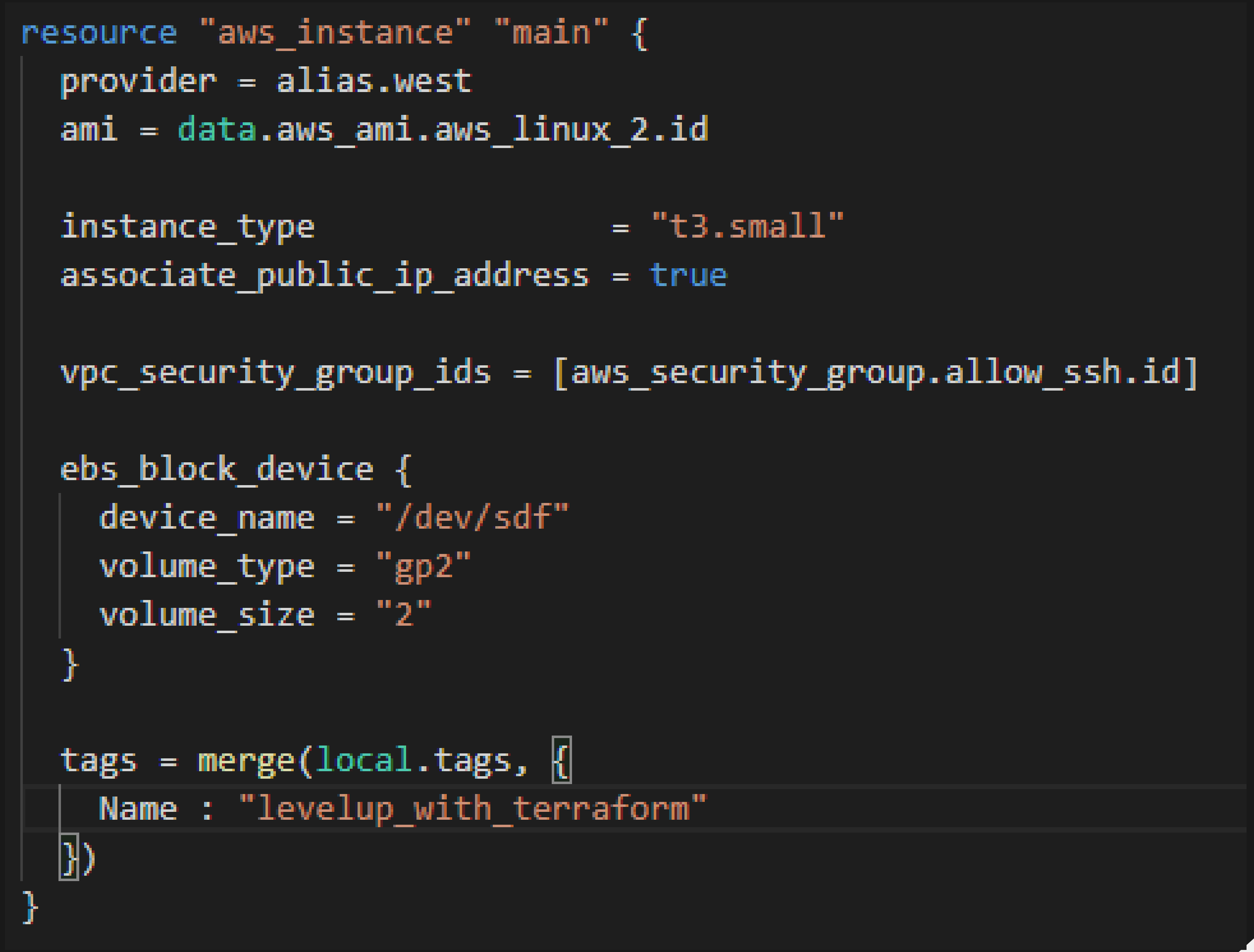
Terraform HCL Introduction

HashiCorp Terraform

HCL Introduction

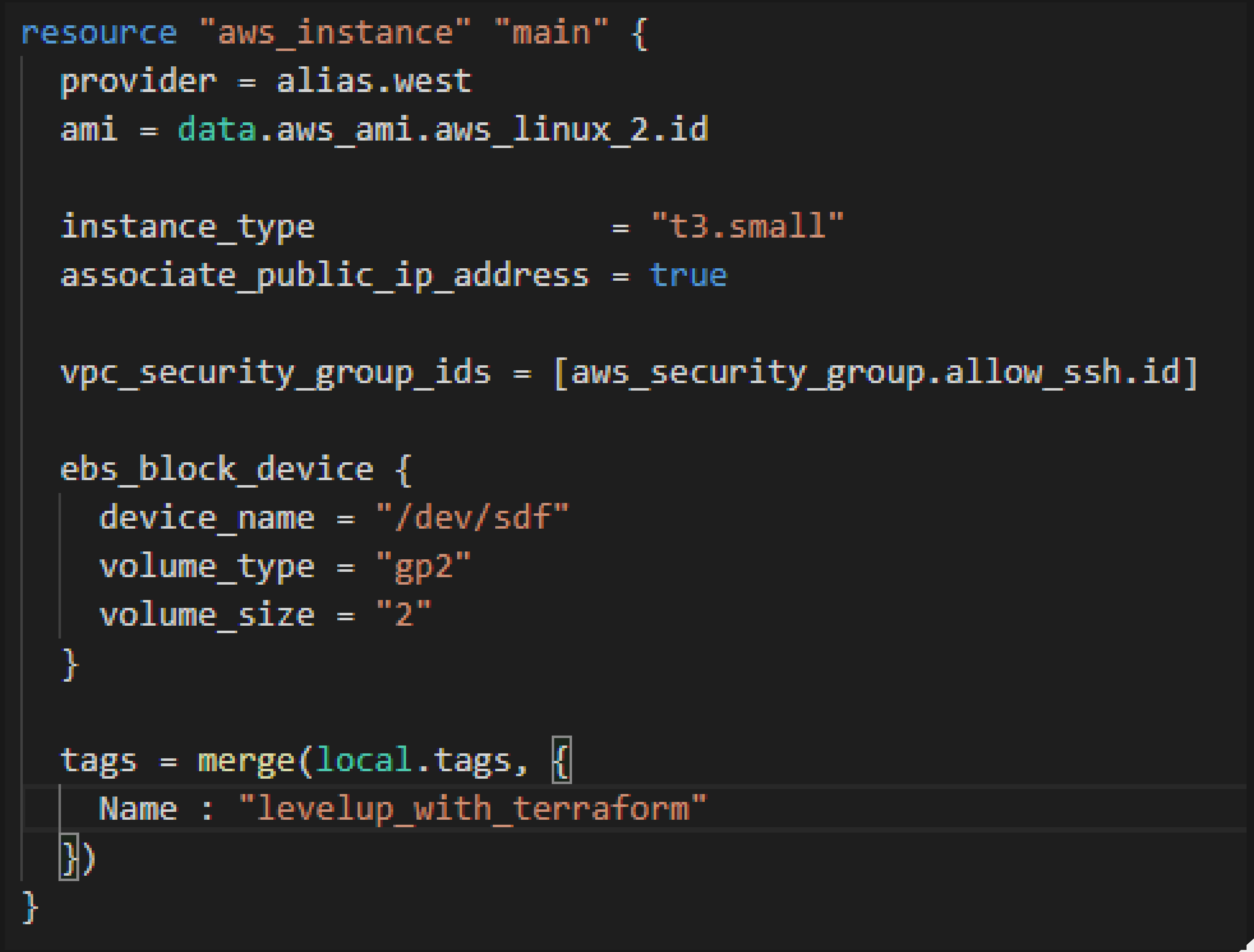
- HCL is the main language of Terraform
  - Developed by HashiCorp
  - More of a configuration language than a programming language
  - Extremely easy to read and learn
  - HCL is specific to Terraform and not adopted by any other framework
- Possible to write Terraform code in JSON
  - No one does this - stick with HCL!



```
resource "aws_instance" "main" {
  provider = alias.west
  ami = data.aws_ami.aws_linux_2.id

  instance_type               = "t3.small"
  associate_public_ip_address = true

  vpc_security_group_ids = [aws_security_group.allow_ssh.id]

  ebs_block_device {
    device_name = "/dev/sdf"
    volume_type = "gp2"
    volume_size = "2"
  }

  tags = merge(local.tags, {
    Name : "levelup_with_terraform"
  })
}
```

HCL Introduction

- Creating resources in Terraform is all about:
  - Declaring inputs and providing them values
  - Every resource has set of inputs, required or optional
  - Values provided to inputs, define how the resource is created
  - instance_type = "t3.small" defines our ec2 instance to use a t3.small instance type
- Creating infrastructure in Terraform is easy:
  - Emphasis on platform knowledge (AWS, GCP, etc.)

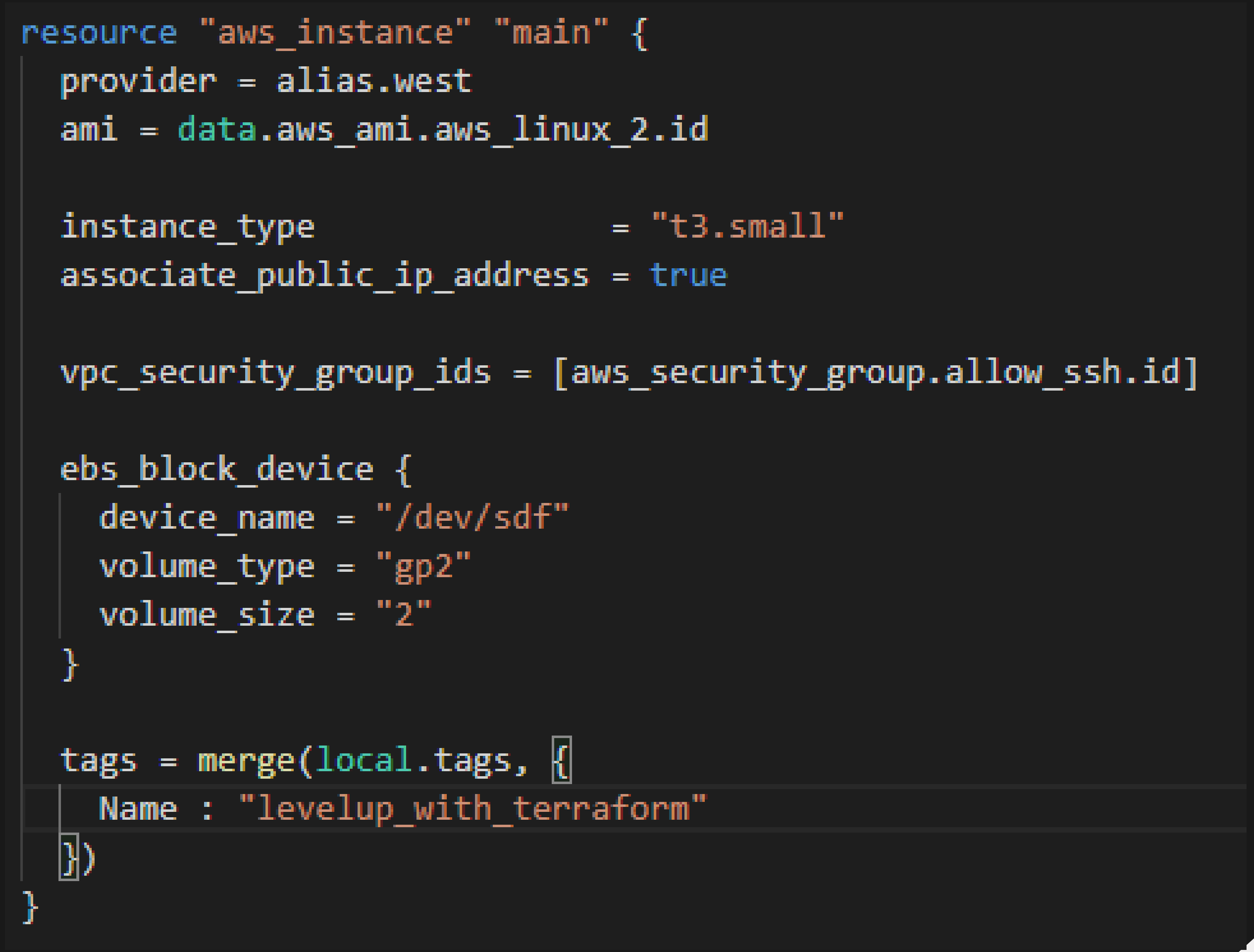


```
resource "aws_instance" "main" {
  provider = alias.west
  ami = data.aws_ami.aws_linux_2.id

  instance_type               = "t3.small"
  associate_public_ip_address = true

  vpc_security_group_ids = [aws_security_group.allow_ssh.id]

  ebs_block_device {
    device_name = "/dev/sdf"
    volume_type = "gp2"
    volume_size = "2"
  }

  tags = merge(local.tags, {
    Name : "levelup_with_terraform"
  })
}
```

HashiCorp Terraform

HCL Introduction

- Being a good Terraform coder is about:
  - Implementing readable code
  - Implementing best practices - using right file names, good variable names, good resource names, etc.
  - Reducing duplication - using modules for collective resources, local variables for static defined variables, built-in functions, etc.
  - Organizing projects and resources
  - Understanding state internals and resource dependencies
- Always consult Terraform resource documentation for list of available inputs and outputs
  - Every resource has collection of required and optional inputs
  - Every resource has collection of outputs - attributes exposed upon resource creation

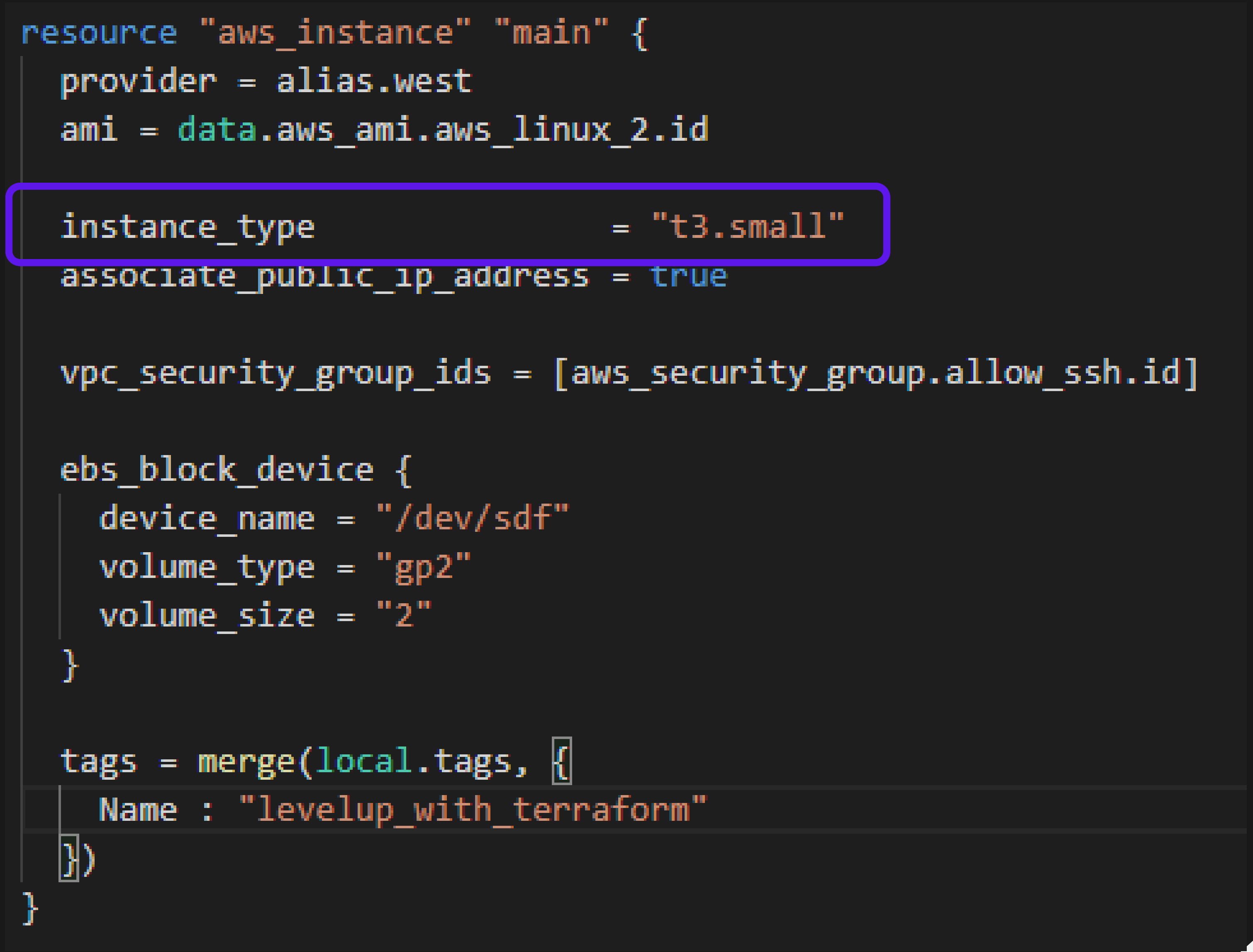


```
resource "aws_instance" "main" {
  provider = alias.west
  ami = data.aws_ami.aws_linux_2.id

  instance_type               = "t3.small"
  associate_public_ip_address = true

  vpc_security_group_ids = [aws_security_group.allow_ssh.id]

  ebs_block_device {
    device_name = "/dev/sdf"
    volume_type = "gp2"
    volume_size = "2"
  }

  tags = merge(local.tags, {
    Name : "levelup_with_terraform"
  })
}
```

HashiCorp
Terraform

Arguments

- Arguments assign a value to an attribute or variable
  - In example, we assign a string "t3.small" to an attribute called instance_type which is a part of the aws_instance resource type
  - The accepted value type depends on the attribute
  - For instance_type attribute, aws_instance is expecting a string
- In resources, arguments are used to provide information to the resource, instructing it on how the resource is created
  - Create an EC2 instance, using a t3.small instance type
  - Reference the documentation for resource type to get full list of required and optional arguments

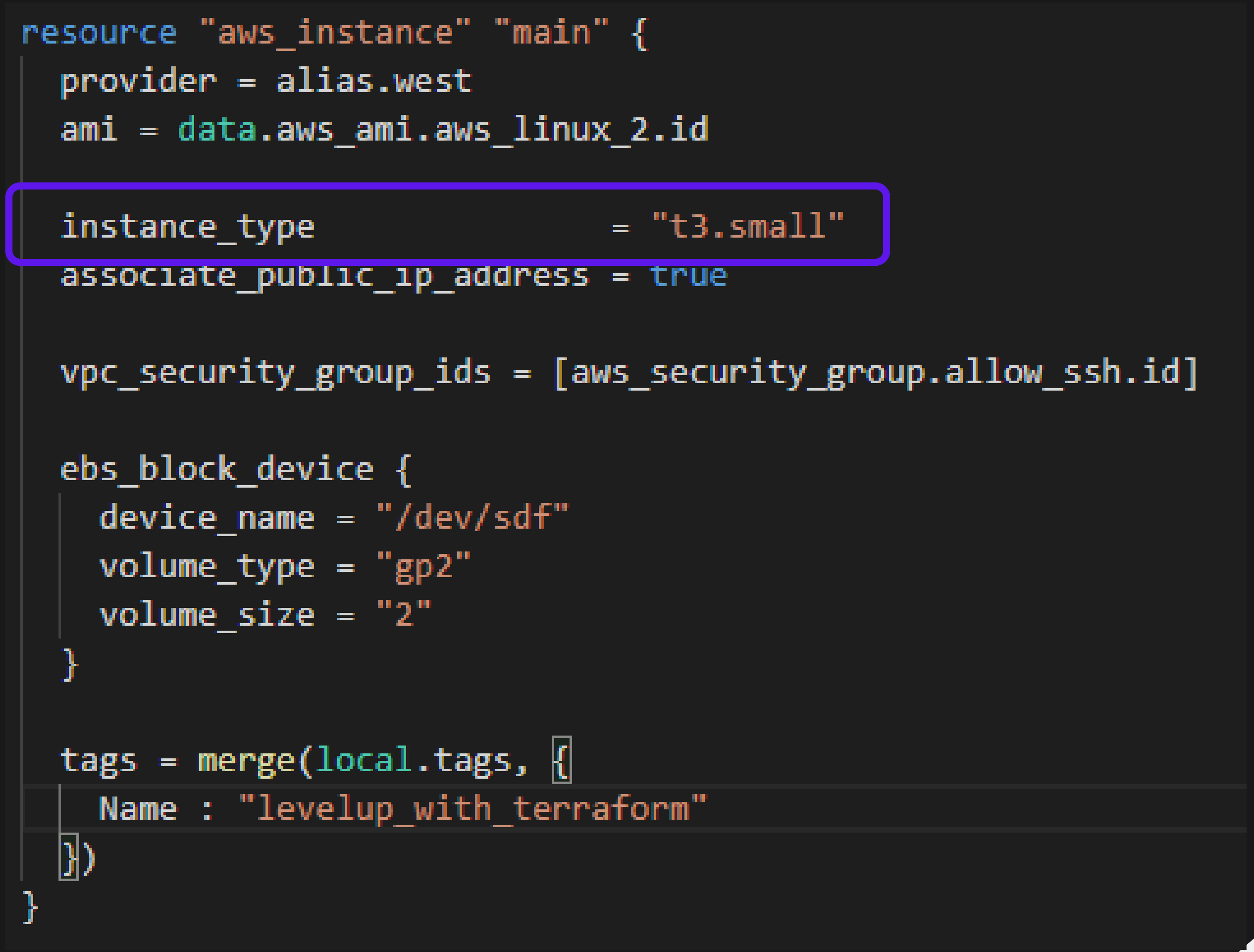


```
resource "aws_instance" "main" {
  provider = alias.west
  ami = data.aws_ami.aws_linux_2.id

  instance_type               = "t3.small"
  associate_public_ip_address = true

  vpc_security_group_ids = [aws_security_group.allow_ssh.id]

  ebs_block_device {
    device_name = "/dev/sdf"
    volume_type = "gp2"
    volume_size = "2"
  }

  tags = merge(local.tags, {
    Name : "levelup_with_terraform"
  })
}
```

HashiCorp
Terraform

Arguments

- Arguments can be of varying types
  - string: A string literal or a collection of characters
  - number: A numeric value (both whole and fractional)
  - bool: True or false
  - list: An array, sequence of values accessible by index
    - [1, 3, 5, 6]
    - ["us-west-1a", "us-west-1c"]
  - map: An object
    - {name = "Bob", age = "24"}
  - null: Value which represents absence or omission
- Argument types can also be casted to other types, if permissible
  - Casting a string "5" into a number 5

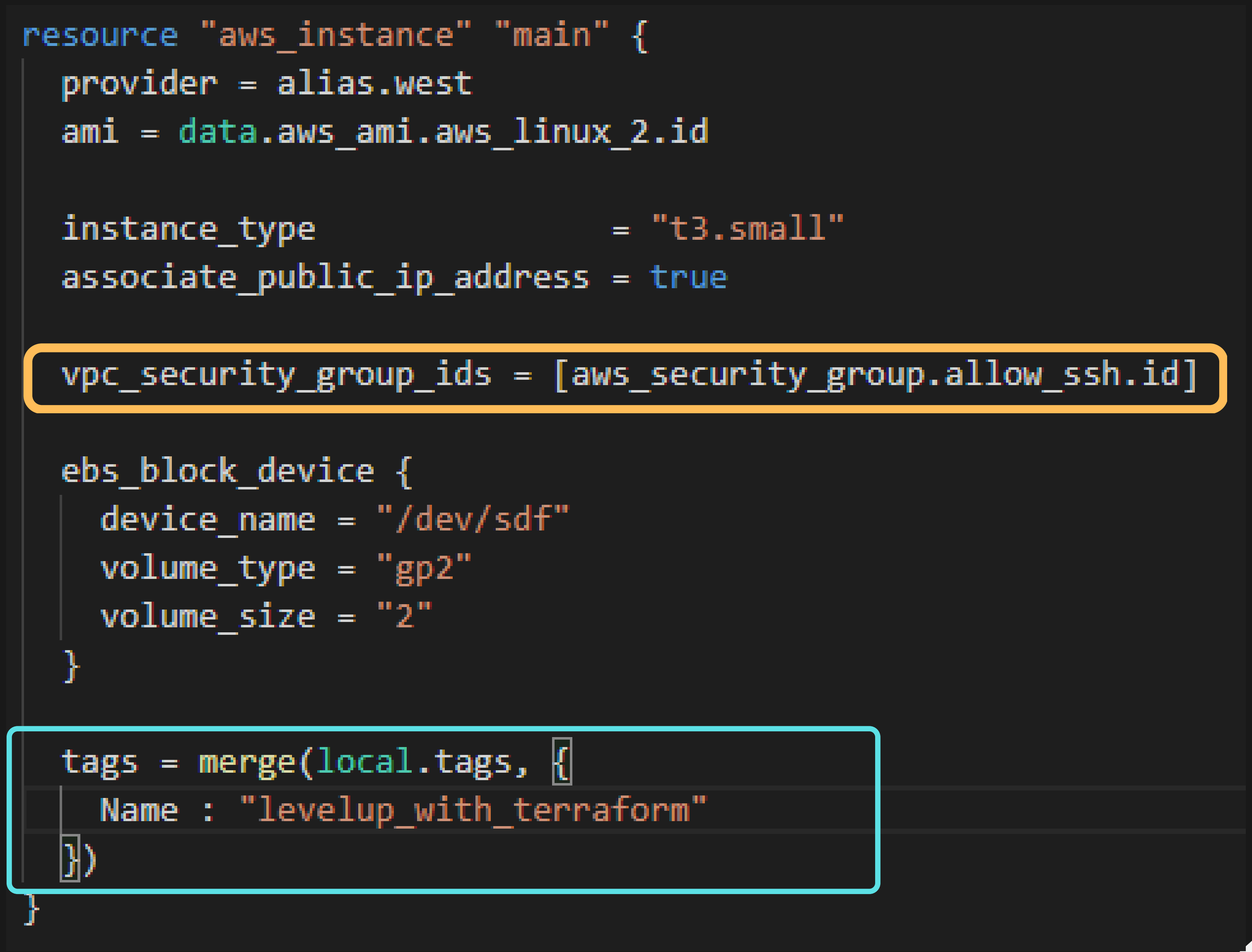


```
resource "aws_instance" "main" {
  provider = alias.west
  ami = data.aws_ami.aws_linux_2.id

  instance_type               = "t3.small"
  associate_public_ip_address = true

  vpc_security_group_ids = [aws_security_group.allow_ssh.id]

  ebs_block_device {
    device_name = "/dev/sdf"
    volume_type = "gp2"
    volume_size = "2"
  }

  tags = merge(local.tags, {
    Name : "levelup_with_terraform"
  })
}
```

HashiCorp Terraform

Arguments

- A list or an array in Terraform, is similar to arrays in other programming languages like Python
  - Starts at a zero index
  - Denoted by a square bracket
  - Values can be accessed by supplying a valid index
  - local.some_list[3]
- Maps are like objects, similar to objects in other languages like dictionaries in Python
  - Denoted by curly braces containing series of key-value pairs
  - Values can be accessed by supplying a valid key string
  - local.some_obj["name"]
  - local.some_obj.attributename

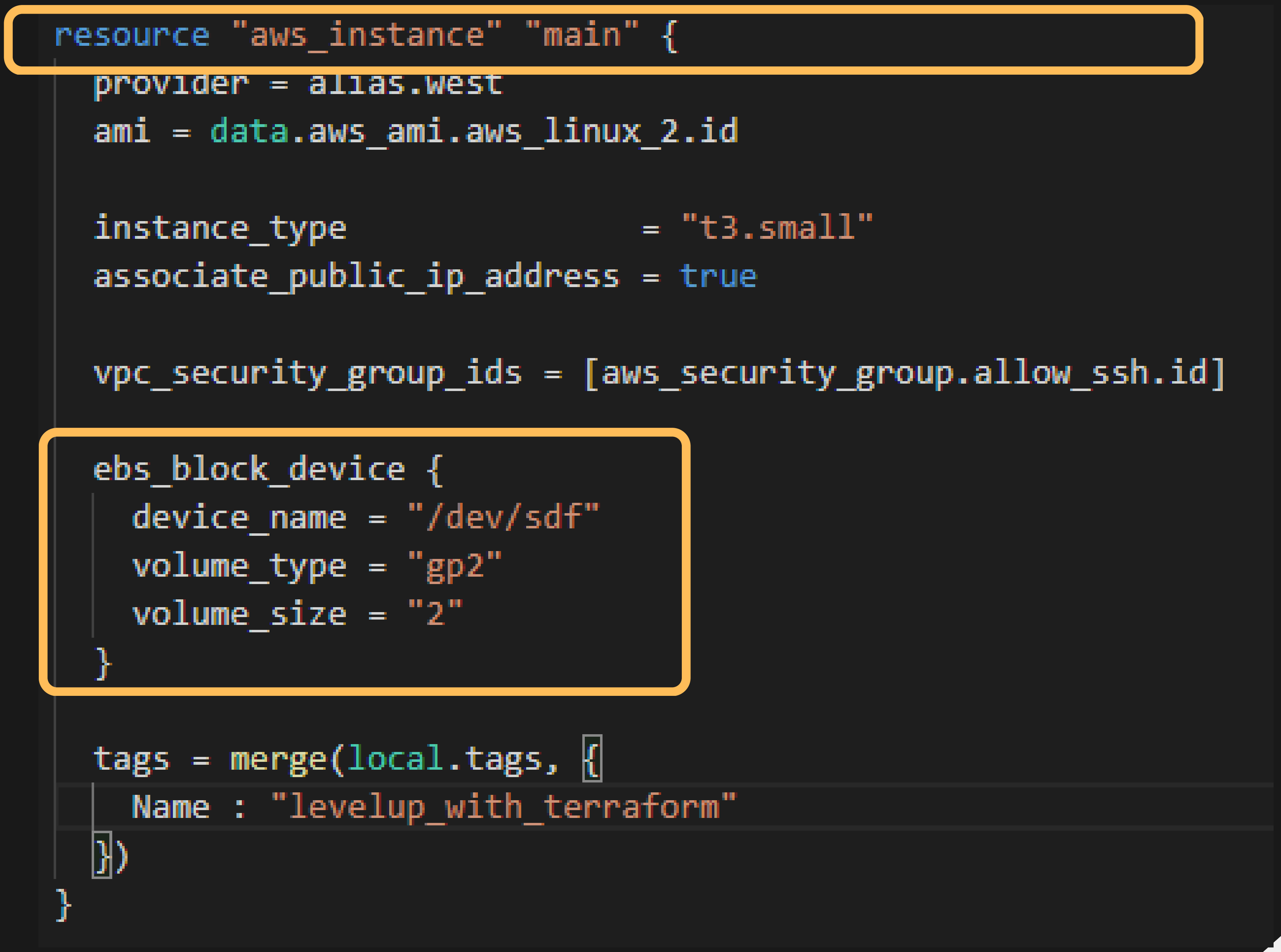


```
resource "aws_instance" "main" {
  provider = alias.west
  ami = data.aws_ami.aws_linux_2.id

  instance_type               = "t3.small"
  associate_public_ip_address = true

  vpc_security_group_ids = [aws_security_group.allow_ssh.id]

  ebs_block_device {
    device_name = "/dev/sdf"
    volume_type = "gp2"
    volume_size = "2"
  }

  tags = merge(local.tags, {
    Name : "levelup_with_terraform"
  })
}
```

HashiCorp
Terraform

Blocks

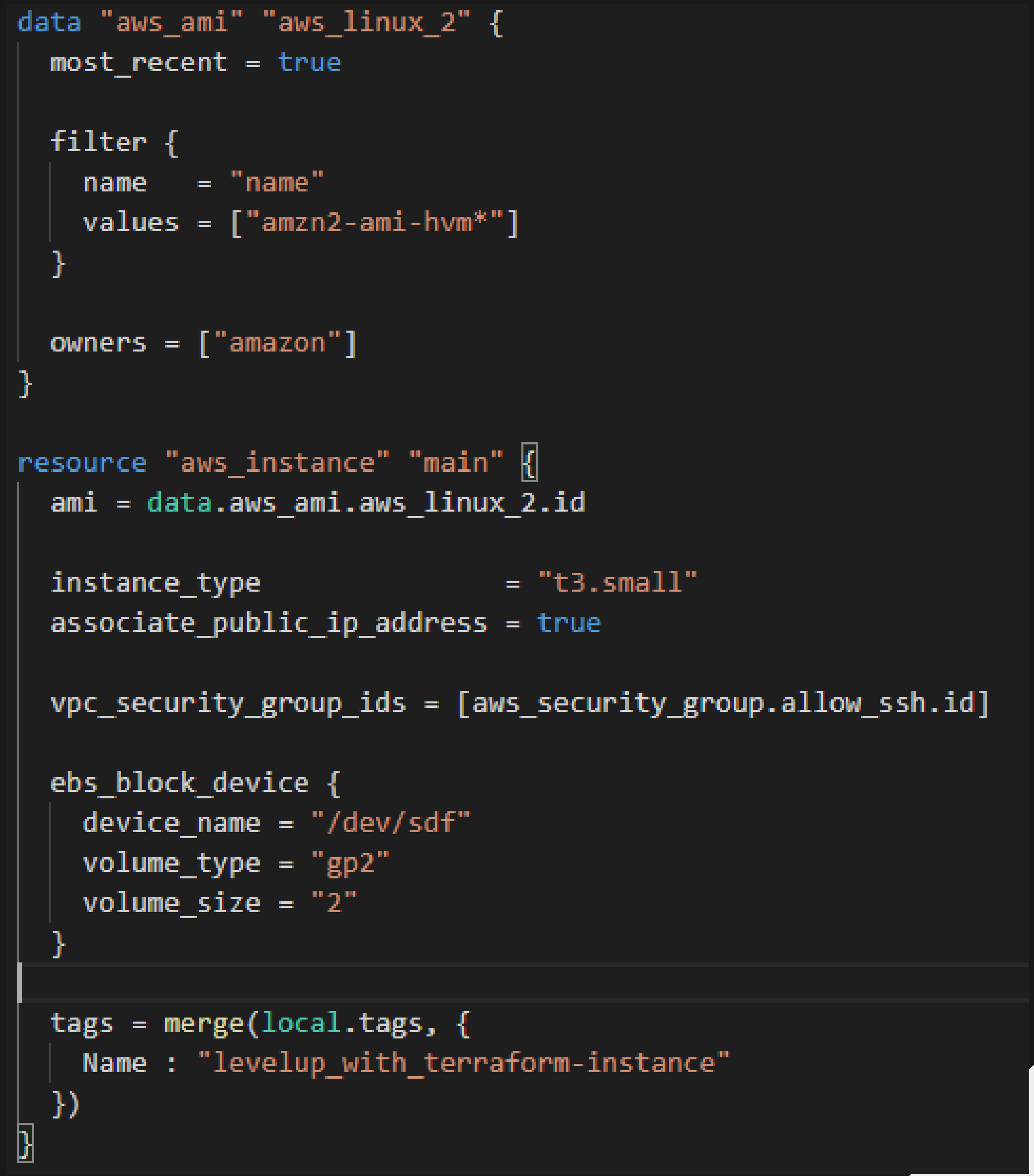
- Blocks are the fundamental building blocks of Terraform
  - Blocks are used to define some content
  - Like Arguments, Blocks also have types
  - Resource type block - denotes some resource to create
- Block types come with required labels
  - Resource type requires two labels
    - resource_type: aws_instance
    - identifier: a name assigned to the resource
  - Some Blocks have no labels required, like the global terraform block
- Like Maps, Blocks are denoted by the curly brackets, but does not require an equals sign
- Blocks can also be nested within other blocks



```
resource "aws_instance" "main" {
  provider = alias.west
  ami = data.aws_ami.aws_linux_2.id

  instance_type               = "t3.small"
  associate_public_ip_address = true

  vpc_security_group_ids = [aws_security_group.allow_ssh.id]

  ebs_block_device {
    device_name = "/dev/sdf"
    volume_type = "gp2"
    volume_size = "2"
  }

  tags = merge(local.tags, {
    Name : "levelup_with_terraform"
  })
}
```

Creating Resources

- When creating Resources or Modules, understanding Arguments & Blocks cover 80% of your usecases
  - Understand the basics & fundamentals
  - Ready to start creating Terraform resources and modules!
- Working with Terraform mostly involves creating Resources or Modules with Blocks, and providing values with Arguments
  - Utilizing official documentation is key
    - Impossible to memorize everything
- HCL language is simple to read and quick to learn
  - Although simple, leaves plenty of room for complex logic



```
data "aws_ami" "aws_linux_2" {
  most_recent = true

  filter {
    name   = "name"
    values = ["amzn2-ami-hvm*"]
  }

  owners = ["amazon"]
}

resource "aws_instance" "main" {
  ami = data.aws_ami.aws_linux_2.id

  instance_type               = "t3.small"
  associate_public_ip_address = true

  vpc_security_group_ids = [aws_security_group.allow_ssh.id]

  ebs_block_device {
    device_name = "/dev/sdf"
    volume_type = "gp2"
    volume_size = "2"
  }

  tags = merge(local.tags, {
    Name : "levelup_with_terraform-instance"
  })
}
```

HashiCorp
Terraform