



# Terraform language, expressions, console and functions

Subtitle or speaker name



# Conditions and Terms of Use

Microsoft Confidential

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet website references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

## Copyright and Trademarks

© 2016 Microsoft Corporation. All rights reserved.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see **Use of Microsoft Copyrighted Content** at  
<https://www.microsoft.com/en-us/legal/intellectualproperty/permissions/default.aspx>

Microsoft®, Internet Explorer®, Outlook®, SkyDrive®, Windows Vista®, Zune®, Xbox 360®, DirectX®, Windows Server® and Windows® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

Terraform language, console, expressions  
and more..

# Terraform Language

- Terraform uses its own configuration language, designed to allow concise descriptions of infrastructure.
- The Terraform language is declarative, describing an intended goal rather than the steps to reach that goal.

## Expressions

- Expressions are used to refer to or compute values within a configuration.
  - Simplest example are literals such as *"hello"* or *5*
- Terraform language also allows more complex expressions such as references to data exported by resources, arithmetic, conditional evaluation, and a number of built-in functions.

# Terraform comes with an interactive console for evaluating expressions

```
vsonline:~/workspace/repos/tfw$ terraform console
```

```
> "hello"
```

```
hello
```

```
>
```

```
> █
```



# Arithmetic and Logical Operators

An *operator* is a type of expression that transforms or combines one or more other expressions.

The Terraform language has a set of operators for both arithmetic and logic, which are like operators in programming languages such as JavaScript or Ruby.

```
vsonline:~/workspace/repos/tfw$ terraform console
> 100 / 3
33.333333333333336
>

> 1 > 2
false
>

> 10 > 5 ? "true value" : "false value"
true value
>
```

# Type Constraints and values

```
variable "region" {  
  type = string  
  default = "UK South"  
}  
  
locals {  
  tags = {  
    "cost_center" = "contoso research"  
  }  
}
```

## Supported type keywords

string

number

bool

Allow you to restrict the type of value that will be accepted as the value for a variable.

If no type constraint is set then a value of any type is accepted.

## Collections and complex-types via type constructors

list(<TYPE>)

set(<TYPE>)

map(<TYPE>)

object({<ATTR NAME> = <TYPE>, ... })

tuple([<TYPE>, ...])

# count and for\_each meta arguments (keeping it DRY)

Only use count  
for binary  
operations. I.e. 0  
or 1 resources

```
resource "azurerm_resource_group" "rg" {  
  count      = var.create_resource_group ? 1 : 0  
  name      = var.rsesource_group_name  
  location  = var.region  
  tags      = var.tags  
}
```

Use for\_each for  
0 or many  
resources

```
resource "azurerm_resource_group" "rg" {  
  for_each  = var.resource_groups  
  name      = each.value.name  
  location  = var.region  
  tags      = var.tags  
}
```

*Note: A given resource block cannot use both `count` and `for\_each`. We'll see other meta arguments later*



## Why only use count for binary 0 or 1 resources?

```
# variables.tf
variable rg_names {
    type = list(string)
}

# terraform.tfvars
rg_names = [
    "contoso_rg",
    "contoso_dev_rg",
    "contoso_staging_rg"
]

# main.tf
resource "azurerm_resource_group" "rgs" {
    count      = length(var.rg_names)
    name      = var.rg_names[count.index]
    location  = var.region
    tags      = var.tags
}

# outputs.tf
output "rg_ids" {
    value = azurerm_resource_group.rgs.*.id
}
```



What would happen if we removed the middle item?

# Iterating over collections

```
> [for i, s in ["hello", "world"] : upper(s)]  
[  
  "HELLO",  
  "WORLD",  
]
```



i is the 0  
based index

```
> {for k, v in {a = "hello ", b = "world "} : k =>  
  upper(v)}  
{  
  a = "HELLO"  
  b = "WORLD"  
}
```

```
output "items_ids" {  
  value = [for item in local.list_of_objects : item.id]  
}
```

## Splat expressions (for lists only)

```
output "items_ids" {  
  value = local.list_of_objects.*.id  
}
```

# Dynamic blocks

NOTE: Modern provider SDK no longer uses blocks

```
resource "azurerm_virtual_network" {
  name = "vnet-example"
  location = "West Europe"
  resource_group_name = "rg-example"
  address_space = ["10.0.0.0/16"]

  subnet {
    name = "subnet-example"
    address_prefix = "10.0.0.0/24"
  }

  subnet {
    name = "subnet-example2"
    address_prefix = "10.0.1.0/24"
  }

  subnet {
    name = "subnet-example3"
    address_prefix = "10.0.2.0/24"
  }
}
```

Hard-coded  
blocks



```
variable "subnets" {
  type = map(object({
    name = string
    address_prefix = string
  }))
  default = {
    subnet1 = {
      name = "subnet-example"
      address_prefix = "10.0.0.0/24"
    }
    subnet2 = {
      name = "subnet-example2"
      address_prefix = "10.0.1.0/24"
    }
    subnet3 = {
      name = "subnet-example3"
      address_prefix = "10.0.2.0/24"
    }
  }
}

resource "azurerm_virtual_network" {
  name = "vnet-example"
  location = "West Europe"
  resource_group_name = "rg-example"
  address_space = ["10.0.0.0/16"]

  dynamic "subnet" {
    for_each = var.subnets
    content {
      name = subnet.value.name
      address_prefix = subnet.value.address_prefix
    }
  }
}
```

Blocks  
defined in  
variable

# Interpolations and Directives

A `${ ... }` sequence is an interpolation, which evaluates the expression given between the markers, converts the result to a string if necessary, and then inserts it into the final string

```
name = "${var.prefix}_rg"
```

A `%{ ... }` sequence is a directive, which allows for conditional results and iteration over collections, similar to conditional and for expressions.

```
name = "Hello, %{ if var.name != "" }${var.name}%{ else }unnamed%{ endif }!"
```

# Terraform Import Block

The terraform import block is used to import multiple existing infrastructure components.

Requires Terraform CLI  $\geq 1.5.0$

It can also optionally generate the config for you.

```
import {  
  to = azurerm_resource_group.existing_rg  
  id = "/subscriptions/<subscription_id>/resourceGroups/rg_created_via_portal"  
}
```

```
resource "azurerm_resource_group" "existing_rg" {  
  name      = "rg_created_via_portal"  
  location = "UK South"  
}
```

```
terraform plan -generate-config-out=generated_resources.tf
```

# Terraform Moved Block

The terraform moved block is used to refactor code.

It update the key in state without destroying and recreating resources

Supports collections or individual keys

```
moved {  
  from = azurerm_resource_group.demo  
  to   = module.demo.azurerm_resource_group.demo  
}
```

# Terraform Removed Block

The terraform removed block is used to remove existing resources from state without destroying them.

```
removed {  
  from = azurerm_resource_group.existing_rg  
  lifecycle {  
    destroy = false  
  }  
}
```

# Terraform Data Sources

Data sources allow data to be fetched or computed for use elsewhere in Terraform configuration.

```
data "azurerm_resource_group" "pre_existing_rg" {  
    name = "demo_rg"  
}
```

*# Access data source*

```
data.azurerm_resource_group.pre_existing_rg.id
```




# Functions

# Functions

The Terraform language includes a number of built-in functions that you can call from within expressions to transform and combine values.

The general syntax for function calls is a function name followed by comma-separated arguments in parentheses

```
vsonline:~/workspace$ terraform console   
> min(99, 9, 999)  
9  
> merge({"dept"="research", "env"="dev"}, {"company"="contoso"})  
{  
  "company" = "contoso"  
  "dept" = "research"  
  "env" = "dev"  
}  
> exit
```

# Terraform Built-in Functions by category

Numeric

String

Collection

Encoding

File System

Date and  
Time

Hash and  
Crypto

IP Network

Type  
Conversion

# Functions on Terraform console

```
> max(12,54,3)
```

```
54
```

```
> max([12,54,3]...)
```

```
54
```

```
> split(",", "foo,bar,baz")
```

```
[
```

```
  "foo",
```

```
  "bar",
```

```
  "baz",
```

```
]
```

```
> join(" ", ["foo", "bar", "baz"])
```

```
foo bar baz
```

```
> lower("HELLO")
```

```
hello
```

```
> keys({a=1, c=2, d=3})
```

```
[
```

```
  "a",
```

```
  "c",
```

```
  "d",
```

```
]
```

```
> sort(["e", "d", "a"])
```

```
[
```

```
  "a",
```

```
  "d",
```

```
  "e"
```

```
]
```

```
> toset(["c", "b", "b"])
```

```
[
```

```
  "b",
```

```
  "c",
```

```
]
```

```
> base64decode("SGVsbG8gV29ybGQ=")
```

```
Hello World
```

```
> sha256("hello world")
```

```
b94d27b9934d3e08a52e52d7da7dabfac48  
4efe37a5380ee9088f7ace2efcde9
```

```
> cidrsubnet("172.16.0.0/16", 8, 0)
```

```
172.16.0.0/24
```

```
> file("${path.module}/hello.txt")
```

```
Hello World
```

```
> jsonencode({"hello"="world"})
```

```
{"hello":"world"}
```

*...and many more with new ones  
being added such as `yamlencode`*

# Expressions and Functions

Lab 3

Ask

Discuss

Comment

