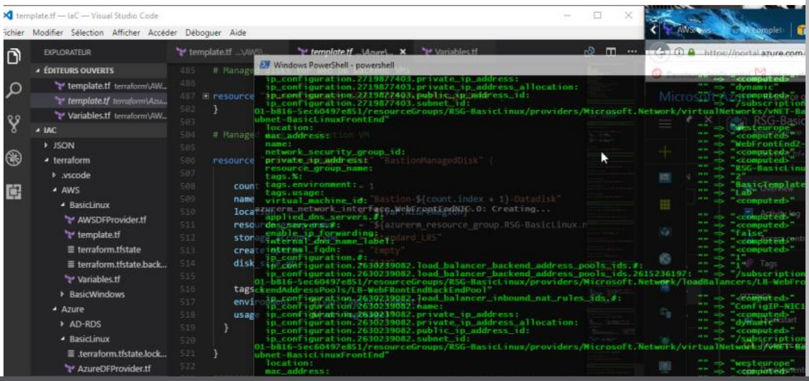


Teknews.cloud

Linking Terraform deployment and Azure Automation config through Event Grid and Logic Apps

David Frappart

07/11/2018

Contributor

Name	Title	Email
David Frappart	Cloud Architect	david@teknews.cloud

Version Control

Version	Date	State
1.0	2019/01/01	Final

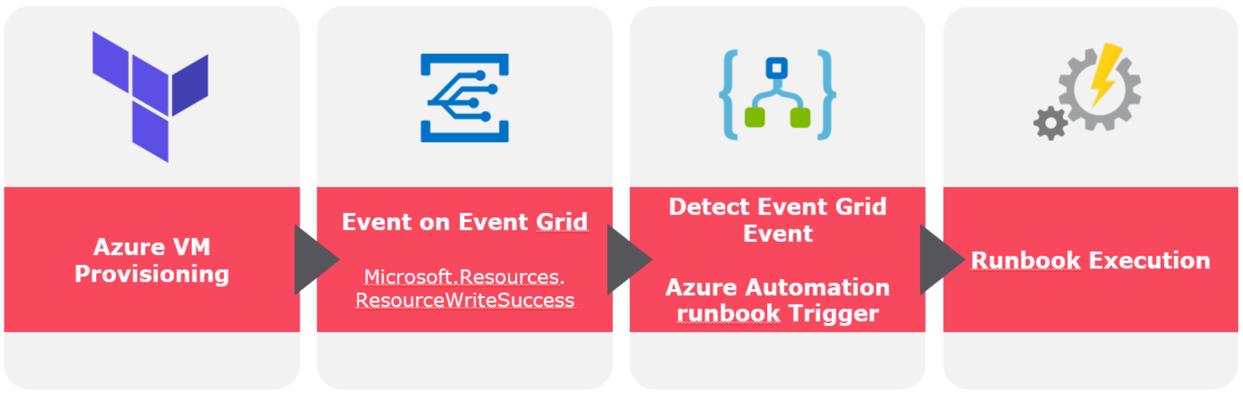
Table of Contents

1 Introduction

I've been quite busy lately so not so much article recently. Sorry about That. Now that we are in 2019, let's try to publish more and more stuff about Cloud and IaC and all the nice technical stuff that makes my jobs.
In this article, we will have a look at the Azure Logic Apps service. This article is based on the talk I had at MS Experiences18, so if you saw me there, it might be familiar. If you did not, well, where were you ? ☺

2 The use case

I came across the Logic Apps while watching a video and heard that it can be bound to Event Grid on a whole subscription. Since Event Grid track all the action on a subscription, it should also track provisioning, from any source, including Terraform.
So then, our use case becomes pretty clear. How to react to event following provisioning and launch an Azure Automation Runbook from there. Schematically, it looks like that:



3 The daily Terraform dose

The focus, for a change, is not so only on terraform. We will use mainly modules already written and available on my github repo here. But for the first time reader of my blog, let's keep a minimum of Terraform, just in case ☺
To summarize, we will launch a VM deployment on an already existing Azure VNet. We will then deploy an Azure Automation Account and credentials. Since the credentials in Azure Automation are a pair user/password, instead of provisioning password from clear text, we will rely on Azure KeyVault to secure this a little.

3.1 Terraform config for the base infrastructure

The base Azure infrastructure looks like this, with the proper variables defined and the access to the azure environment:

```
######################################################
# This file deploys the base Azure Resource
# Resource Group + vNet
######################################################

######################################################################
# Access to Azure
######################################################################

# Configure the Microsoft Azure Provider with Azure provider variable defined in
AzureDFProvider.tf

provider "azurerm" {
  subscription_id = "${var.AzureSubscriptionID1}"
  client_id       = "${var.AzureClientID}"
  client_secret   = "${var.AzureClientSecret}"
  tenant_id       = "${var.AzureTenantID}"
}

######################################################################
# Foundations resources, including ResourceGroup and vNET
######################################################################

# Creating the ResourceGroups

module "ResourceGroupInfra" {
  #Module Location
  source = "github.com/dfrappart/Terra-AZModuletest//Modules//01 ResourceGroup/"

  #Module variable
  RGName              = "${var.RGName}-
${var.EnvironmentUsageTag}${var.EnvironmentTag}-Infra"
  RGLocation          = "${var.AzureRegion}"
  EnvironmentTag      = "${var.EnvironmentTag}"
  EnvironmentUsageTag = "${var.EnvironmentUsageTag}"
}



# Creating VNet

module "VNet" {
  #Module location
  source = "github.com/dfrappart/Terra-AZModuletest//Modules//02 VNet"

  #Module variable
  vNetName            =
"${var.vNetName}${var.EnvironmentUsageTag}${var.EnvironmentTag}"
  RGName              = "${module.ResourceGroupInfra.Name}"
  vNetLocation        = "${var.AzureRegion}"
  vNetAddressSpace    = "${var.vNet1IPRange}"
  EnvironmentTag      = "${var.EnvironmentTag}"
  EnvironmentUsageTag = "${var.EnvironmentUsageTag}"
}


#Creating Storage Account for logs and Diagnostics

module "DiagStorageAccount" {
  #Module location
  source = "github.com/dfrappart/Terra-AZModuletest//Modules//03 StorageAccountGP"

  #Module variable
  StorageAccountName     = "${var.EnvironmentTag}log"
  RGName                 = "${module.ResourceGroupInfra.Name}"
  StorageAccountLocation = "${var.AzureRegion}"
  StorageAccountTier     = "${lookup(var.storageaccounttier, 0)}"
  StorageReplicationType = "${lookup(var.storagereplicationtype, 0)}"
  EnvironmentTag         = "${var.EnvironmentTag}"
  EnvironmentUsageTag    = "${var.EnvironmentUsageTag}"
}

module "LogStorageContainer" {
  #Module location
  source = "github.com/dfrappart/Terra-AZModuletest//Modules//04
StorageAccountContainer"

  #Module variable
  StorageContainerName = "logs"
  RGName               = "${module.ResourceGroupInfra.Name}"
  StorageAccountName   = "${module.DiagStorageAccount.Name}"
  AccessType           = "private"
}


module "LogLogicAppsStorageContainer" {
  #Module location
  source = "github.com/dfrappart/Terra-AZModuletest//Modules//04
StorageAccountContainer"

  #Module variable
  StorageContainerName = "logslgapps"
  RGName               = "${module.ResourceGroupInfra.Name}"
  StorageAccountName   = "${module.DiagStorageAccount.Name}"
  AccessType           = "private"
}


#Creating Storage Account for files exchange

module "FilesExchangeStorageAccount" {
  #Module location
  source = "github.com/dfrappart/Terra-AZModuletest//Modules//03 StorageAccountGP"

  #Module variable
  StorageAccountName     = "${var.EnvironmentTag}file"
  RGName                 = "${module.ResourceGroupInfra.Name}"
  StorageAccountLocation = "${var.AzureRegion}"
  StorageAccountTier     = "${lookup(var.storageaccounttier, 0)}"
  StorageReplicationType = "${lookup(var.storagereplicationtype, 0)}"
  EnvironmentTag         = "${var.EnvironmentTag}"
  EnvironmentUsageTag    = "${var.EnvironmentUsageTag}"
}

#Creating Storage Share

module "InfraFileShare" {
  #Module location
  source = "github.com/dfrappart/Terra-AZModuletest//Modules//05
StorageAccountShare"

  #Module variable
  ShareName          = "infrafileshare"
  RGName             = "${module.ResourceGroupInfra.Name}"
  StorageAccountName = "${module.FilesExchangeStorageAccount.Name}"
}


module "MSExpKeyVault" {
  #Module location

  source = "github.com/dfrappart/Terra-AZModuletest//Modules//27 Keyvault"
  #Module variables
  KeyVaultName               = "MSExpKeyVault"
  KeyVaultRG                 = "${module.ResourceGroupInfra.Name}"
  KeyVaultObjectIDPolicy2 = "${var.AzureServicePrincipalInteractive}"
  KeyVaultObjectIDPolicy1 = "${var.AzureTFSP}"
  KeyVaultTenantID           = "${var.AzureTenantID}"
  KeyVaultSKUName            = "premium"
  EnvironmentTag             = "${var.EnvironmentTag}"
  EnvironmentUsageTag        = "${var.EnvironmentUsageTag}"
}
```

3.2 Azure Automation objects creation

3.2.1 Automation Account

Terraform allows the creation of an Azure Automation Account. To do so, we use the azurerm_automation_account. The resource creation looks like this:

```
##############################################################
#This module allows the creation of an Automation Account
##############################################################


# Automation Account creation

resource "azurerm_automation_account" "TerraAutomationAccount" {
  name                          = "${var.AutomationAccountName}"
  location                      = "${var.AutomationAccountLocation}"
  resource_group_name           = "${var.RGName}"
  sku {
    name = "Basic"
  }

  tags {
    environment = "${var.EnvironmentTag}"
    usage       = "${var.EnvironmentUsageTag}"
  }
}
```

And the module call is as follow:

```
#Automation Account Creation

module "AutoconfigVMs" {

  #Module Location
  source = "github.com/dfrappart/Terra-AZModuletest//Modules//30 Automation
Account/"

  #Module variables
  AutomationAccountName         = "AutoconfigVMs"
  AutomationAccountLocation     = "${var.AzureRegion}"
  RGName                        = "${module.ResourceGroupConfigMgmt.Name}"
  EnvironmentTag                = "${var.EnvironmentTag}"
  EnvironmentUsageTag           = "${var.EnvironmentUsageTag}"

}
```

3.2.2 Automation credentials

When creating an Automation account in the portal, it comes with a lot of others object, including the creation of a service principal in Azure AD, with somewhat elevated privilege. This is nice because it's easy, but not so secure. Fortunately, through Terraform, we only get the Azure Automation Account object. But the absence of the SP makes it necessary to either add it manually or to use credentials with appropriate right on the subscription. Thus the use of azurerm_automation_credential. The resource creation is pretty straightforward:

```
##############################################################
#This module allows the creation of Automation Account
#credentials
##############################################################


# Automation Credentials Creation

resource "azurerm_automation_credential" "TerraAutomationCreds" {
  name                            = "${var.AutomationCredsName}"
  resource_group_name             = "${var.RGName}"
  account_name                    = "${var.AutomationAccountName}"
  username                        = "${var.AutoCredsUserName}"
  password                              = "${var.AutoCredsPwd}"
  description                           = "${var.AutoCredsDescription}"

}
```

And the module call is as follow:

```
#Automation Creds Creation

module "Automationrunas" {

  #Module Location
  source = "github.com/dfrappart/Terra-AZModuletest//Modules//31 Automation
Automation Creds/"

  #Module variables
  AutomationCredsName       = "Automationrunas"
  AutomationAccountName     = "${module.AutoconfigVMs.Name}"
  RGName                    = "${module.ResourceGroupConfigMgmt.Name}"
  AutoCredsUserName         = "user@mydomain.info"
  AutoCredsPwd              =
"${data.azurerm_key_vault_secret.TerraCreatedAutomationSecret.value}"

}
```

Note the value to the AutoCredsPwd, which point to a secret in a keyvault.

3.2.3 Automation DSC

One of the interesting aspect of Azure Automation is its capability to provide PowerShell Desired State features, without the need of a server. We can push DSC config to an Azure Automation account with the resource azurerm_automation_dsc_configuration. The module creating the resource would look like this:

```
##############################################################
#This module allows the creation of Automation DSC Config
##############################################################

#Template for the DSC config file
data "template_file" "DSCConfigTemplate" {
  template = "${file("${path.root}${var.SettingsTemplatePath}")}"
}

# Automation Credentials Creation
```

```
resource "azurerm_automation_dsc_configuration" "TerraAutomationDSCConfig" {
  name                                = "${var.AutomationDSCConfigName}"
  resource_group_name                 = "${var.RGName}"
  automation_account_name             = "${var.AutomationAccountName}"
  location                            = "${var.Location}"
  content_embedded                    =
"${data.template_file.DSCConfigTemplate.rendered}"
  description                         = "${var.AutomationDSCConfigDescription}"
  log_verbose                         = "${var.LogverboseEnabledStatus}"

}
```

While the module call would be like this:

```
module "DSCConfig" {

  #Module Location
  source = "github.com/dfrappart/Terra-AZModuletest//Modules//32 Automation DSC
Config/"

  #Module variables
  AutomationDSCConfigName      = "DSCConfig"
  AutomationAccountName            = "${module.AutoconfigVMs.Name}"
  RGName                           = "${module.ResourceGroupConfigMgmt.Name}"
  Location                         = "${var.AzureRegion}"
  SettingsTemplatePath             = "./Template/DSCConfig.tpl"

}
```

With an example DSC config like this:

```
configuration DSCConfig
{
    Node IsWebServer
    {
        WindowsFeature IIS
        {
            Ensure               = 'Present'
            Name                 = 'Web-Server'
            IncludeAllSubFeature = $true
        }
    }
    Node NotWebServer
    {
        WindowsFeature IIS
        {
            Ensure                  = 'Absent'
            Name                    = 'Web-Server'
        }
    }
}
```

After this resource creation, there are other step to perform, such as compiling the DSC config and registering nodes for the DSC config. We will have a look at how it looks like in the portal.

3.2.4 Automation runbook

While the automation account is a container for all automation related object, the object in which the automation intelligence is living is the runbook. Creating a runbook involves mainly coding the automation steps in PowerShell, or Python. However, the code for configuration still requires to be embedded in the runbook. In case we have already an automation PowerShell code, we can use the terraform resource azurerm_automation_runbook:

```
##############################################################
#This module allows the creation of Automation runbook
##############################################################

#File for the Runbook
data "local_file" "AutomationRunBook" {
  filename = "${path.root}${var.SettingsFilePath}"
}

# Runbook creation

resource "azurerm_automation_runbook" "TerraAutomationRunbook" {
  name                               = "${var.AutomationRunbookName}"
  location                           = "${var.Location}"
  resource_group_name                = "${var.RGName}"
  account_name                       = "${var.AutomationAccountName}"
  log_verbose                        = "${var.LogVerboseActivated}"
  log_progress                       = "${var.LogProgressActivated}"
  description                        = "${var.RunbookDesc}"
  runbook_type                       = "${var.RunbookType}"
  publish_content_link {
    uri       = "https://raw.githubusercontent.com/Azure/azure-quickstart-
templates/master/101-automation-runbook-getvms/Runbooks/Get-AzureVMTutorial.ps1"

  }
  content    = "${data.local_file.AutomationRunBook.content}"

}
```

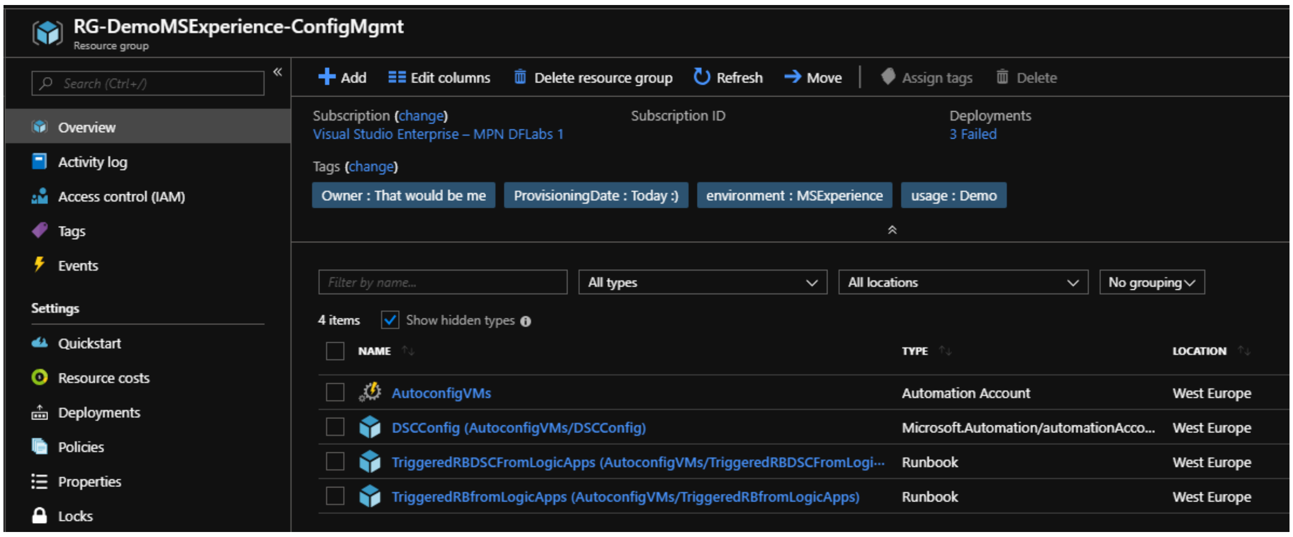
A remarkable point is the fact that an uri for a remote location is mandatory in the object. To bypass the remote script, we then use the parameter content which refer to a local file. We use the data source local_file to pass the path of a local file as displayed in the code sample.
Then we call the module as follow:

```
#Runbook creation

module "RunbookTriggeredfromLogicApps" {

  #Module location
  source = "github.com/dfrappart/Terra-AZModuletest//Modules//34 Automation
Runbook/"

  #Module variables
  AutomationRunbookName = "TriggeredRBfromLogicApps"
  Location              = "${var.AzureRegion}"
  RGName                = "${module.ResourceGroupConfigMgmt.Name}"
  AutomationAccountName = "${module.AutoconfigVMs.Name}"
  SettingsFilePath      = "./Runbook/TriggeredRBfromLogicApps.ps1"
}
```

In this case, the automation script is in a local folder under ./Runbook/.

4 Azure Automation Account, Runbook, Credentials and Desired State – A look at the portal

Now let’s have a look at what we created earlier. On the portal, under our resource group, we have an azure automation account and 2 runbooks and a DSC Config:

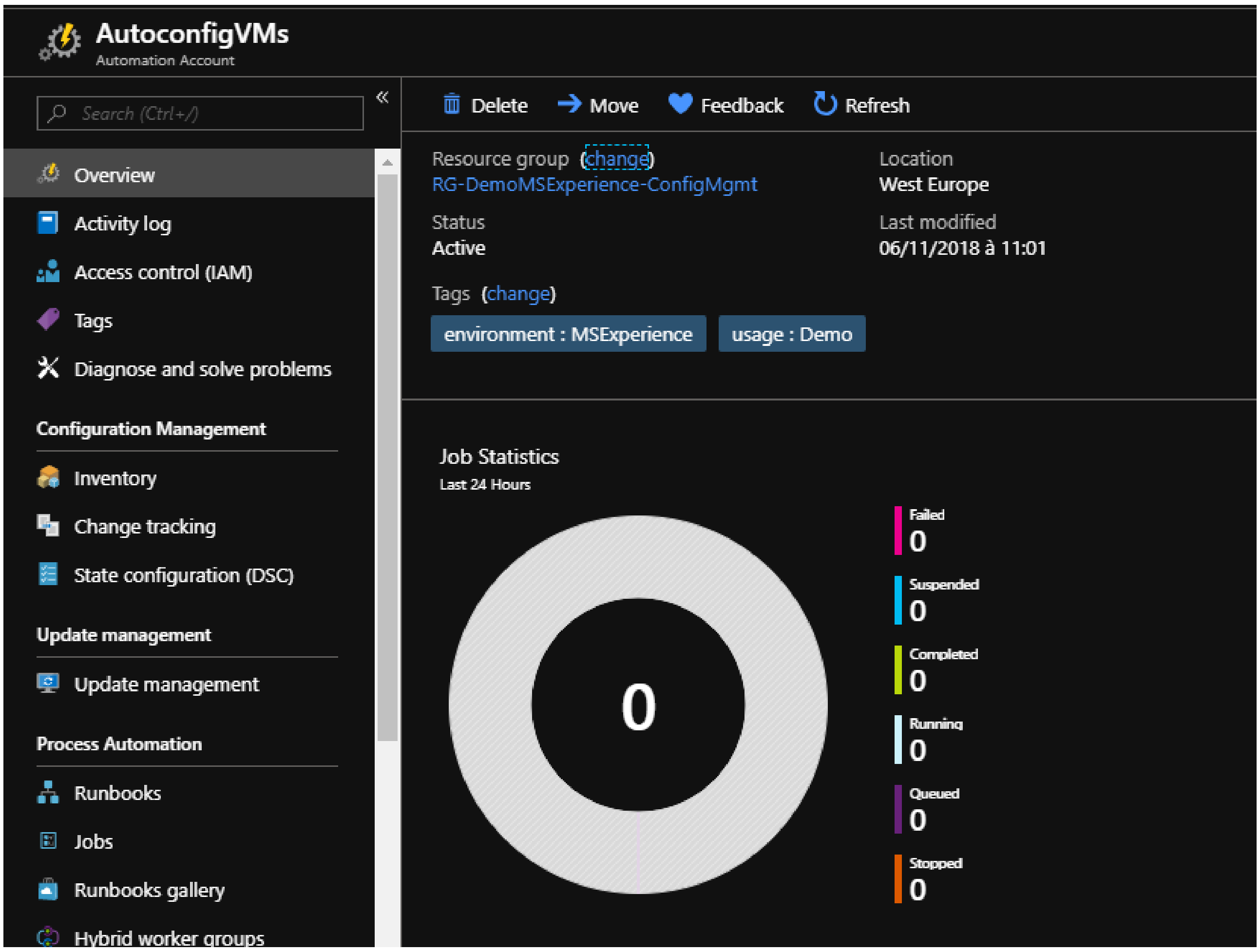

RG-DemoMSExperience-ConfigMgmt
Resource group

Subscription (change): Visual Studio Enterprise – MPN DFLabs 1
Subscription ID
Deployments: 3 Failed

Tags (change): Owner : That would be me | ProvisioningDate : Today :) | environment : MSExperience | usage : Demo

4 items

NAME	TYPE	LOCATION
AutoconfigVMs	Automation Account	West Europe
DSCConfig (AutoconfigVMs/DSCConfig)	Microsoft.Automation/automationAcco...	West Europe
TriggeredRBDSCFromLogicApps (AutoconfigVMs/TriggeredRBDSCFromLogi...	Runbook	West Europe
TriggeredRBfromLogicApps (AutoconfigVMs/TriggeredRBfromLogicApps)	Runbook	West Europe


AutoconfigVMs
Automation Account

Resource group (change): RG-DemoMSExperience-ConfigMgmt
Location: West Europe
Status: Active
Last modified: 06/11/2018 à 11:01
Tags (change): environment : MSExperience | usage : Demo

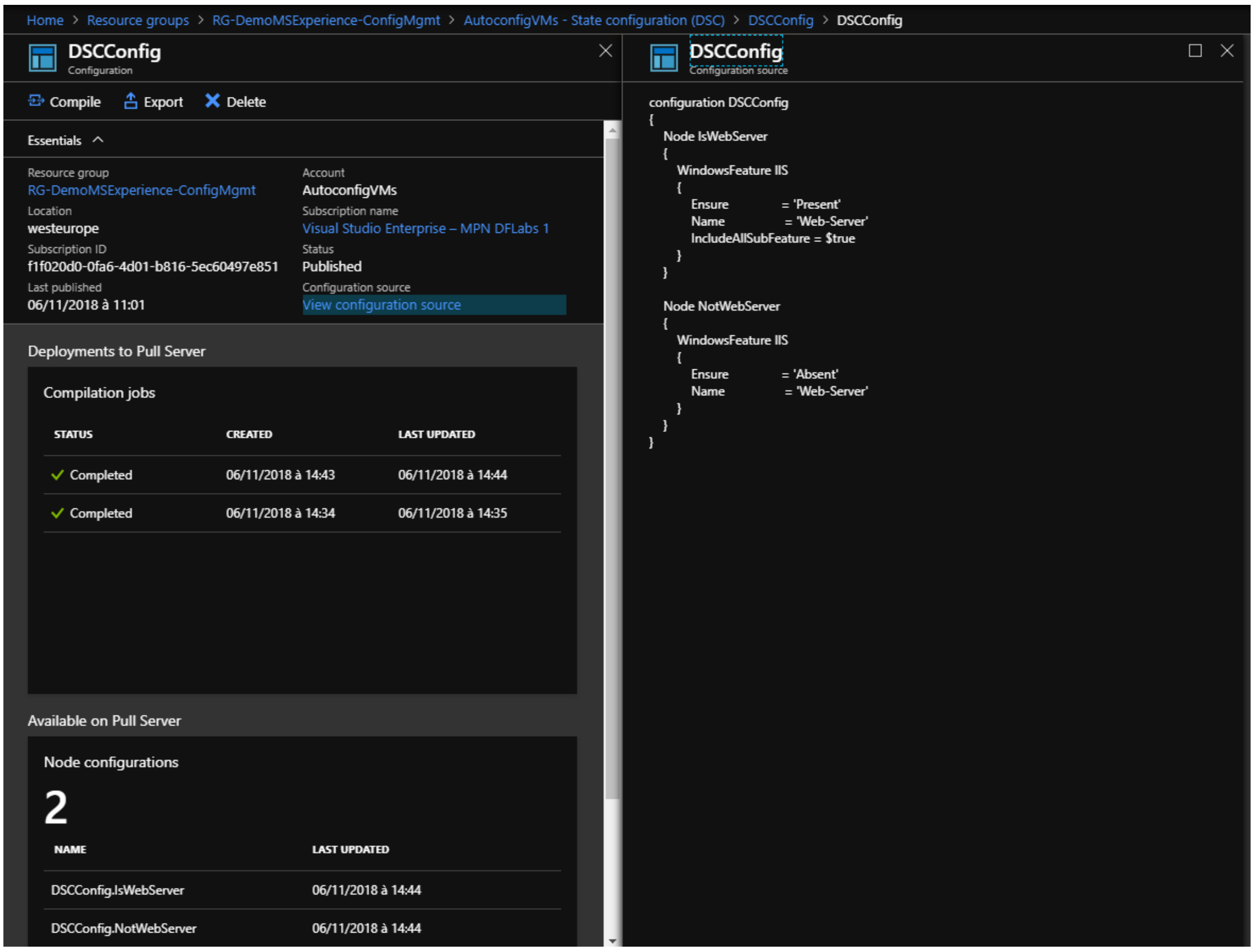
Job Statistics
Last 24 Hours

Failed	0
Suspended	0
Completed	0
Running	0
Queued	0
Stopped	0

In the Automation account, we find the runbooks the credentials and the DSC Config. Let's have a look at the DSC Config

4.1 The DSC Config in the portal

Following the blades to
Home\Resource groups\RG-Name\AutoconfigVMs - State configuration (DSC)\DSCConfig
We find the following screen:


Home > Resource groups > RG-DemoMSExperience-ConfigMgmt > AutoconfigVMs - State configuration (DSC) > DSCConfig > DSCConfig
DSCConfig
Configuration
Compile
Export
Delete
Essentials
DSCConfig
Configuration source

Resource group	RG-DemoMSExperience-ConfigMgmt
Location	westeurope
Subscription ID	f1f020d0-0fa6-4d01-b816-5ec60497e851
Last published	06/11/2018 à 11:01
Account	AutoconfigVMs
Subscription name	Visual Studio Enterprise – MPN DFLabs 1
Status	Published
Configuration source	View configuration source

Deployments to Pull Server

Compilation jobs

STATUS	CREATED	LAST UPDATED
✓ Completed	06/11/2018 à 14:43	06/11/2018 à 14:44
✓ Completed	06/11/2018 à 14:34	06/11/2018 à 14:35

Available on Pull Server

Node configurations

2

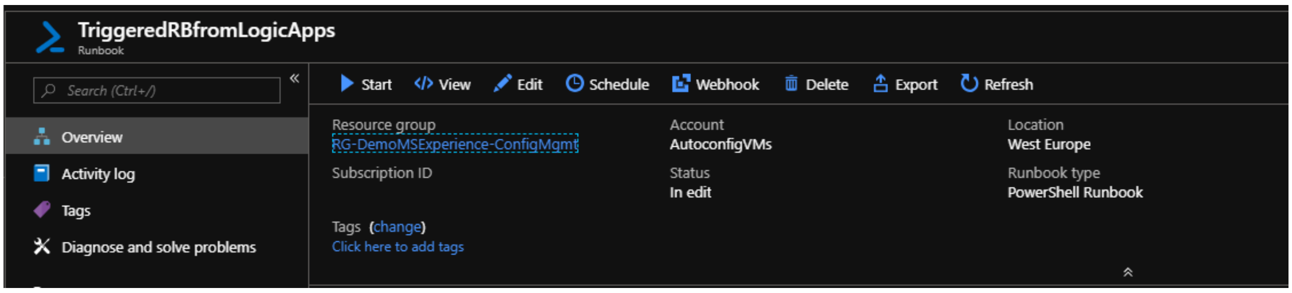
NAME	LAST UPDATED
DSCConfig.IsWebServer	06/11/2018 à 14:44
DSCConfig.NotWebServer	06/11/2018 à 14:44

```
configuration DSCConfig
{
    Node IsWebServer
    {
        WindowsFeature IIS
        {
            Ensure          = 'Present'
            Name            = 'Web-Server'
            IncludeAllSubFeature = $true
        }
    }

    Node NotWebServer
    {
        WindowsFeature IIS
        {
            Ensure          = 'Absent'
            Name            = 'Web-Server'
        }
    }
}
```

This configuration is already complied. But a fresh provisioning will require to perform this action, either in the portal or in PowerShell

4.2 The runbook in the portal

Going in the runbook section we can find the provisioned runbook, and the PowerShell code it should run:



Resource group
RG-DemoMSExperience-ConfigMgmt

Account
AutoconfigVMs

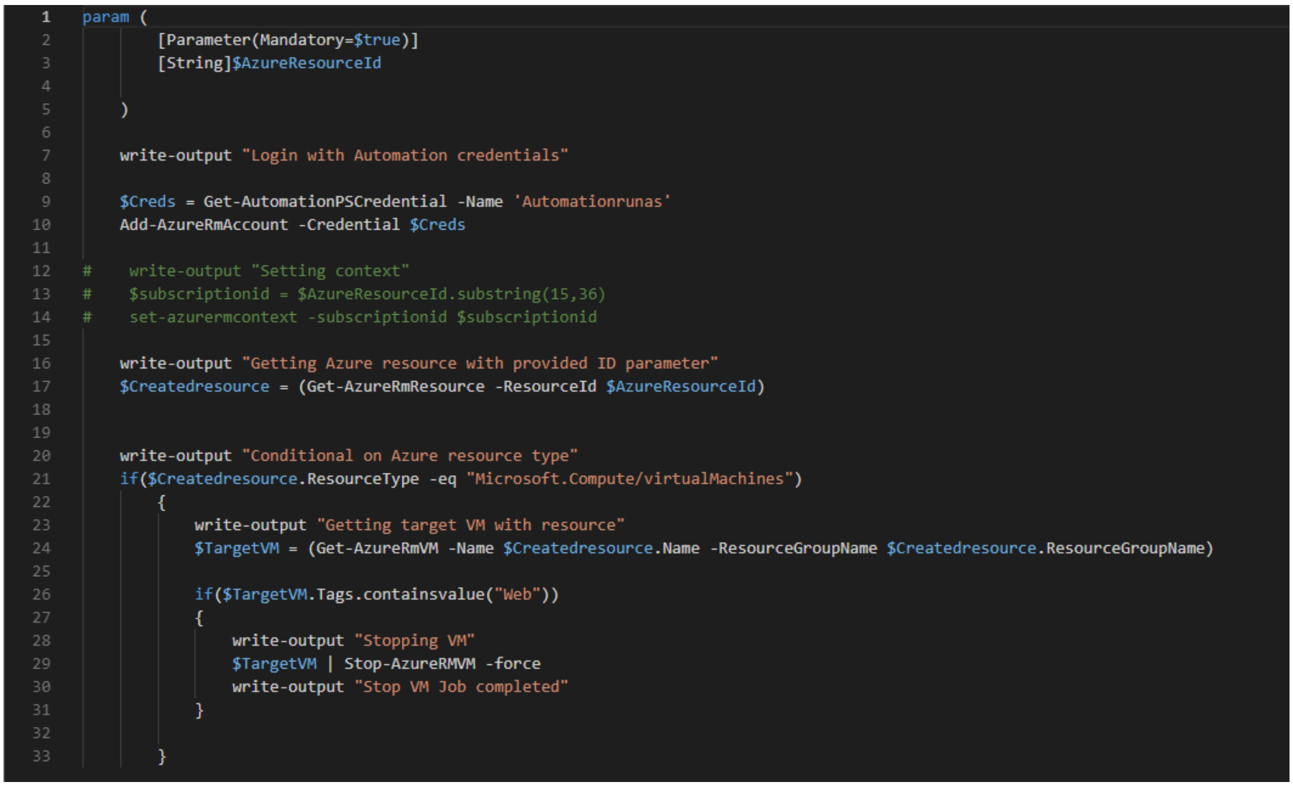
Location
West Europe

Subscription ID

Status
In edit

Runbook type
PowerShell Runbook

Tags (change)
Click here to add tags



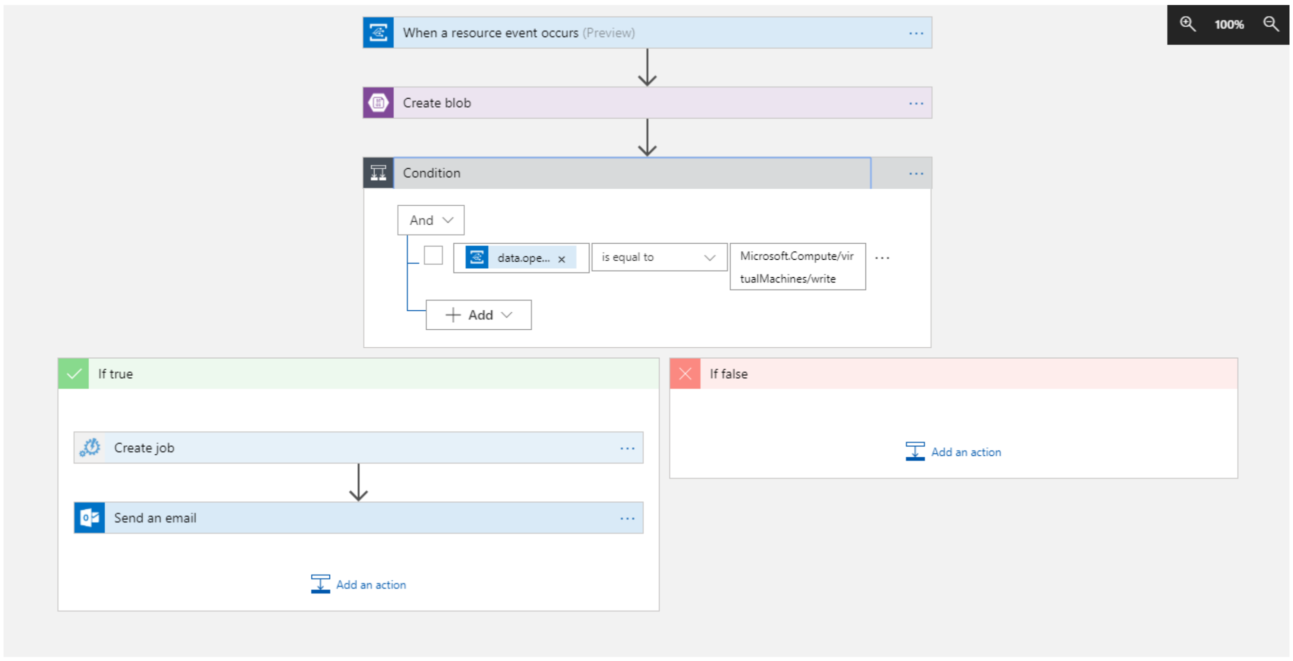
```
param (
        [Parameter(Mandatory=$true)]
        [String]$AzureResourceId

    )

    write-output "Login with Automation credentials"

    $Creds = Get-AutomationPSCredential -Name 'Automationrunas'
    Add-AzureRmAccount -Credential $Creds

#    write-output "Setting context"
#    $subscriptionid = $AzureResourceId.substring(15,36)
#    set-azurermcontext -subscriptionid $subscriptionid

    write-output "Getting Azure resource with provided ID parameter"
    $Createdresource = (Get-AzureRmResource -ResourceId $AzureResourceId)


    write-output "Conditional on Azure resource type"
    if($Createdresource.ResourceType -eq "Microsoft.Compute/virtualMachines")
        {
            write-output "Getting target VM with resource"
            $TargetVM = (Get-AzureRmVM -Name $Createdresource.Name -ResourceGroupName $Createdresource.ResourceGroupName)

            if($TargetVM.Tags.containsvalue("Web"))
            {
                write-output "Stopping VM"
                $TargetVM | Stop-AzureRMVM -force
                write-output "Stop VM Job completed"
            }

        }
```

5 Creating the logic app

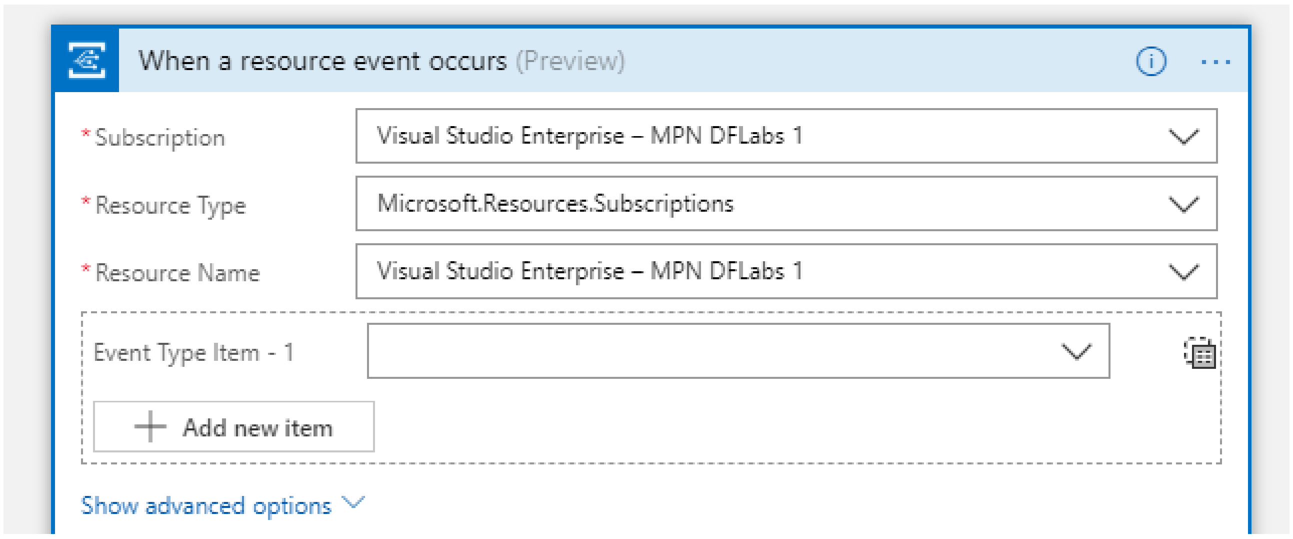
At this point, we have the terraform code, which is kind of the familiar part and some PowerShell config through Azure Automation. To chain all this, we will use Azure Logic App

5.1 The logic Apps designer

Logic Apps are available in the portal, by searching Logic Apps. Once in the appropriate place, we are able to create a Logic Apps through the Designer, as follow:



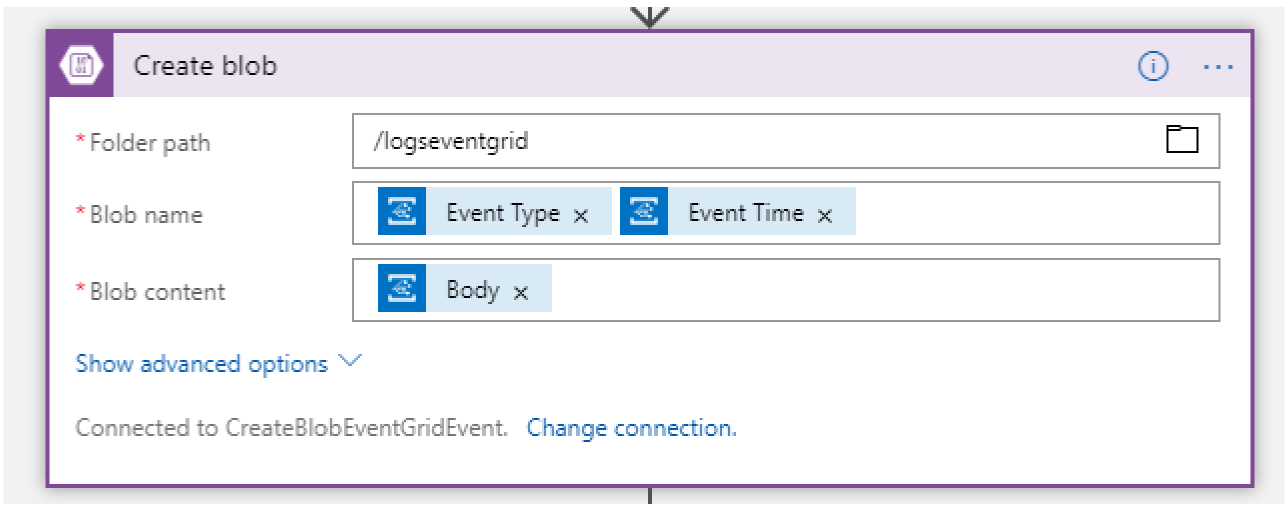
This logic app reacts to an event from Azure Event Grid. We just need to specify the target subscription and the kind of resource we ant to monitor. In this case, the resource is the full subscription, so the resource type match the value Microsoft.Resources.Subscriptions.



When a resource event occurs (Preview)

* Subscription	Visual Studio Enterprise – MPN DFLabs 1
* Resource Type	Microsoft.Resources.Subscriptions
* Resource Name	Visual Studio Enterprise – MPN DFLabs 1
Event Type Item - 1	

Since I like to track what I do, the first step is to create a blob for all events occurring on the subscription:

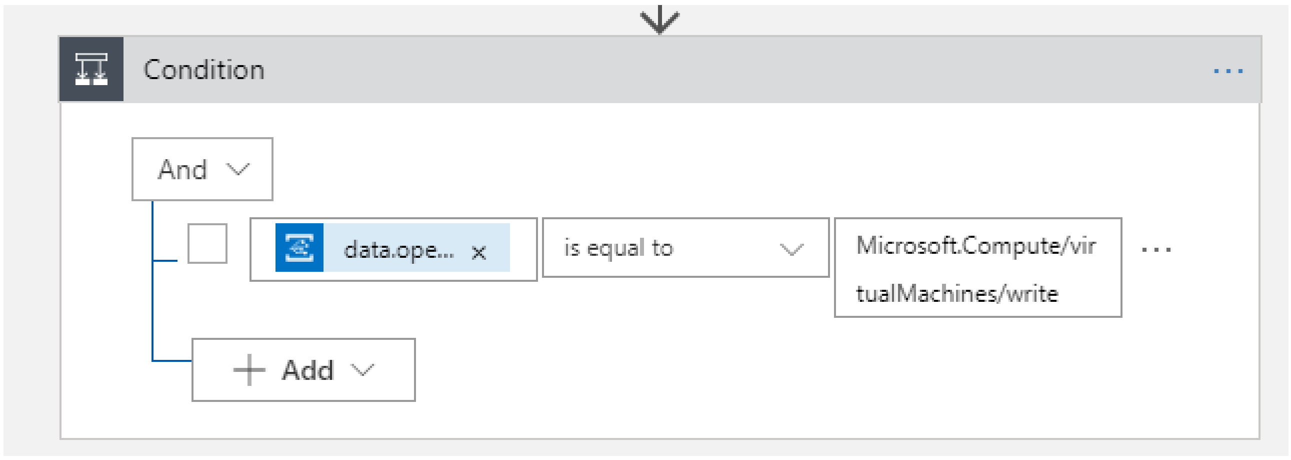

Create blob

*Folder path: /logseventgrid
*Blob name: Event Type × Event Time ×
*Blob content: Body ×

Show advanced options

Connected to CreateBlobEventGridEvent. Change connection.

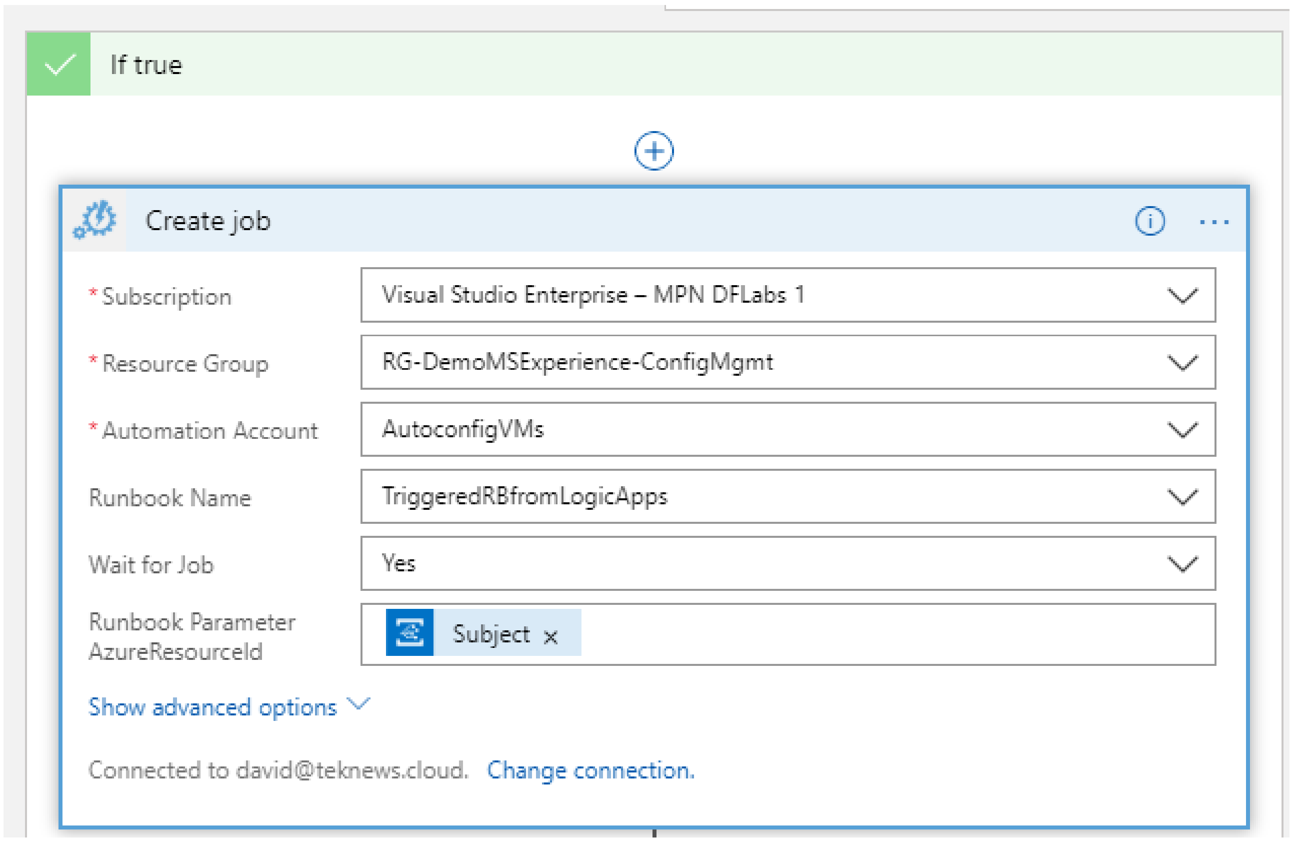
Then we add a condition to filter only the kind of event that match a virtual machine write:


Condition

And | data.ope... × | is equal to | Microsoft.Compute/virtualMachines/write

Add

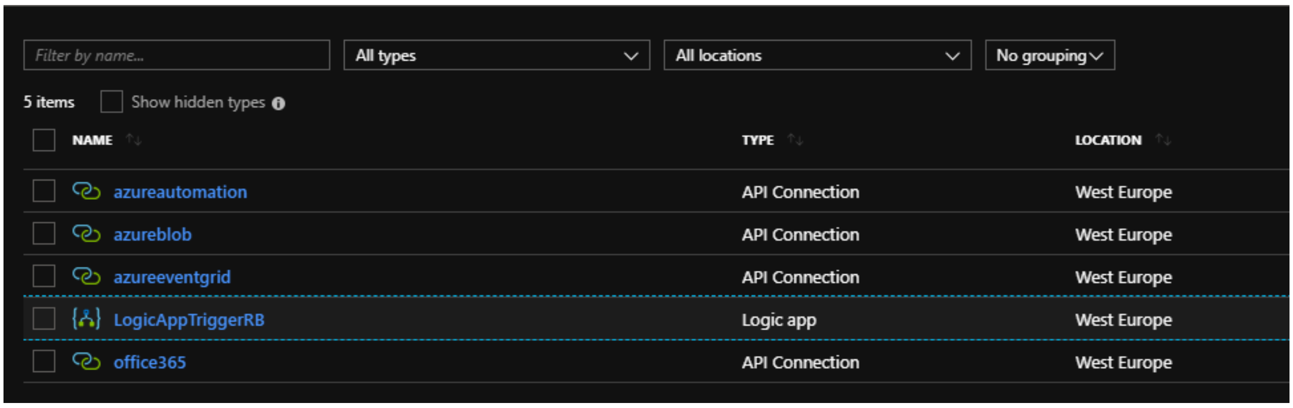
On condition matched, we launch an Azure Automation runbook:



* Subscription	Visual Studio Enterprise – MPN DFLabs 1
* Resource Group	RG-DemoMSExperience-ConfigMgmt
* Automation Account	AutoconfigVMs
Runbook Name	TriggeredRBfromLogicApps
Wait for Job	Yes
Runbook Parameter AzureResourceId	Subject

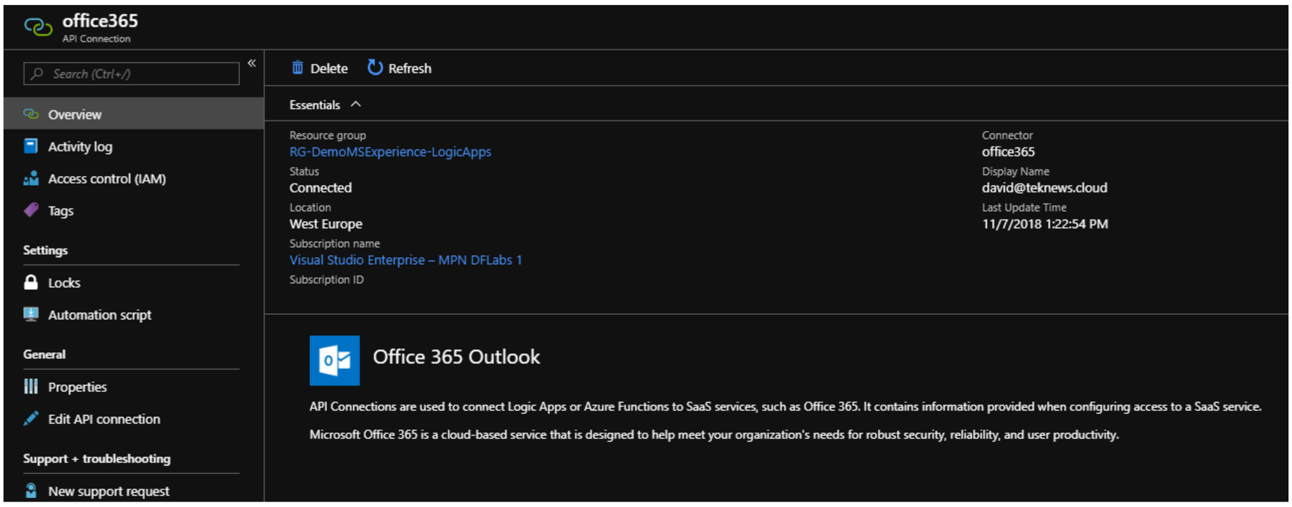
Connected to david@teknews.cloud. Change connection.

As a parameter, since we defined the runbook to require an Azure Resource Id, we need to provide one. Fortunately, the Subject of the event is the Azure Resource Id on which the event is occurring.
That's all for the high level overview. You may wonder how are specified the value for the blobs creation or for the conditional. That's a good question and it's what we're going to see in the next section. But just before that, for those wondering, no Azure Event Grid is required before manipulating a logic app referring to an event grid. As a matter of fact, The Logic Apps designer creates automatically the interactions with other Azure / Microsoft services when we put the objects in the designer. After adding some object, we can see in the logic apps section the associated API connections:



NAME	TYPE	LOCATION
azureautomation	API Connection	West Europe
azureblob	API Connection	West Europe
azureeventgrid	API Connection	West Europe
LogicAppTriggerRB	Logic app	West Europe
office365	API Connection	West Europe

Looking at the object for Office365, we have the following:


office365
API Connection

Resource group: RG-DemoMSExperience-LogicApps
Status: Connected
Location: West Europe
Subscription name: Visual Studio Enterprise – MPN DFLabs 1
Subscription ID

Connector: office365
Display Name: david@teknews.cloud
Last Update Time: 11/7/2018 1:22:54 PM

Office 365 Outlook

API Connections are used to connect Logic Apps or Azure Functions to SaaS services, such as Office 365. It contains information provided when configuring access to a SaaS service.

Microsoft Office 365 is a cloud-based service that is designed to help meet your organization's needs for robust security, reliability, and user productivity.

5.2 Anatomy of an event grid event

To manipulate correctly Logic apps with event grid, we should understand of what is composed an event grid event.
It is JSON based, as usual, and it contains a subject, a type, a time stamp and an id. It also contains a topic and last but not least, it contains as all event a data section.
The subject is the resource id of the Azure resource which is the subject tof the event. The topic, refers to the mntored subscription. In the event data, the property operationName is our filtering tool. Below is an event for a resource group creation:

```
{
    "subject":"/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourcegroups/RG-DemoMSExperience-VMs",
    "eventType":"Microsoft.Resources.ResourceWriteSuccess",
    "eventTime":"2018-11-06T15:50:00.7560408Z",
    "id":"708fd14c-9b64-4aa9-9e86-d349e4c70cb0",
    "data":
        {
            "authorization":
                {
                    "scope":"/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourcegroups/RG-DemoMSExperience-VMs",

"action":"Microsoft.Resources/subscriptions/resourcegroups/write",
                    "evidence":
                        {
                            "role":"Owner",
                            "roleAssignmentScope":"/subscriptions/xxxxxxxx-xxxx-
xxxx-xxxx-xxxxxxxxxxxx",
                            "roleAssignmentId":"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
                            "roleDefinitionId":"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
                            "principalId":"xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
                            "principalType":"ServicePrincipal"
                        }
                },
            "claims":
                {
                    "aud":"https://management.azure.com/",
                    "iss":"https://sts.windows.net/e0c45235-95fe-4bd6-96ca-
2d529f0ebde4/",
                    "iat":"1541519097",
                    "nbf":"1541519097",
                    "exp":"1541522997",
                    "aio":"42RgYMhPLvzCHHHTwGrrmYA5j6b5AwA=",
                    "appid":"xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
                    "appidacr":"1",

"http://schemas.microsoft.com/identity/claims/identityprovider":"https://sts.window
s.net/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/",

"http://schemas.microsoft.com/identity/claims/objectidentifier":"xxxxxxxx-xxxx-
xxxx-xxxx-xxxxxxxxxxxx",

"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier":"xxxxxxxx-
xxxx-xxxx-xxxx-xxxxxxxxxxxx",

"http://schemas.microsoft.com/identity/claims/tenantid":"xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx",
                    "uti":"IDb8vmfxbUO7UQFBVI4oAA",
                    "ver":"1.0"
                },
            "correlationId":"xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
            "httpRequest":
                {
                    "clientRequestId":"",
                    "clientIpAddress":"193.108.21.251",
                    "method":"PUT",
                    "url":"https://management.azure.com/subscriptions/xxxxxxxx-
xxxx-xxxx-xxxx-xxxxxxxxxxxx/resourcegroups/RG-DemoMSExperience-VMs?api-
version=2017-05-10"
```

```
                },
            "resourceProvider":"Microsoft.Resources",
            "resourceUri":"/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourcegroups/RG-DemoMSExperience-VMs",

"operationName":"Microsoft.Resources/subscriptions/resourcegroups/write",
            "status":"Succeeded",
            "subscriptionId":"xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
            "tenantId":"xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
        },
    "dataVersion":"2",
    "metadataVersion":"1",
    "topic":"/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
}
```

5.3 Customizing the Logic Apps in the code view

Now that we have some more information on the event, let’s go back to the Logic Apps, and more precisely on the code view. It displays a JSON view of the logic apps. We can find in the code 2 main sections, the connections and the definition:

```
{
    "$connections": {…
    },
    "definition": {…
    }
}
```

In the connections section, we can see the connections referring to the different elements of the logic app we created, here, the connections to Azure Automation, Azure storage and Office 365:

```
    "$connections": {
        "value": {
            "azureautomation": {
                "connectionId": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/RG-DemoMSExperience-
LogicApps/providers/Microsoft.Web/connections/azureautomation",
                "connectionName": "azureautomation",
                "id": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/providers/Microsoft.Web/locations/westeurope/managedApis/azureautomati
on"
            },
            "azureblob": {
                "connectionId": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/RG-DemoMSExperience-
LogicApps/providers/Microsoft.Web/connections/azureblob",
                "connectionName": "azureblob",
                "id": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/providers/Microsoft.Web/locations/westeurope/managedApis/azureblob"
            },
            "azureeventgrid": {
                "connectionId": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/RG-DemoMSExperience-
LogicApps/providers/Microsoft.Web/connections/azureeventgrid",
                "connectionName": "azureeventgrid",
                "id": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/providers/Microsoft.Web/locations/westeurope/managedApis/azureeventgri
d"
            },
            "office365": {
                "connectionId": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/RG-DemoMSExperience-
LogicApps/providers/Microsoft.Web/connections/office365",
                "connectionName": "office365",
                "id": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/providers/Microsoft.Web/locations/westeurope/managedApis/office365"
            }
        }
    },
```

Now in the definition section we get the details on the trigger of the logic app, and of the actions that are occurring after the trigger. For the trigger, we created the logic app to launch when a resource event occur in Event Grid. It shows as follow in the code view:

```
        "triggers": {
            "When_a_resource_event_occurs": {
                "inputs": {
                    "body": {
                        "properties": {
                            "destination": {
                                "endpointType": "webhook",
                                "properties": {
                                    "endpointUrl": "@{listCallbackUrl()}"
                                }
                            },
                            "topic": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx"
```

```
                        }
                    },
                    "host": {
                        "connection": {
                            "name":
"@parameters('$connections')['azureeventgrid']['connectionId']"
                        }
                    },
                    "path": "/subscriptions/@{encodeURIComponent('xxxxxxxx-xxxx-
xxxx-xxxx-
xxxxxxxxxxxx')}/providers/@{encodeURIComponent('Microsoft.Resources.Subscriptions')
}/resource/eventSubscriptions",
                    "queries": {
                        "x-ms-api-version": "2017-06-15-preview"
                    }
                },
                "splitOn": "@triggerBody()",
                "type": "ApiConnectionWebhook"
            }
        }
```

We can see the topic referring to the Subscription we are monitoring:

```
"topic": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
```

For the actions, we have 2 mains actions, one Create_blob action and one Condition action:

```
        "actions": {
            "Condition": {…
            },
            "Create_blob": {…
            }

        },
```

In the Create_blob action, we get the details on what we put in the blob each time the logic app occurs, the method used, here a post, the folder path, here `"/logseventgrid"`, the name of the blob `"@{triggerBody()?['eventType']}@{triggerBody()?['eventTime']}"`, and the body of the blob, which is the body of the event `"body": "@{triggerBody()}"`:

```
            "Create_blob": {
                "inputs": {
                    "body": "@{triggerBody()}",
                    "host": {
                        "connection": {
                            "name":
"@parameters('$connections')['azureblob']['connectionId']"
                        }
                    },
                    "method": "post",
                    "path": "/datasets/default/files",
                    "queries": {
                        "folderPath": "/logseventgrid",
                        "name":
"@{triggerBody()?['eventType']}@{triggerBody()?['eventTime']}",
                        "queryParametersSingleEncoded": true
                    }
                },
                "runAfter": {},
                "runtimeConfiguration": {
                    "contentTransfer": {
                        "transferMode": "Chunked"
                    }
                },
                "type": "ApiConnection"
            }
```

The condition is a little trickier, we do need to specify in the code view the desired condition, here we want to act when the event match an operation on a virtual machine write. The condition in this case is as follow:

```
                "expression": {
                    "and": [
                        {
                            "equals": [
                                "@triggerBody()?['data']['operationName']",
                                "Microsoft.Compute/virtualMachines/write"
                            ]
                        }
                    ]
                },
```

As displayed, we filter event for which the operationName is equal to Microsoft.compute/virtualMachines/write.
Then we have the actions following the condition matched. The main one is the Create_job, which launch an automation runbook:

```
                    "Create_job": {
                        "inputs": {
                            "body": {
                                "properties": {
                                    "parameters": {
                                        "AzureResourceId":
"@triggerBody()?['subject']"
                                    }
                                }
                            },
                            "host": {
                                "connection": {
                                    "name":
"@parameters('$connections')['azureautomation']['connectionId']"
                                }
                            },
                            "method": "put",
                            "path": "/subscriptions/@{encodeURIComponent('xxxxxxxx-
xxxx-xxxx-xxxx-xxxxxxxxxxxx')}/resourceGroups/@{encodeURIComponent('RG-
DemoMSExperience-
ConfigMgmt')}/providers/Microsoft.Automation/automationAccounts/@{encodeURIComponen
t('AutoconfigVMs')}/jobs",
                            "queries": {
                                "runbookName": "TriggeredRBfromLogicApps",
                                "wait": true,
                                "x-ms-api-version": "2015-10-31"
                            }
                        },
                        "runAfter": {},
                        "type": "ApiConnection"
                    },
```

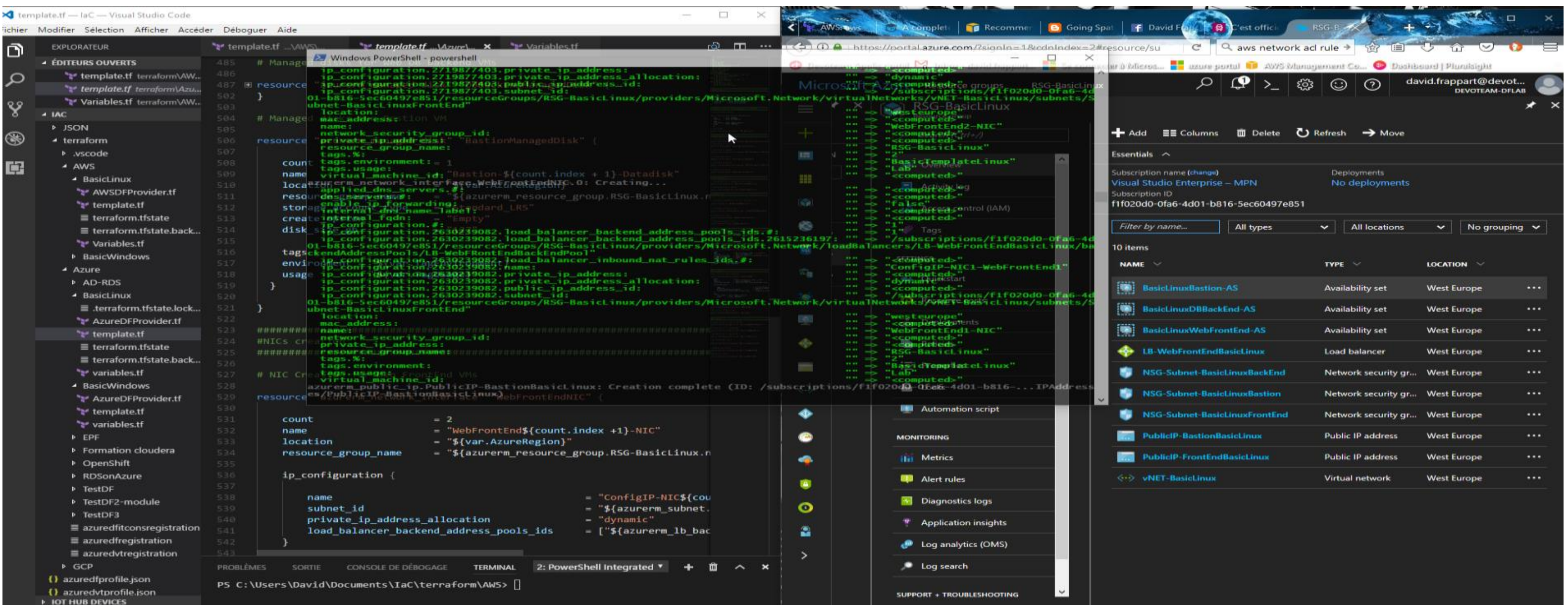
In my case, I used the code view to specify the condition value that I was looking for. After that, going back to the designer, I was able to see the condition as displayed earlier in the graphical view.

6 Testing the scenario & Conclusion

Once everything is created, we can test the scenario by provisioning a VM in our environment. It will trigger the logic app and launch the associated runbook, here a simple stop vm action.
A few though here, the condition for the logic app is to filter event matching a virtual machine write. A VM creation is a virtual machine write. However, a runbook executing on a VM is also a virtual machine write. So if we solely base our logic app condition on this filter, it will trigger after the execution of the runbook and eventually, launch again the runbook. In our case, since it is a stop vm, it will fail the second time and thus not launch again the logic app. However, measures to avoid a loop should be though about, particularly if we want to use Azure automation DSC capabilities. I confess, for now, I'm still in a testing phase so I did not look into this area. My main objective was to start playing with logic apps and find some serverless use case for ops people. Hopefully, you find that interesting. Until next time

Another Tech blog

My thoughts and experiences on Cloud related tek


```
  ip_configuration.2630239082.private_ip_address:
  ip_configuration.2630239082.private_ip_address_allocation:
  ip_configuration.2630239082.public_ip_address_id:
  ip_configuration.2630239082.subnet_id:
01-b816-5ec60497e851/resourceGroups/RSG-BasicLinux/providers/Microsoft.Network
ubnet-BasicLinuxFrontEnd"
  location:
  mac_address:
  name:
  network_security_group_id:
  private_ip_address:
  resource_group_name:
  tags.%:
  tags.environment:
  tags.usage:
  virtual_machine_id:
azurerm_public_ip.PublicIP-BastionBasicLinux: Creation complete (ID: /subscrip
es/PublicIP-BastionBasicLinux)
```

```
    count                         = 2
    name                          = "WebFrontEnd${count.index +1}-NIC"
    location                      = "${var.AzureRegion}"
    resource_group_name           = "${azurerm_resource_group.RSG-BasicLinux.n

    ip_configuration {

        name                                    = "ConfigIP-NIC${cou
        subnet_id                               = "${azurerm_subnet.
        private_ip_address_allocation           = "dynamic"
        load_balancer_backend_address_pools_ids = ["${azurerm_lb_bac
    }
```

```
PS C:\Users\David\Documents\IaC\terraform\AWS>
```

Subscription name (change)
Visual Studio Enterprise – MPN
Subscription ID
f1f020d0-0fa6-4d01-b816-5ec60497e851
Deployments
No deployments

10 items

NAME	TYPE	LOCATION
BasicLinuxBastion-AS	Availability set	West Europe
BasicLinuxDBBackEnd-AS	Availability set	West Europe
BasicLinuxWebFrontEnd-AS	Availability set	West Europe
LB-WebFrontEndBasicLinux	Load balancer	West Europe
NSG-Subnet-BasicLinuxBackEnd	Network security gr...	West Europe
NSG-Subnet-BasicLinuxBastion	Network security gr...	West Europe
NSG-Subnet-BasicLinuxFrontEnd	Network security gr...	West Europe
PublicIP-BastionBasicLinux	Public IP address	West Europe
PublicIP-FrontEndBasicLinux	Public IP address	West Europe
vNET-BasicLinux	Virtual network	West Europe