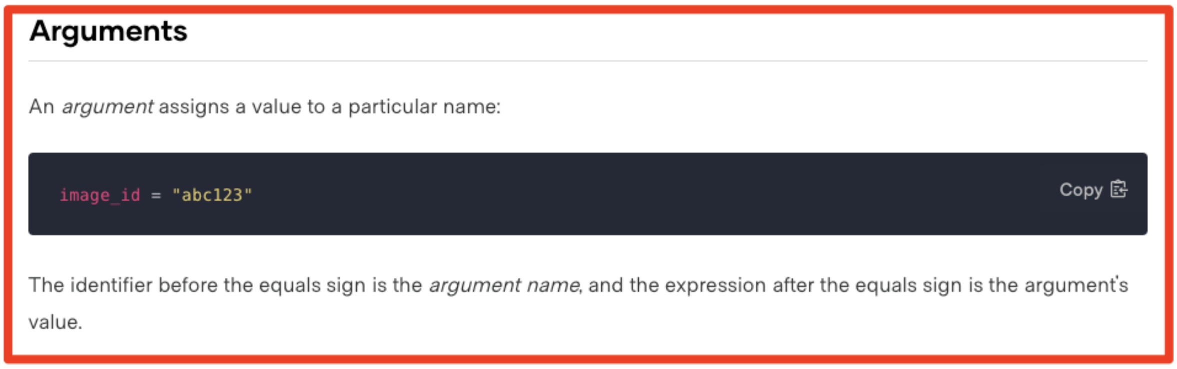


Basic bullet point reference:

- Single line comments start with `#`
- Multi-line comments are wrapped with `/*` and `*/`
- Values are assigned with the syntax of `key = value` (whitespace doesn't matter). The value can be any primitive (string, number, boolean), a list, or a map.
- Strings are in double-quotes.
- Strings can interpolate other values using syntax wrapped in `${}`, such as `${var.foo}`. The full syntax for interpolation is documented here.
- Multiline strings can use shell-style "here doc" syntax, with the string starting with a marker like `<<EOF` and then the string ending with `EOF` on a line of its own. The lines of the string and the end marker must *not* be indented.
- Numbers are assumed to be base 10. If you prefix a number with `0x`, it is treated as a hexadecimal number.
- Boolean values: `true`, `false`.
- Lists of primitive types can be made with square brackets (`[]`). Example: `["foo", "bar", "baz"]`.
- Maps can be made with braces (`{}`) and colons (`:`): `{ "foo": "bar", "bar": "baz" }`. Quotes may be omitted on keys, unless the key starts with a number, in which case quotes are required. Commas are required between key/value pairs for single line maps. A newline between key/value pairs is sufficient in multi-line maps.

Arguments and Blocks

The Terraform language syntax is built around two key syntax constructs: arguments and blocks.

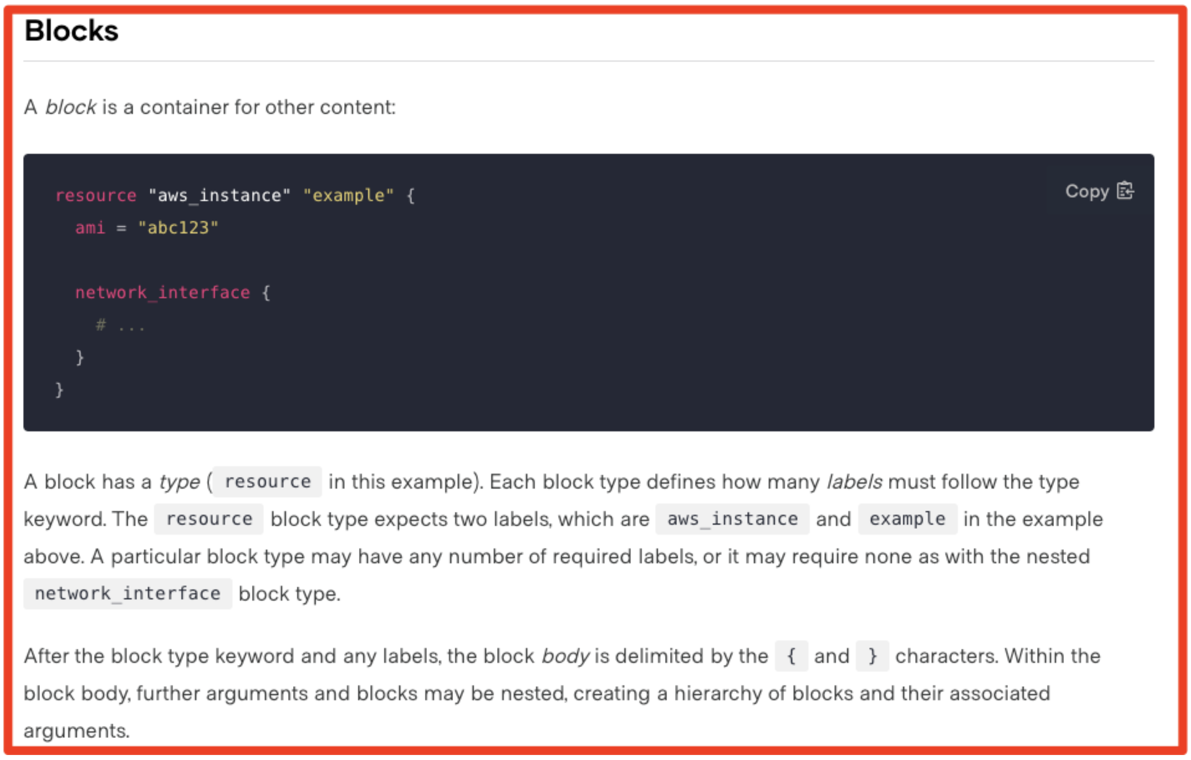


Arguments

An *argument* assigns a value to a particular name:

```
image_id = "abc123"
```

The identifier before the equals sign is the *argument name*, and the expression after the equals sign is the argument's value.



Blocks

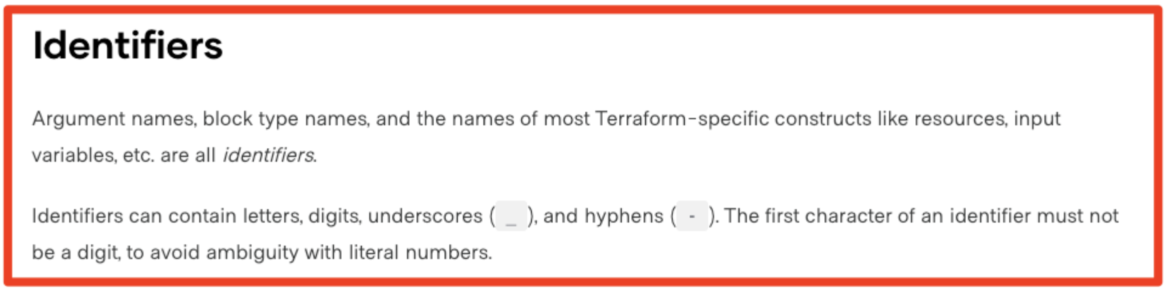
A *block* is a container for other content:

```
resource "aws_instance" "example" {
  ami = "abc123"

  network_interface {
    # ...
  }
}
```

A block has a *type* (`resource` in this example). Each block type defines how many *labels* must follow the type keyword. The `resource` block type expects two labels, which are `aws_instance` and `example` in the example above. A particular block type may have any number of required labels, or it may require none as with the nested `network_interface` block type.

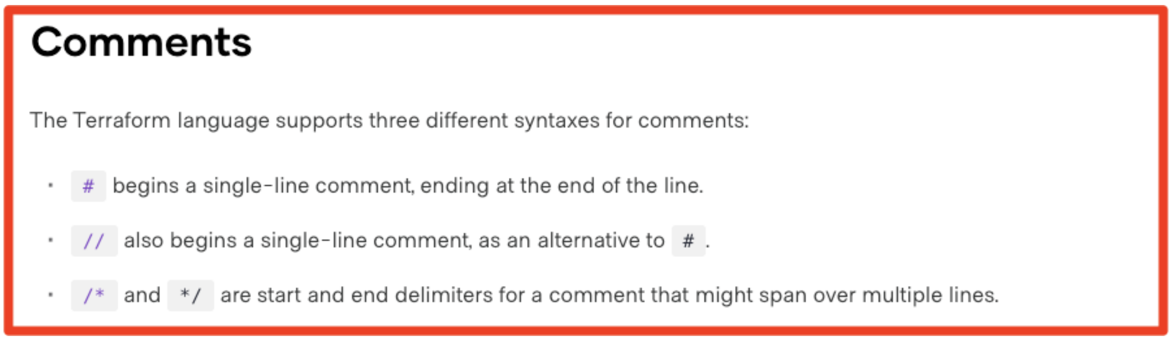
After the block type keyword and any labels, the block *body* is delimited by the `{` and `}` characters. Within the block body, further arguments and blocks may be nested, creating a hierarchy of blocks and their associated arguments.



Identifiers

Argument names, block type names, and the names of most Terraform-specific constructs like resources, input variables, etc. are all *identifiers*.

Identifiers can contain letters, digits, underscores (`_`), and hyphens (`-`). The first character of an identifier must not be a digit, to avoid ambiguity with literal numbers.



Comments

The Terraform language supports three different syntaxes for comments:

- `#` begins a single-line comment, ending at the end of the line.
- `//` also begins a single-line comment, as an alternative to `#`.
- `/*` and `*/` are start and end delimiters for a comment that might span over multiple lines.