



Infra-as-Code for your Data Solution in Azure

Terraform & Azure Resource Manager (ARM)

Sjoerd Borneman

Data Engineer/Consultant at InSpark



Email:

sjoerdborneman@hotmail.com



LinkedIn:

<https://www.linkedin.com/in/sjoerdborneman/>



Twitter:

<https://twitter.com/sjoerdborneman>

Objectives Presentation

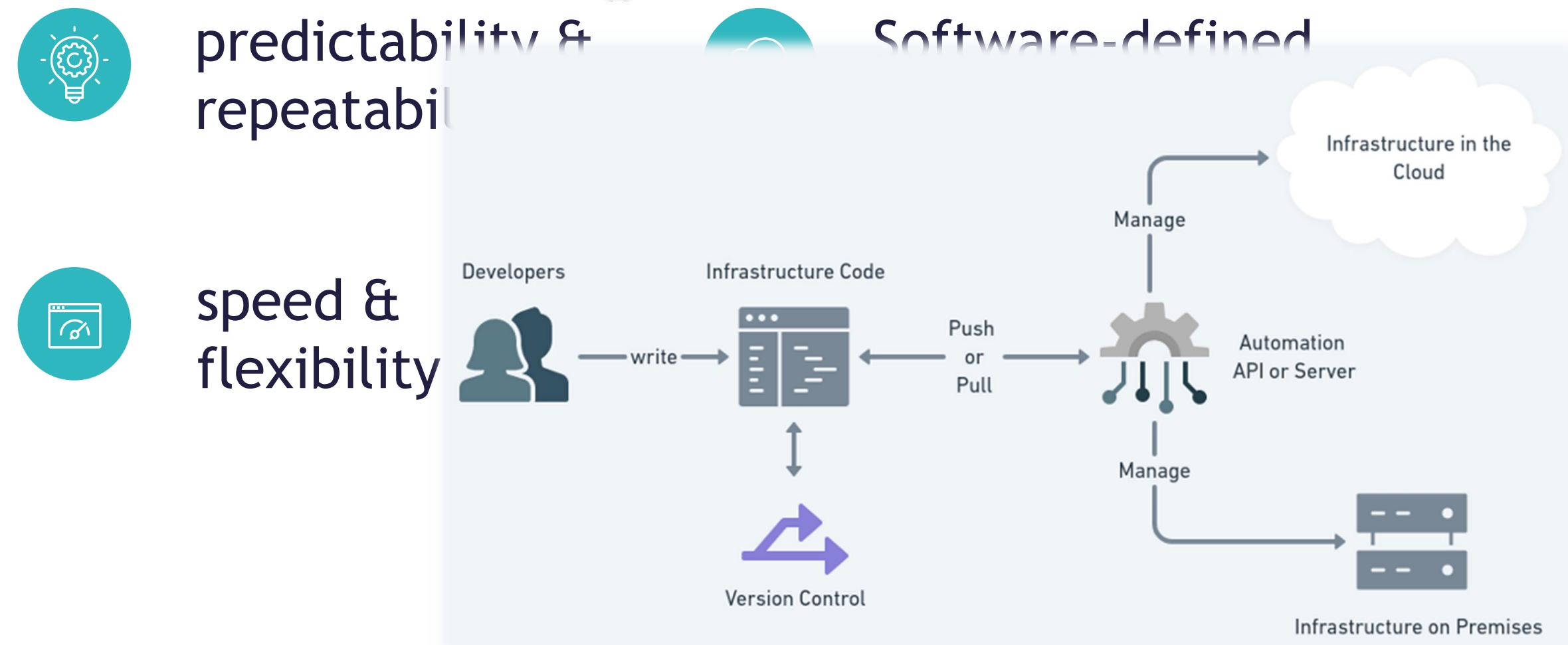
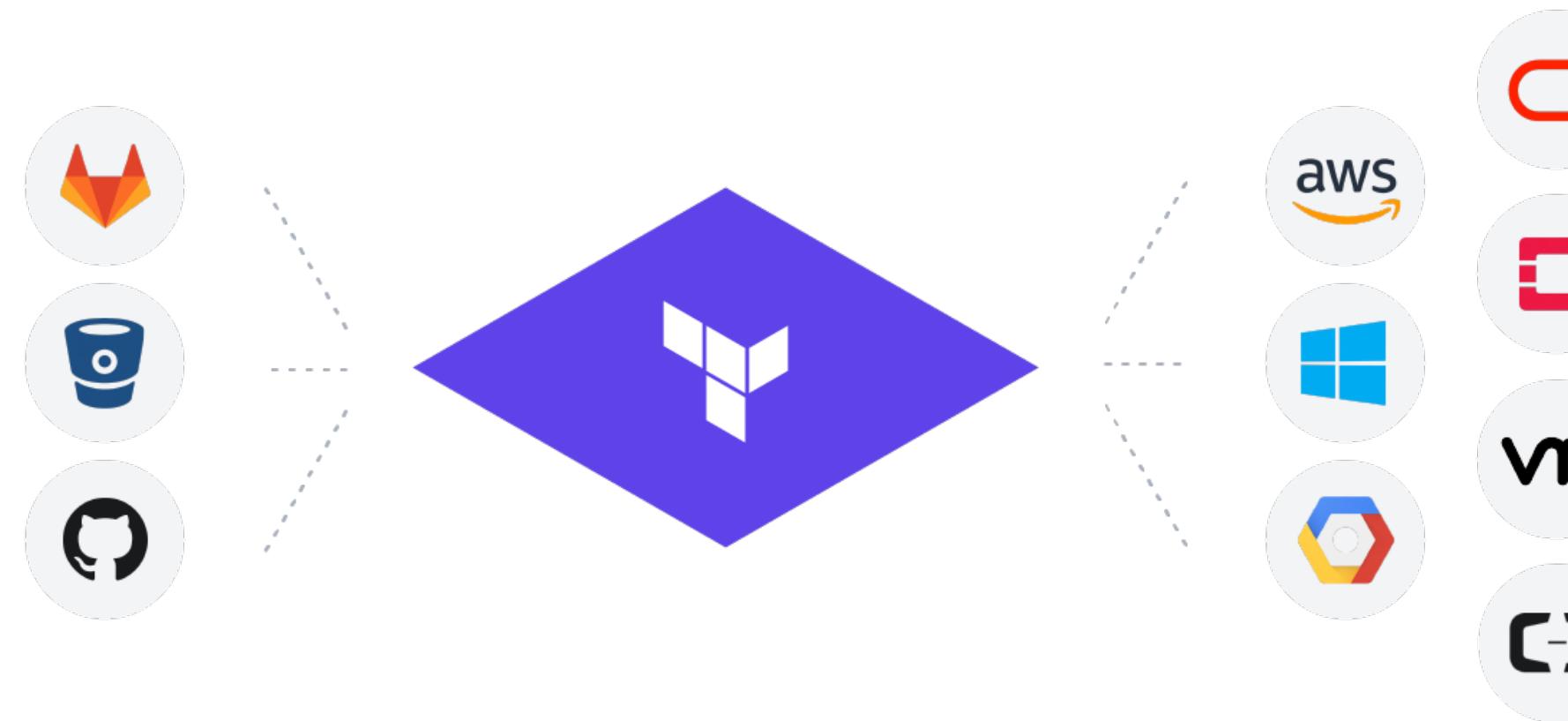
- You know what Infrastructure-as-Code (IaC) is
- Basic understanding of Terraform
- Basic understanding of Azure ARM Templates
- Knowing how to deploy an Azure Data Solution using IaC
- The use of Azure DevOps when deploying an Azure Data Solution using IaC

Goal?



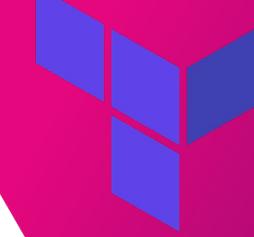
Infra-as-Code Tools

Why Infrastructure-as-Code?

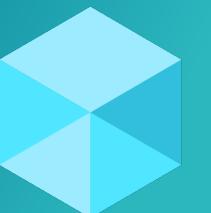


HashiCorp

Terraform



Infrastructure-as-Code Technologies



Azure Resource
Manager Templates

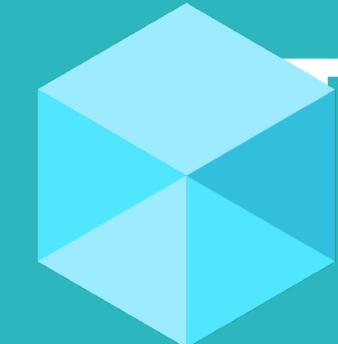
Terraform

Hashicorp

- Declarative provisioning and infrastructure orchestration tool
- Leading cloud providers
- Written in GO programming language
- Modular files
- User-friendly syntax
- Operator confidence

```
resource "azurerm_sql_server" "sqlserver" {  
    for_each           = local.all_servers  
    name               = each.value.server_name  
    resource_group_name = var.resource_groups[each.key]  
    location           = var.resource_group_location  
    version            = "12.0"  
    tags               = "Data Solution"  
  
    administrator_login      = each.value.sqlserver.admin  
    administrator_login_password = module.passwords.result[each.key]  
  
    identity {  
        type     = "SystemAssigned"  
    }  
}  
  
data "azuread_group" "sql_administrators" {  
    for_each = local.all_servers  
  
    name = each.value.sqlserver.administrator_group  
}  
  
resource "azurerm_sql_active_directory_administrator" "admins" {  
    for_each = azurerm_sql_server.sqlserver  
  
    server_name          = each.value.name  
    resource_group_name = each.value.resource_group_name  
    login                = data.azuread_group.sql_administrators[each.key].name  
    tenant_id            = var.tenant_id  
    object_id            = data.azuread_group.sql_administrators[each.key].id  
  
    depends_on = [azurerm_sql_server.sqlserver]  
}
```

Azure Resource Manager Templates



- Declarative provisioning and infrastructure orchestration tool

```
2   "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
3   "contentVersion": "1.0.0.0",
4   "parameters": {
5     "serverName": {Parameters
6       "type": "string",
7       "metadata": {
8         "description": "The name of the database server."
9       }
10    },
11    "databaseName": {
12      "type": "string",
13      "metadata": {
14        "description": "The name of the database."
15      }
16    },
17    "storageEndpoint": {
18      "type": "string",
19      "metadata": {
20        "description": "The endpoint of the datalake."
21      }
22    },
23    "emailAddresses": {
24      "type": "array"
25    }
26  }
27
28  {
29    "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
30    "contentVersion": "",
31    "apiProfile": "",
32    "parameters": {},
33    "variables": {},
34    "functions": [],
35    "resources": [],
36    "outputs": {}
37  }
38
39  {
40    "type": "Microsoft.Security/securityAlertPolicies",
41    "name": "[concat(parameters('serverName'), '/', parameters('databaseName'))]",
42    "properties": {
43      "isEnabled": true,
44      "emailSubscriptionAdmins": true,
45      "emails": "[split(parameters('emailAddresses'), ',')]"
46    }
47  },
48  {
49    "dependsOn": [
50      "[concat('Microsoft.Sql/servers/', parameters('serverName'), '/databases/', parameters('databaseName'), '/securityAlertPolicies/default')]"
51    ]
52  }
53}
54
55}
```

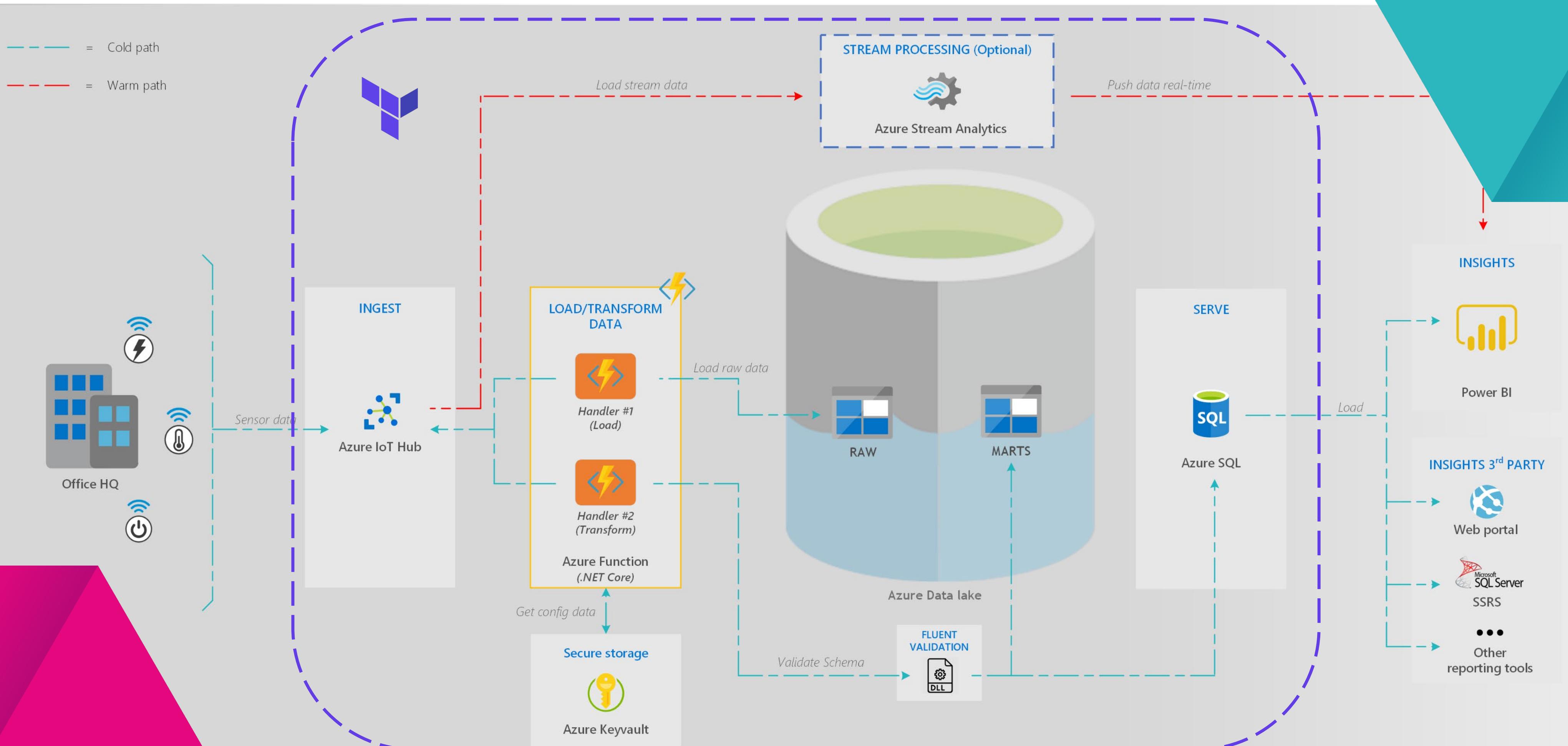
- Json configuration files
- Create any Azure resource
- Tracked deployments



INSPARK

Neextapp: Provisioningsw~~it~~d~~el~~e~~te~~ral~~o~~rm & ARM

Azure IoT Data Solution



#1 Setup terraform solution

1. Create a storage account in Azure to host your Terraform state files

Home > Subscriptions > MSDN - Sjoerd Borneman >

terraform Resource group

Search (Ctrl+ /) Add Edit columns Delete resource group Refresh Export to CSV Open query Assign tags Move Delete Export template Feedback

Overview

Subscription (change) : MSDN - Sjoerd Borneman Deployments : 1 Succeeded

Subscription ID : [REDACTED]

Tags (change) : Click here to add tags

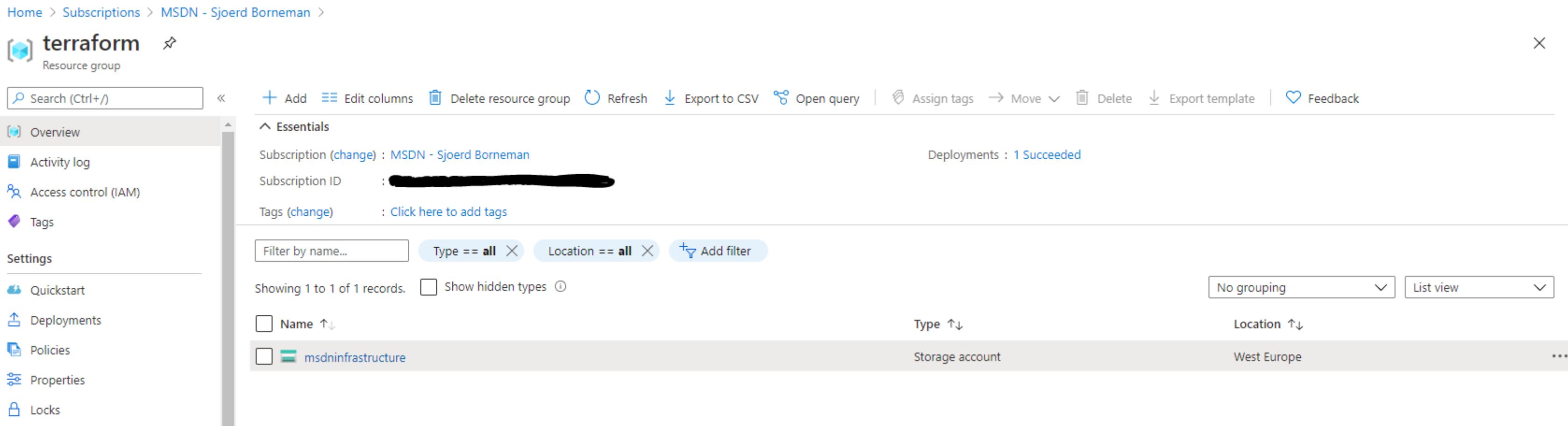
Filter by name... Type == all Location == all Add filter

Showing 1 to 1 of 1 records. Show hidden types

No grouping List view

Name ↑	Type ↑↓	Location ↑↓
msdninfrastructure	Storage account	West Europe

Quickstart Deployments Policies Properties Locks



But what are state files...? 🤔

#1 Setup terraform solution

2. Create a container to store all the Terraform state files

Home > msdninfrastructure

msdninfrastructure | Containers

Storage account

The screenshot shows the 'Containers' page for the 'msdninfrastructure' storage account. On the left, there's a sidebar with links: Overview, Activity log, Tags, Diagnose and solve problems, and Access Control (IAM). The main area has a search bar, a 'Container' button, and other navigation links. A search input field says 'Search containers by prefix'. Below it, a table lists two containers:

Name	Last modified
\$logs	9/23/2020, 3:57:12 PM
tfstate	9/23/2020, 5:43:54 PM

The 'tfstate' container is highlighted with a red box.

#1 Setup terraform solution

3. Save the access key of the storage account

The screenshot shows the 'Access keys' page for an Azure Storage account named 'msdninfrastructure'. The left sidebar contains navigation links for Overview, Activity log, Tags, Diagnose and solve problems, Access Control (IAM), Data transfer, Events, Storage Explorer (preview), Settings, Access keys (which is selected and highlighted in grey), Geo-replication, CORS, Configuration, Encryption, and Shared access signature. The main content area displays the storage account name 'msdninfrastructure' and two access keys: 'key1' and 'key2'. The 'key1' section is highlighted with a red rectangle around its 'Key' field, which contains a series of dots indicating a long string of characters. Below it is a 'Connection string' field also containing dots. The 'key2' section follows the same pattern. A note at the top right encourages users to store access keys securely using Azure Key Vault and to regenerate them regularly. A link to learn more about regenerating storage access keys is provided.

msdninfrastructure | Access keys

Storage account

Search (Ctrl+ /)

Use access keys to authenticate your applications when making requests to this Azure storage account. Store your access keys securely - for example, using Azure Key Vault - and don't share them. We recommend regenerating your access keys regularly. You are provided two access keys so that you can maintain connections using one key while regenerating the other.

When you regenerate your access keys, you must update any Azure resources and applications that access this storage account to use the new keys. This action will not interrupt access to disks from your virtual machines.

Learn more about regenerating storage access keys

Storage account name
msdninfrastructure

Show keys

key1

Key

Connection string

key2

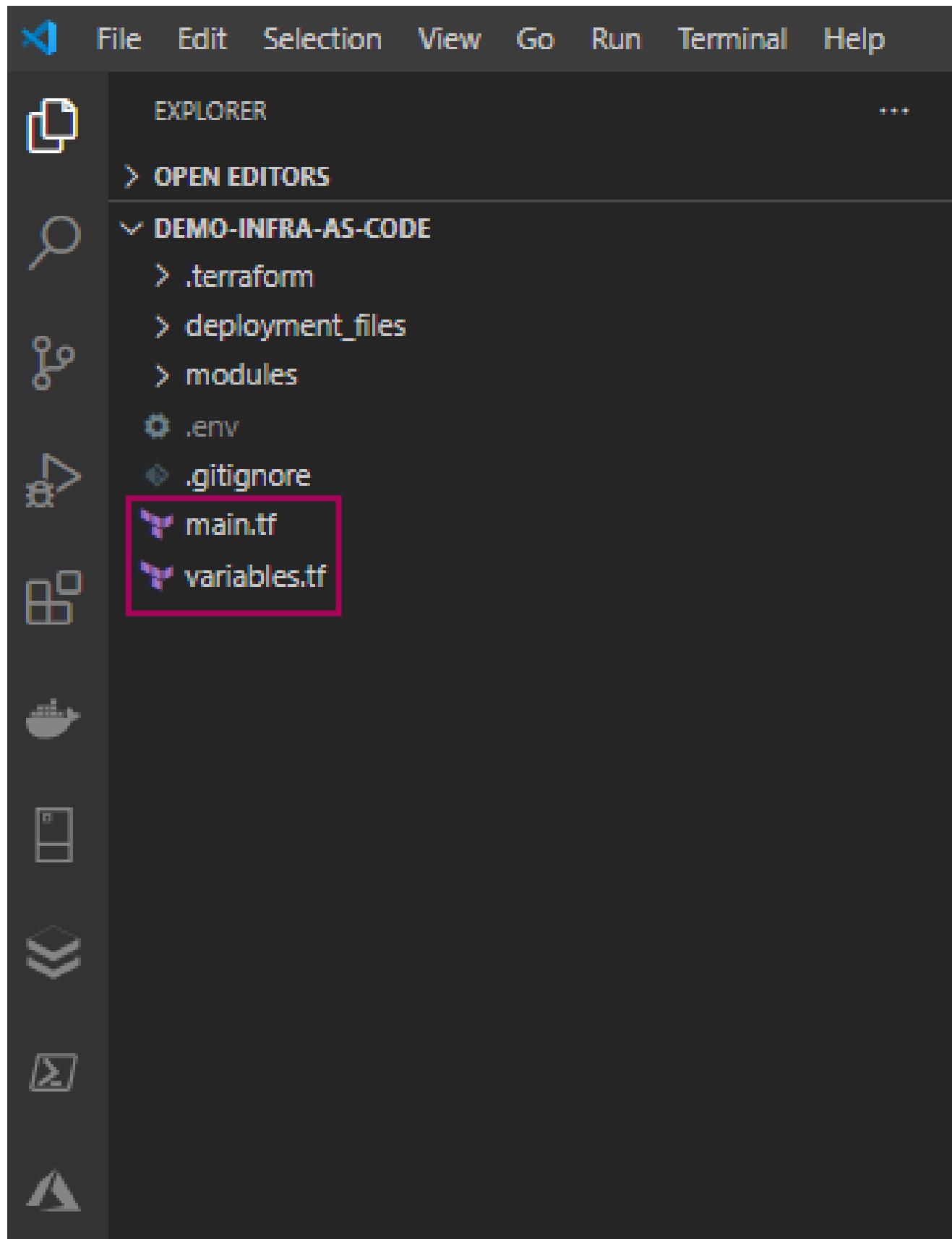
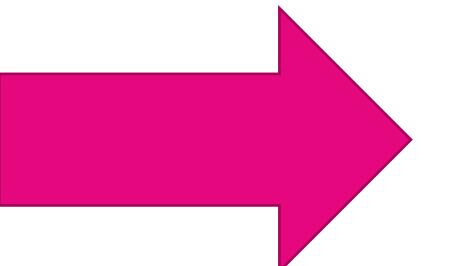
Key

Connection string

#1 Setup terraform solution

4. Create a terraform solution with the following directory structure

```
├── README.md  
├── main.tf  
├── variables.tf  
└── outputs.tf  
...  
└── modules/  
    ├── nestedA/  
    │   ├── README.md  
    │   ├── variables.tf  
    │   ├── main.tf  
    │   └── outputs.tf  
    └── nestedB/  
        .../  
└── examples/  
    ├── exampleA/  
    │   └── main.tf  
    └── exampleB/  
        .../
```



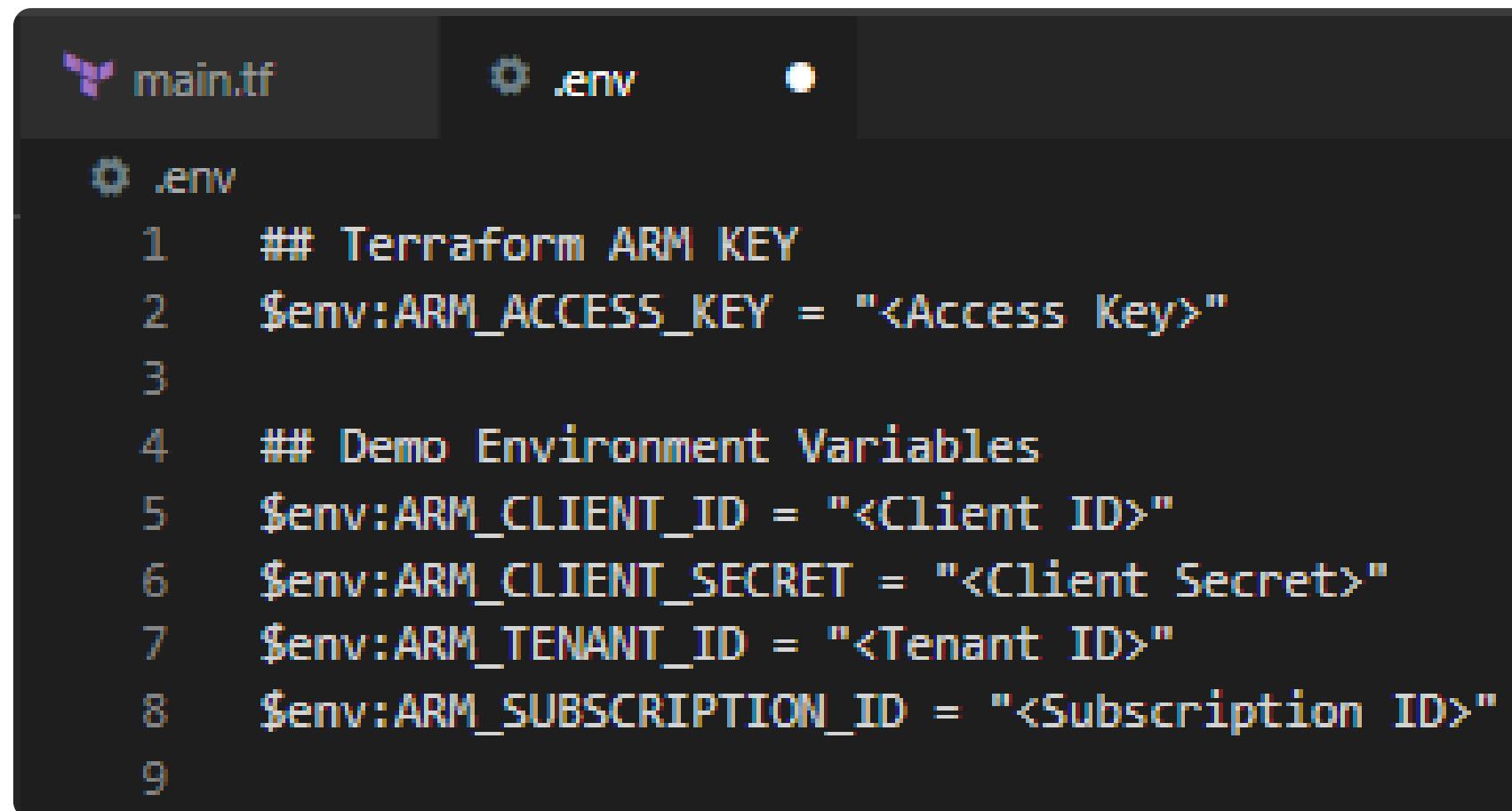
#1 Setup terraform solution

5. Write in the main.tf file to which provider we are going to deploy our infrastructure to, and which storage account is hosting our state files

```
# connection details to the terraform backend
# which hosts our terraform state files.
terraform {
  backend "azurerm" {
    storage_account_name = "msdninfrastructure"      ← Storage account name that hosts the state files
    container_name         = "tfstate"                  ← Container that contains the state files
    key                   = "infrastructure.tfstate"   ← Name of the Blob used to retrieve the state files
  }
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~> 2.24"
    }
  }
}
```

#1 Setup terraform solution

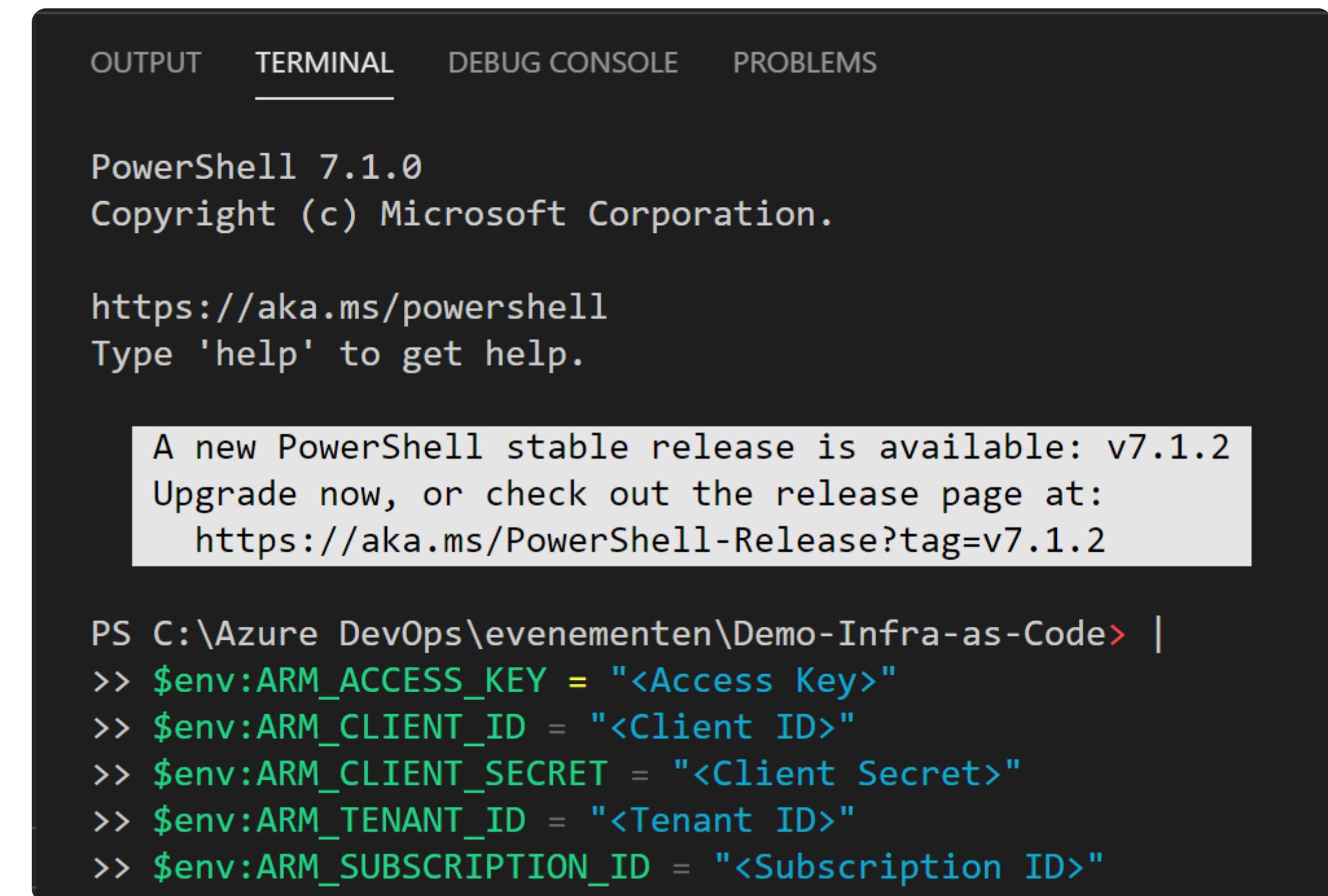
6. Fill the following environment variables with the correct credentials to authenticate to Azure
(for this example we are authenticating using a Service Principal)



The image shows a code editor interface with two tabs: 'main.tf' and '.env'. The '.env' tab is active, displaying the following content:

```
## Terraform ARM KEY
$env:ARM_ACCESS_KEY = "<Access Key>"

## Demo Environment Variables
$env:ARM_CLIENT_ID = "<Client ID>"
$env:ARM_CLIENT_SECRET = "<Client Secret>"
$env:ARM_TENANT_ID = "<Tenant ID>"
$env:ARM_SUBSCRIPTION_ID = "<Subscription ID>"
```



The image shows a PowerShell terminal window with the following output:

```
PowerShell 7.1.0
Copyright (c) Microsoft Corporation.

https://aka.ms/powershell
Type 'help' to get help.

A new PowerShell stable release is available: v7.1.2
Upgrade now, or check out the release page at:
https://aka.ms/PowerShell-Release?tag=v7.1.2
```

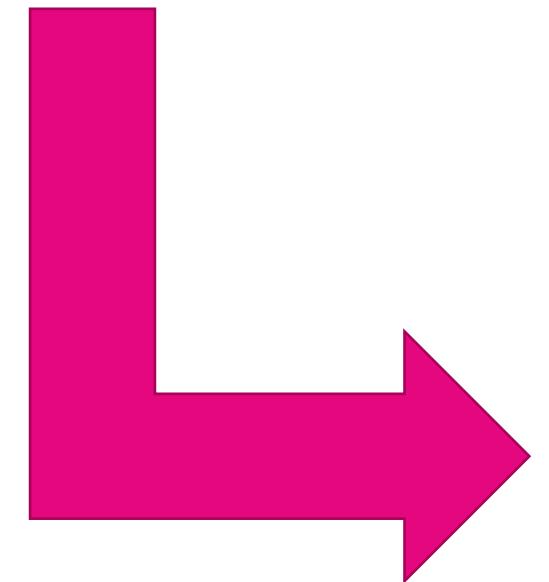
PS C:\Azure DevOps\evenementen\Demo-Infra-as-Code> |
>> \$env:ARM_ACCESS_KEY = "<Access Key>"
>> \$env:ARM_CLIENT_ID = "<Client ID>"
>> \$env:ARM_CLIENT_SECRET = "<Client Secret>"
>> \$env:ARM_TENANT_ID = "<Tenant ID>"
>> \$env:ARM_SUBSCRIPTION_ID = "<Subscription ID>"

#1 Setup terraform solution

7. Initialize the working directory and create a terraform workspace

Run the following commands:

```
terraform init
```



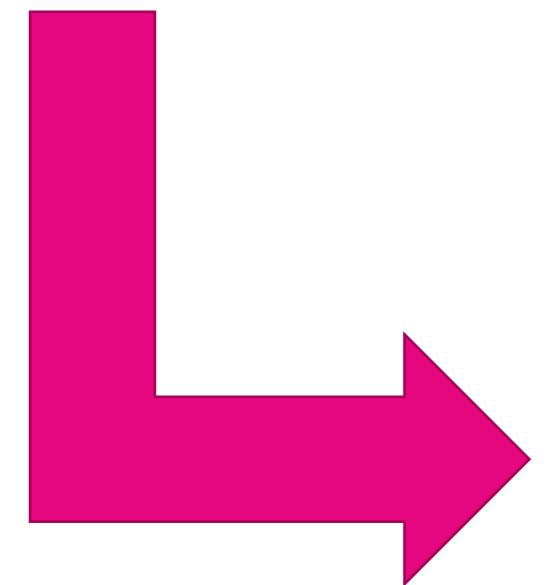
```
PS C:\Azure DevOps\evenementen\Demo-Infra-as-Code> terraform init  
  
Initializing the backend...  
  
Initializing provider plugins...  
- Using previously-installed hashicorp/azurerm v2.24.0  
  
Terraform has been successfully initialized!
```

#1 Setup terraform solution

7. Initialize the working directory and create a terraform workspace

Run the following commands:

```
terraform workspace list
```

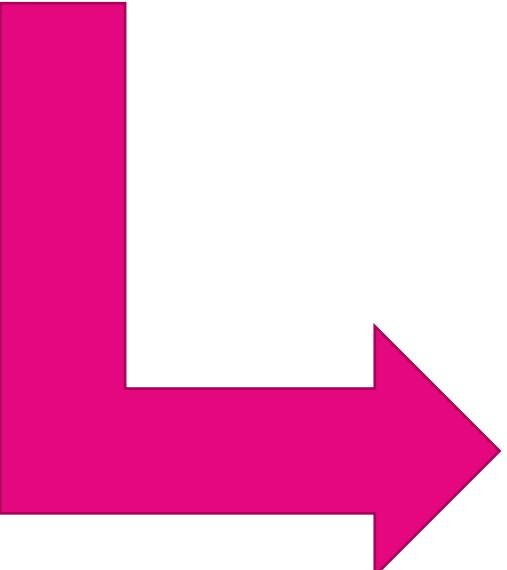
A screenshot of a terminal window is shown on the right. The window has tabs at the top labeled "OUTPUT", "TERMINAL", "DEBUG CONSOLE", and "PROBLEMS". The "TERMINAL" tab is active. The terminal output shows a PowerShell prompt "PS C:\Azure DevOps\evenementen\Demo-Infra-as-Code>" followed by the command "terraform workspace list". The output of the command is "* default", indicating the current workspace.

#1 Setup terraform solution

7. Initialize the working directory and create a terraform workspace

Run the following commands:

```
terraform workspace new eventDemo  
terraform workspace list
```



```
OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

PS C:\Azure DevOps\evenementen\Demo-Infra-as-Code> terraform workspace new eventDemo
Created and switched to workspace "eventDemo"!

You're now on a new, empty workspace. Workspaces isolate their state,
so if you run "terraform plan" Terraform will not see any existing state
for this configuration.
PS C:\Azure DevOps\evenementen\Demo-Infra-as-Code> terraform workspace list
  default
* eventDemo
```

#1 Setup terraform solution

7. Initialize the working directory and create a terraform workspace

Home > msdninfrastructure >

tfstate Container

Search (Ctrl+/)

Upload Change access level Refresh Delete Change tier Acquire lease Break lease

Overview

Authentication method: Access key (Switch to Azure AD User Account)
Location: tfstate

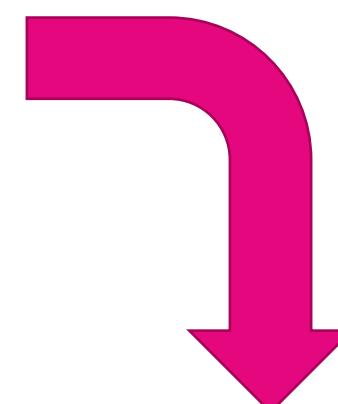
Search blobs by prefix (case-sensitive)

Name	Modified
No results	

Access Control (IAM)

Settings

Access policy Properties Metadata



Home > msdninfrastructure >

tfstate Container

Search (Ctrl+/)

Upload Change access level Refresh Delete Change tier Acquire lease Break lease View snapshots

Overview

Authentication method: Access key (Switch to Azure AD User Account)
Location: tfstate

Search blobs by prefix (case-sensitive)

Name	Modified
infrastructure.tfstateenv:eventDemo	3/4/2021, 8:27:47 PM

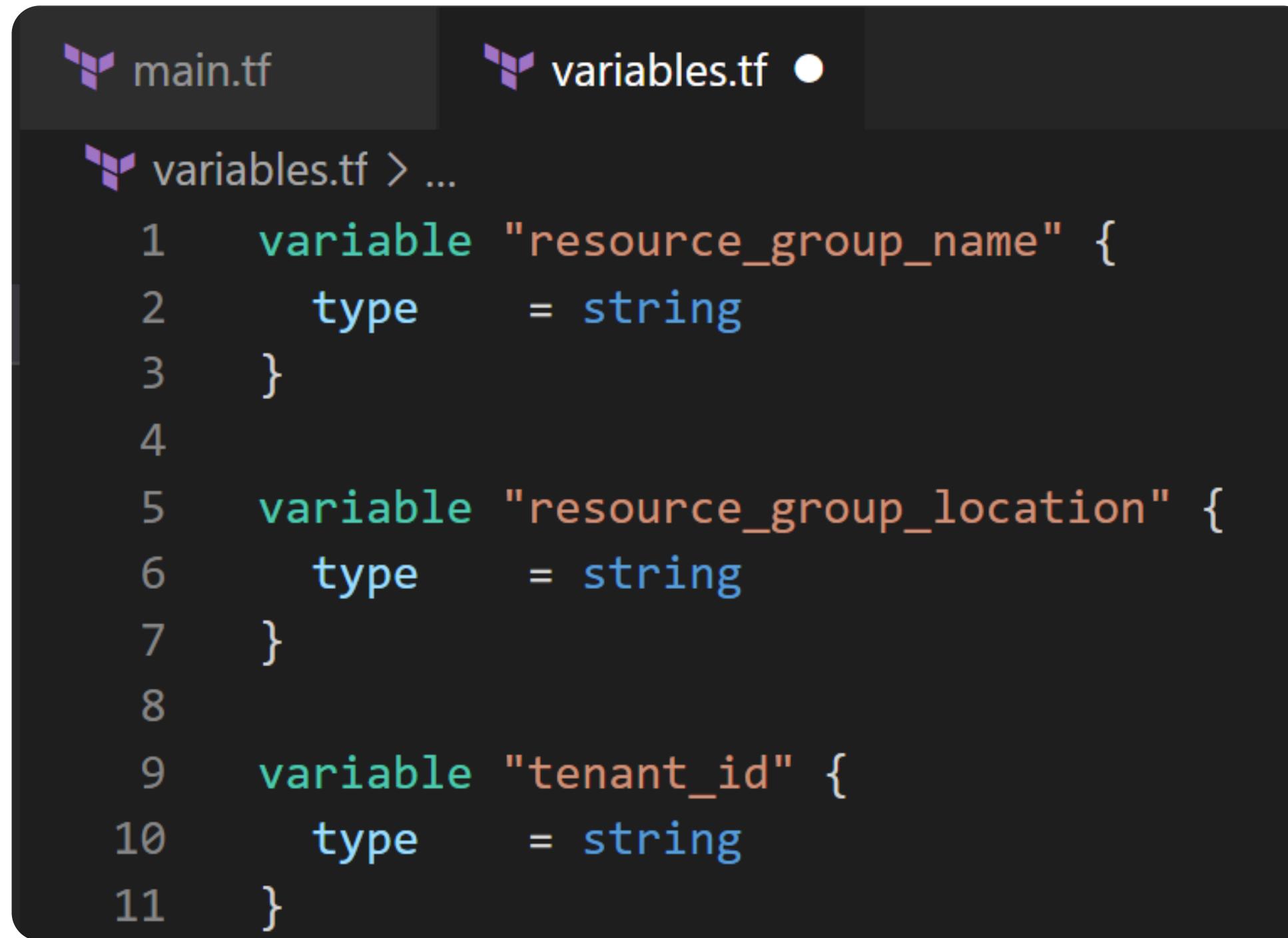
Access Control (IAM)

Settings

Shared access signature Access policy Properties Metadata

#2 Create a variables.tf & .tfvars file

1. Create variables in a variables.tf file that serves as parameters for your terraform module



```
main.tf
variables.tf

variables.tf > ...
1  variable "resource_group_name" {
2      type    = string
3  }
4
5  variable "resource_group_location" {
6      type    = string
7  }
8
9  variable "tenant_id" {
10     type    = string
11 }
```

#2 Create a variables.tf & .tfvars file

2. Fill in the required variables with the correct values. For this demo we are going to use the following values:

```
main.tf          variables.tf          dataSolution.tfvars ●  
dataSolution.tfvars  
1 resource_group_name      = "DEMO-EUW-IOT-SOLUTION-INFRA-AS-CODE-RG"  
2 resource_group_location = "West Europe"  
3 tenant_id              = "<Tenant ID>"  
4  
5
```

#3 Define the Azure IoT data services

: Azure Resource Group

```
# declaring the Azure provider with the desired version
provider "azurerm" {
    version = "=2.24.0"
    features {}
}

# connection details to the terraform backend which hosts our terraform state files.
terraform {
    backend "azurerm" {
        storage_account_name = "msdninfrastructure"
        container_name        = "tfstate"
        key                  = "infrastructure.tfstate"
    }
}
```

```
# define variables in locals so we can use it multiple times within a module without repeating it.
locals {
    resource_name_prefix      = "demo%solution%01"
}

# create a resource group which contains all the necessary
resource "azurerm_resource_group" "demo_iot_solution_rg" {
    name        = var.resource_group_name
    location   = var.resource_group_location
}
```

```
s within a module without repeating it. Locals
```

```
└── main.tf
    └── variables.tf
        └── dataSolution.tfvars •
            └── dataSolution.tfvars
                1   resource_group_name      = "DEMO-EUW-IOT-SOLUTION-INFRA-AS-CODE-RG"
                2   resource_group_location = "West Europe"
                3   tenant_id              = "<Tenant ID>"
```

#3 Define the Azure IoT data services

: Azure IoT Hub

The diagram illustrates the components of an Azure IoT Hub and how they are defined in a Terraform configuration file (`main.tf`).

Resource Group Configuration:

```
resource "azurerm_resource_group" "demo_iot_solution_rg" {  
    name      = var.resource_group_name  
    location  = var.resource_group_location  
}
```

IoT Hub Configuration:

```
# create a IoT Hub in Azure with the necessary configurations  
resource "azurerm_iothub" "iothub" {  
    name          = format(local.resource_name_prefix, "euw", "iot") # ← demoeuwsolutioniot01  
    resource_group_name = azurerm_resource_group.demo_iot_solution_rg.name # dependency  
    location       = azurerm_resource_group.demo_iot_solution_rg.location # dependency  
  
    sku {  
        name      = "S1"  
        capacity  = "1"  
    }  
  
    # here you can whitelist ip's to restrict traffic to the IoT Hub  
    ip_filter_rule {  
        name      = "home-ip"  
        ip_mask   = "192.0.0.0" # dummy ip  
        action    = "Accept"  
    }  
  
    # add tags for e.g. azure policies, resource management, audits, etc  
    tags = {  
        purpose  = "demoEnvironment"  
        subject  = "InfrastructureAsCode"  
    }  
}
```

Diagram Components:

- Stock Keeping Unit (SKU):** Represented by a blue box containing the `sku` block. It is connected to the `name` field of the `azurerm_iothub` resource.
- IP Rules:** Represented by a blue box containing the `ip_filter_rule` block. It is connected to the `ip_filter_rule` field of the `azurerm_iothub` resource.
- Resource tags:** Represented by a blue box containing the `tags` block. It is connected to the `tags` field of the `azurerm_iothub` resource.
- locals:** Represented by a pink box containing the `locals` block. It is connected to the `name` field of the `azurerm_iothub` resource.

Azure IoT Hub: A light gray box labeled "INGEST" containing a network icon. It receives data from the `azurerm_iothub` resource via a red dashed arrow.

#3 Define the Azure IoT data services

: Azure Data lake Gen2

```
main.tf
```

```
# create a Data lake gen2 in Azure with the necessary configurations
resource "azurerm_storage_account" "datalake_gen2" {
    name                  = format(local.resource_name_prefix, "euw", "dl") # ← demoeuwsolutiondl01
    resource_group_name   = azurerm_resource_group.riot_solution_rg.name
    location              = azurerm_resource_group.riot_solution_rg.location
    account_kind          = "StorageV2"
    account_tier           = "Standard"
    account_replication_type = "LRS"

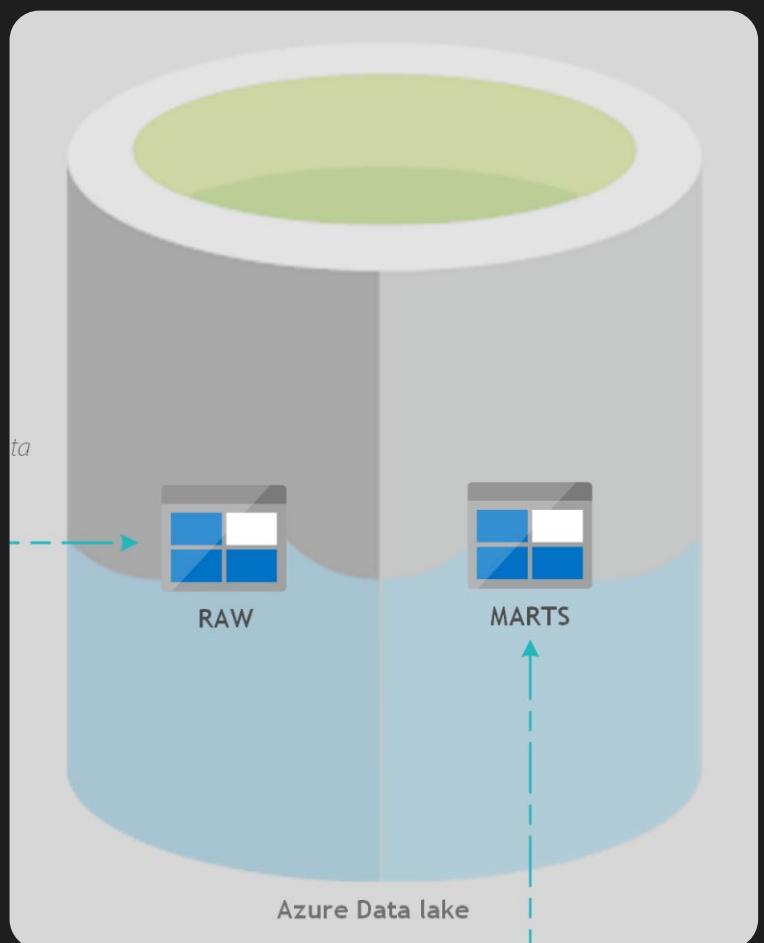
    # actual switch for ADLS Gen2
    is_hns_enabled = true

    # security
    enable_https_traffic_only = true

    tags = {
        purpose = "demoEnvironment"
        subject = "InfrastructureAsCode"
    }
}
```

Enable Gen2

HTTPS Only



#3 Define the Azure IoT data services

: Azure Stream Analytics

```
main.tf
```

```
# create a Stream Analytics resource in Azure with the necessary configurations
resource "azurerm_stream_analytics_job" "stream_analytics" {
    name          = format(local.resource_name_prefix, "euw", "sa") # ← demoeuwsolutionSa01
    resource_group_name = azurerm_resource_group.iot_solution_rg.name
    location      = azurerm_resource_group.iot_solution_rg.location
    compatibility_level = "1.1"
    streaming_units     = 3

    transformation_query = <<QUERY
        SELECT *
        INTO [YourOutputAlias]
        FROM [YourInputAlias]
    QUERY
}

tags = {
    purpose = "demoEnvironment"
    subject = "InfrastructureAsCode"
}
```

ASA Query

The diagram features a dashed blue rectangular border labeled 'STREAM PROCESSING (Optional)' at the top. Inside this border is a grey rounded rectangle containing a grey gear icon with three blue wavy lines representing data flow. Below the gear is the text 'Azure Stream Analytics'.

#3 Define the Azure IoT data services

: Azure Function (storage)

```
main.tf
```

```
# create a storage account for the azure function in Azure with the necessary configurations.
resource "azurerm_storage_account" "functions_storage" {
    name          = format(local.resource_name_prefix, "euw", "stg") # ← demoeuwsolutionstg01
    resource_group_name = azurerm_resource_group.iot_solution_rg.name
    location      = azurerm_resource_group.iot_solution_rg.location
    account_tier   = "Standard"
    account_replication_type = "LRS"

    # security
    enable_https_traffic_only = true

    tags = {
        purpose = "demoEnvironment"
        subject = "InfrastructureAsCode"
    }
}
```

The diagram illustrates the architecture of an Azure Function (.NET Core). It features a central gray box labeled "LOAD/TRANSFORM DATA" with a yellow lightning bolt icon at the top right. Two orange boxes, each with a lightning bolt icon, represent "Handler #1 (Load)" and "Handler #2 (Transform)". Arrows point from both handlers to the central data processing box. A blue arrow points upwards from the bottom of the central box towards the bottom edge of the slide.

#3 Define the Azure IoT data services

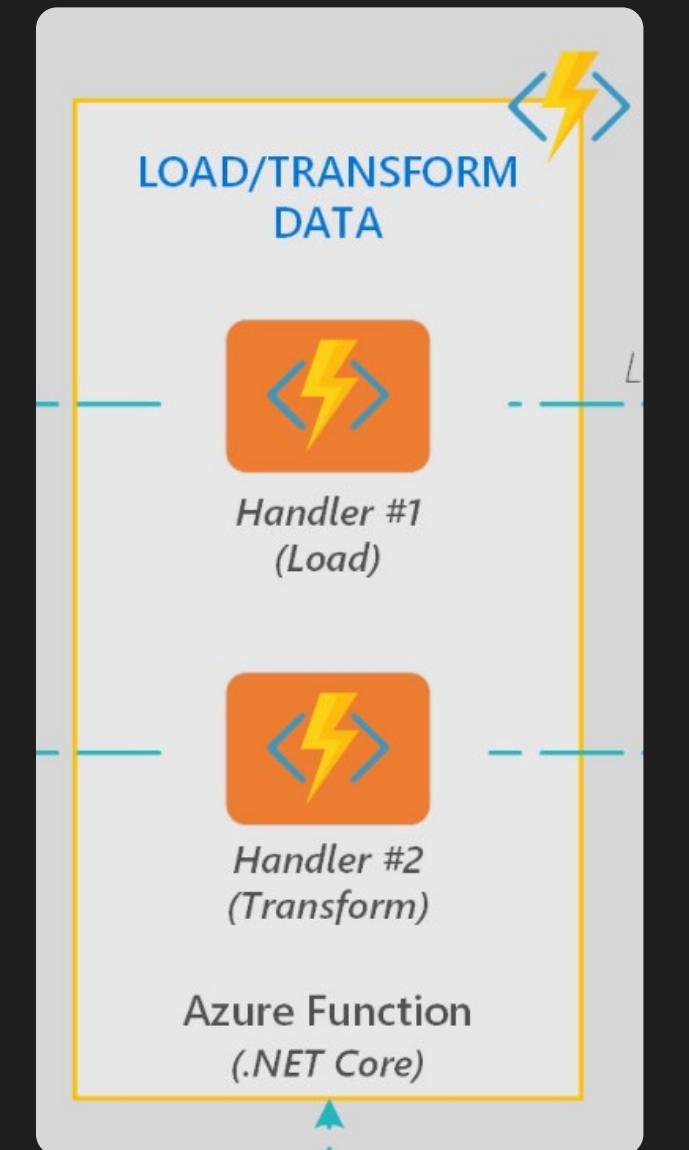
: Azure Function (service plan)

main.tf

```
# create a app service plan for the azure function in Azure with the necessary configurations.
resource "azurerm_app_service_plan" "functions_service_plan" {
    name          = format(local.resource_name_prefix, "euw", "appsplan") # ← demoeuwsolutionappsplan01
    resource_group_name = azurerm_resource_group.iot_solution_rg.name
    location      = azurerm_resource_group.iot_solution_rg.location
    kind          = "FunctionApp"

    sku {
        tier = "Dynamic"
        size = "Y1"
    }

    tags = {
        purpose = "demoEnvironment"
        subject = "InfrastructureAsCode"
    }
}
```



#3 Define the Azure IoT data services

: Azure Function

main.tf

```
# create a function resource in Azure with the necessary configurations.
resource "azurerm_function_app" "functions" {
    name                  = format(local.resource_name_prefix, "euw", "fnc") # ← demoeuwsolutionfnc01
    resource_group_name   = azurerm_resource_group.iot_solution_rg.name
    location              = azurerm_resource_group.iot_solution_rg.location
    app_service_plan_id   = azurerm_app_service_plan.functions_service_plan.id          # dependency
    storage_account_name  = azurerm_storage_account.functions_storage.name            # dependency
    storage_account_access_key = azurerm_storage_account.functions_storage.primary_access_key # dependency

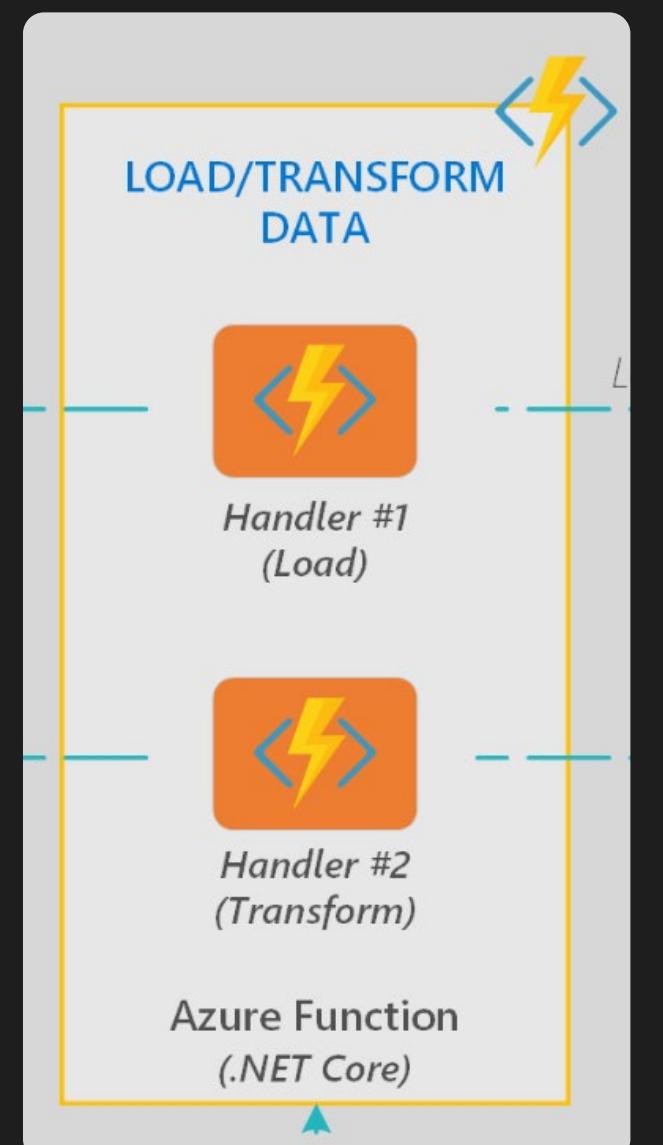
    version = "~3"
    auth_settings {
        enabled      = true
        runtime_version = "dotnet"
    }

    app_settings = {
        iothub_connectionstring      = azurerm_iothub.iothub.event_hub_events_endpoint
        datalake_gen2_connectionstring = azurerm_storage_account.datalake_gen2.primary_connection_string
    }

    tags = {
        purpose = "demoEnvironment"
        subject = "InfrastructureAsCode"
    }
}
```

Authentication settings

Application settings



#3 Define the Azure IoT data services

: Azure SQL Server

```
main.tf
```

```
# create a SQL Server in Azure with the necessary configurations
resource "azurerm_sql_server" "sqlserver" {
    name          = format(local.resource_name_prefix, "euw", "sqlsvr") # ← demoeuwsolutionSqlsvr01
    resource_group_name = azurerm_resource_group.iot_solution_rg.name
    location      = azurerm_resource_group.iot_solution_rg.location
    version       = "12.0"

    # sql admin user
    administrator_login      = "dummy_admin"
    administrator_login_password = "abcdefghijklmnop01!"

    identity {
        type = "SystemAssigned"
    }

    tags = {
        purpose = "demoEnvironment"
        subject = "InfrastructureAsCode"
    }
}
```

The diagram illustrates the connection between two Azure components and an external service. On the left, a code snippet defines an Azure SQL Server resource named 'sqlserver'. It includes configuration for the server's name, resource group, location, and version. It also specifies a 'sql admin user' with the login 'dummy_admin' and a complex password. Additionally, it defines a 'Managed Identity' with a 'SystemAssigned' type. Two blue arrows point from the 'Managed Identity' and 'SQL Admin' sections of the code towards a central box labeled 'Azure SQL'. To the right of this box is a teal arrow pointing upwards, and above the box is the word 'SERVE'.

SQL Admin

Managed Identity

SERVE

Azure SQL

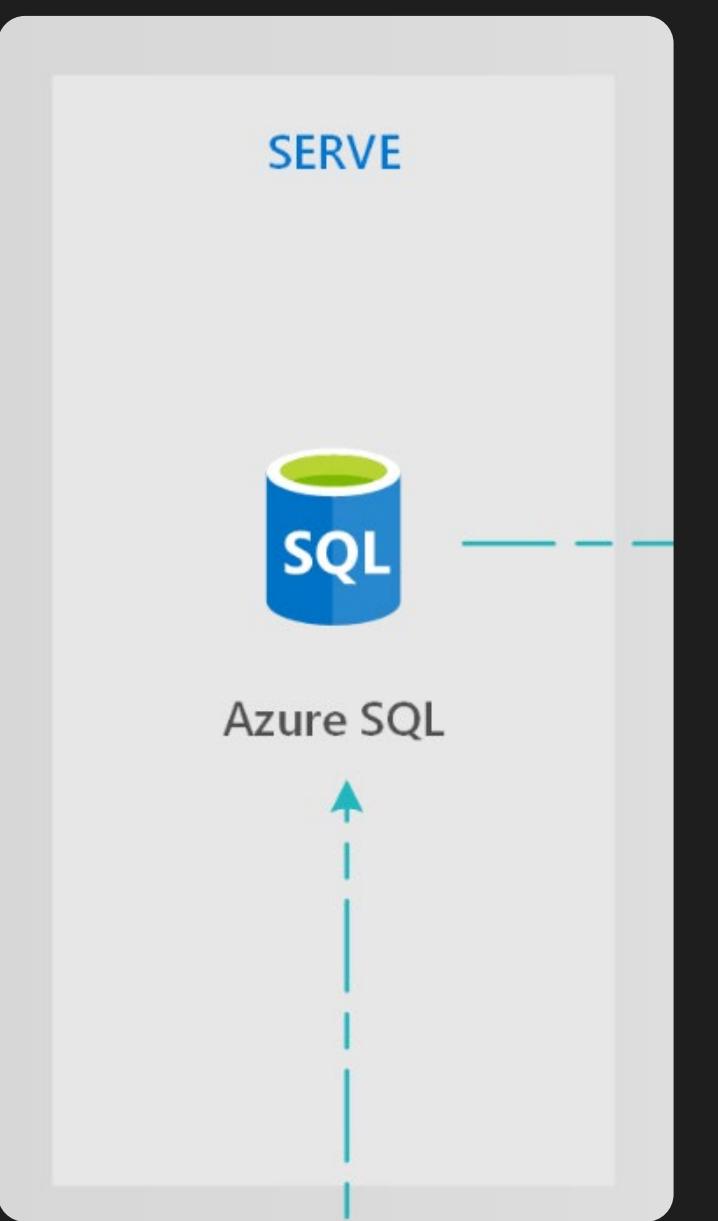
#3 Define the Azure IoT data services

: Azure SQL Database

main.tf

```
# create a SQL Database in Azure with the necessary configurations
resource "azurerm_sql_database" "database" {
    name          = format(local.resource_name_prefix, "euw", "sqldb") # ← demoeuwsolutionsqldb01
    resource_group_name = azurerm_resource_group.iot_solution_rg.name
    location      = azurerm_resource_group.iot_solution_rg.location
    server_name   = azurerm_sql_server.sqlserver.name
    edition       = "Standard"

    tags = {
        purpose = "demoEnvironment"
        subject = "InfrastructureAsCode"
    }
}
```



#5 Generate execution plan

The execution plan specifies what actions Terraform will take to achieve the desired state defined in the configuration files

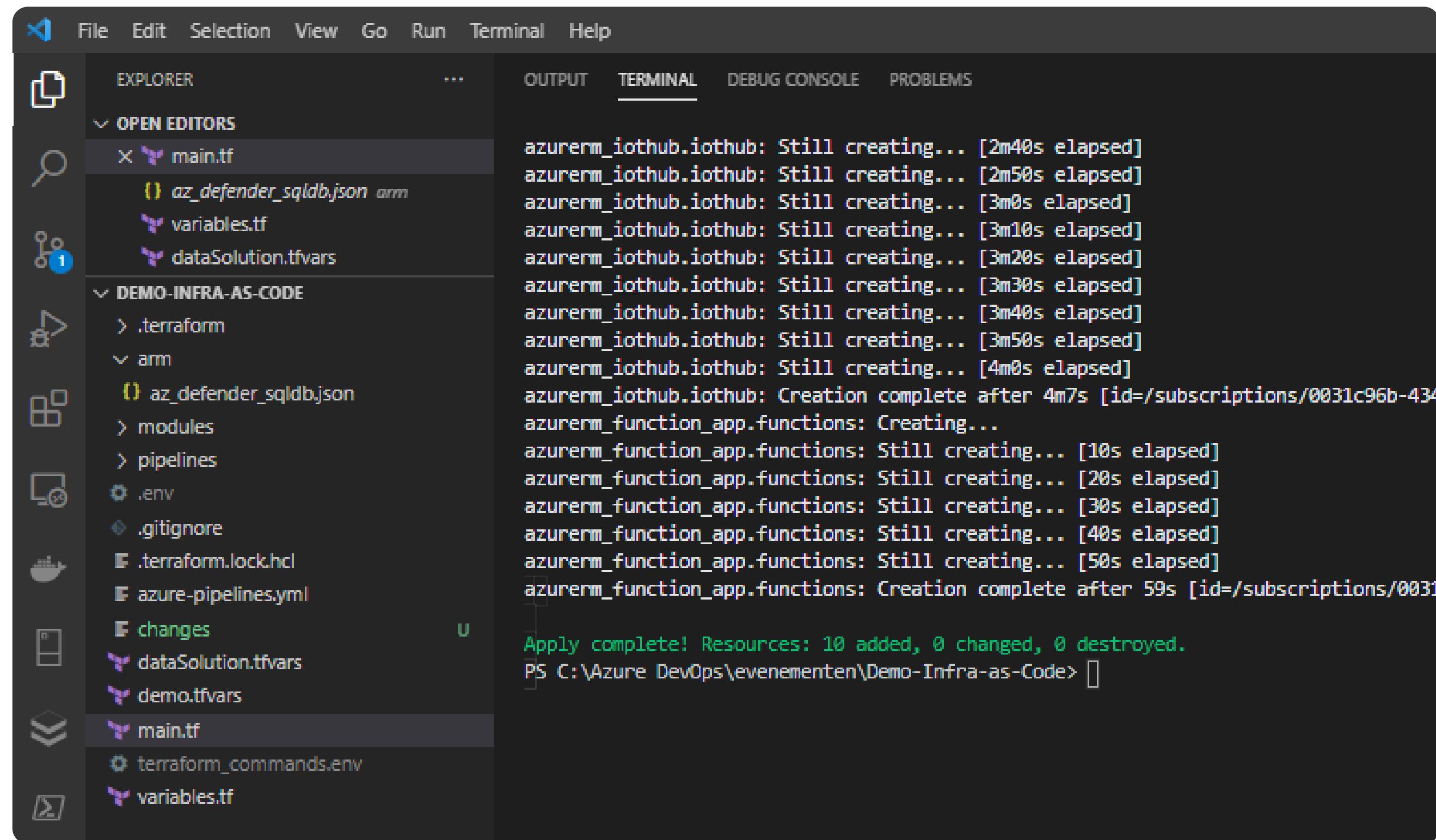
Run the *terraform plan* command to generate and save the execution plan for your configuration

```
terraform plan --var-file .\dataSolution.tfvars -out changes
```


#6 Deploy configuration

Run the *terraform apply* command to apply your configuration

terraform apply changes



```
azurerm_iothub.iothub: Still creating... [2m40s elapsed]
azurerm_iothub.iothub: Still creating... [2m50s elapsed]
azurerm_iothub.iothub: Still creating... [3m0s elapsed]
azurerm_iothub.iothub: Still creating... [3m10s elapsed]
azurerm_iothub.iothub: Still creating... [3m20s elapsed]
azurerm_iothub.iothub: Still creating... [3m30s elapsed]
azurerm_iothub.iothub: Still creating... [3m40s elapsed]
azurerm_iothub.iothub: Still creating... [3m50s elapsed]
azurerm_iothub.iothub: Still creating... [4m0s elapsed]
azurerm_iothub.iothub: Creation complete after 4m7s [id=/subscriptions/0031c96b-434
azurerm_function_app.functions: Creating...
azurerm_function_app.functions: Still creating... [10s elapsed]
azurerm_function_app.functions: Still creating... [20s elapsed]
azurerm_function_app.functions: Still creating... [30s elapsed]
azurerm_function_app.functions: Still creating... [40s elapsed]
azurerm_function_app.functions: Still creating... [50s elapsed]
azurerm_function_app.functions: Creation complete after 59s [id=/subscriptions/0031
Apply complete! Resources: 10 added, 0 changed, 0 destroyed.
PS C:\Azure DevOps\evenementen\Demo-Infra-as-Code> 
```

#6 Deploy configuration

Run the *terraform apply* command to apply your configuration

terraform apply changes

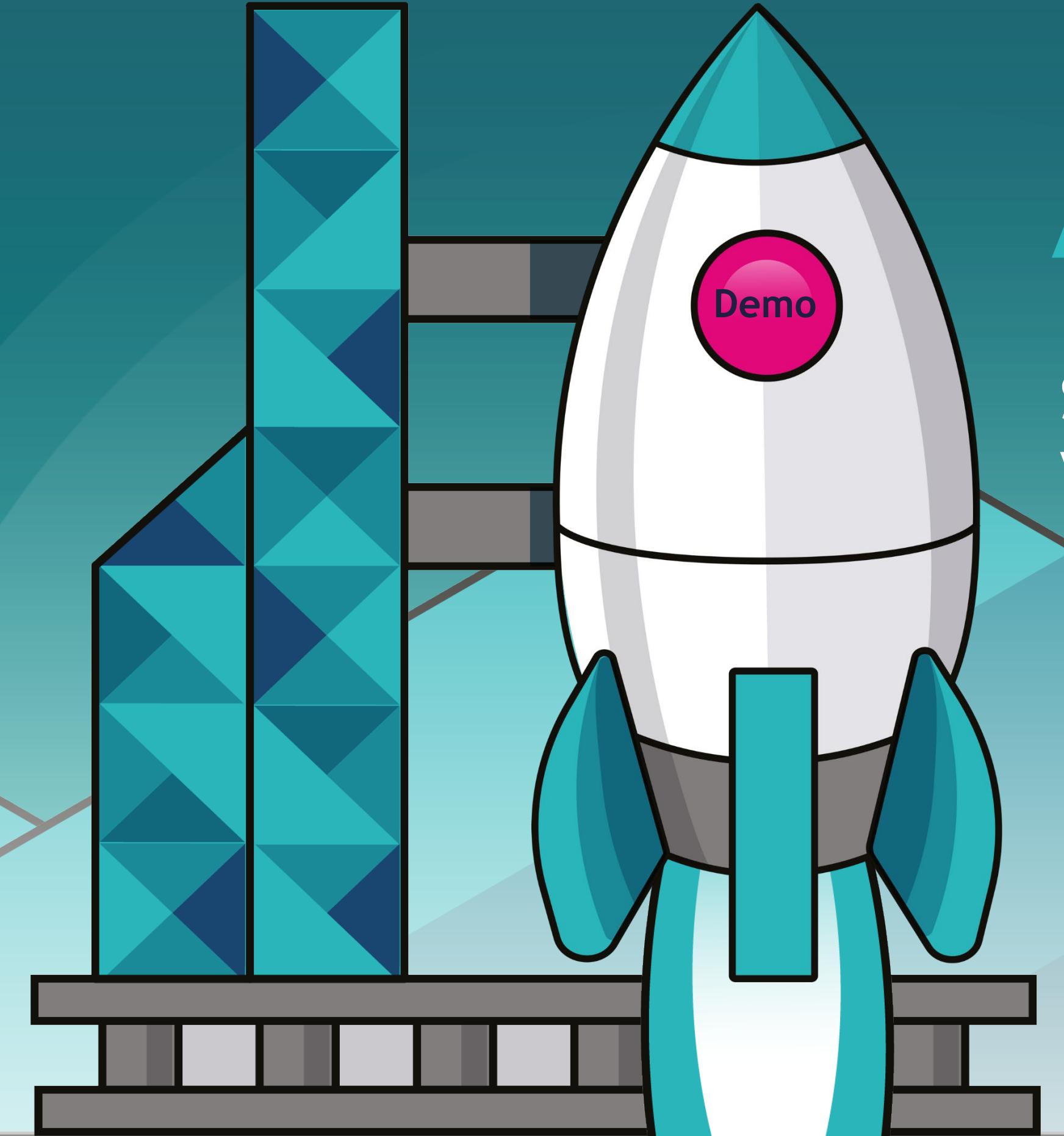
The screenshot shows the Microsoft Azure Resource Group Overview page for the resource group 'DEMO-EUW-IOT-SOLUTION-INFRA-AS-CODE-RG'. The page displays various details about the resource group, including its subscription, location, and tags. A table lists the deployed resources, which are highlighted with a red border.

Resource Group Details:

- Subscription (change) : MSDN - Sjoerd Borneman
- Subscription ID : 0031c96b-434c-4ace-a7ca-75f3739b646b
- Tags (change) : Click here to add tags
- Deployments : No deployments
- Location : West Europe

Resources Deployed:

Name	Type	Location
demoeuwsolutionappsplan01	App Service plan	West Europe
demoeuwsolutiondl01	Storage account	West Europe
demoeuwsolutionfnc01	Function App	West Europe
demoeuwsolutioniot01	IoT Hub	West Europe
demoeuwsolutionsa01	Stream Analytics job	West Europe
demoeuwsolutionsql01 (demoeuwsolutionsqlsvr01/demoeuwsolutionsql01)	SQL database	West Europe
demoeuwsolutionsqlsvr01	SQL server	West Europe
demoeuwsolutionstg01	Storage account	West Europe
demoeuwsolutionvault01	Key vault	West Europe

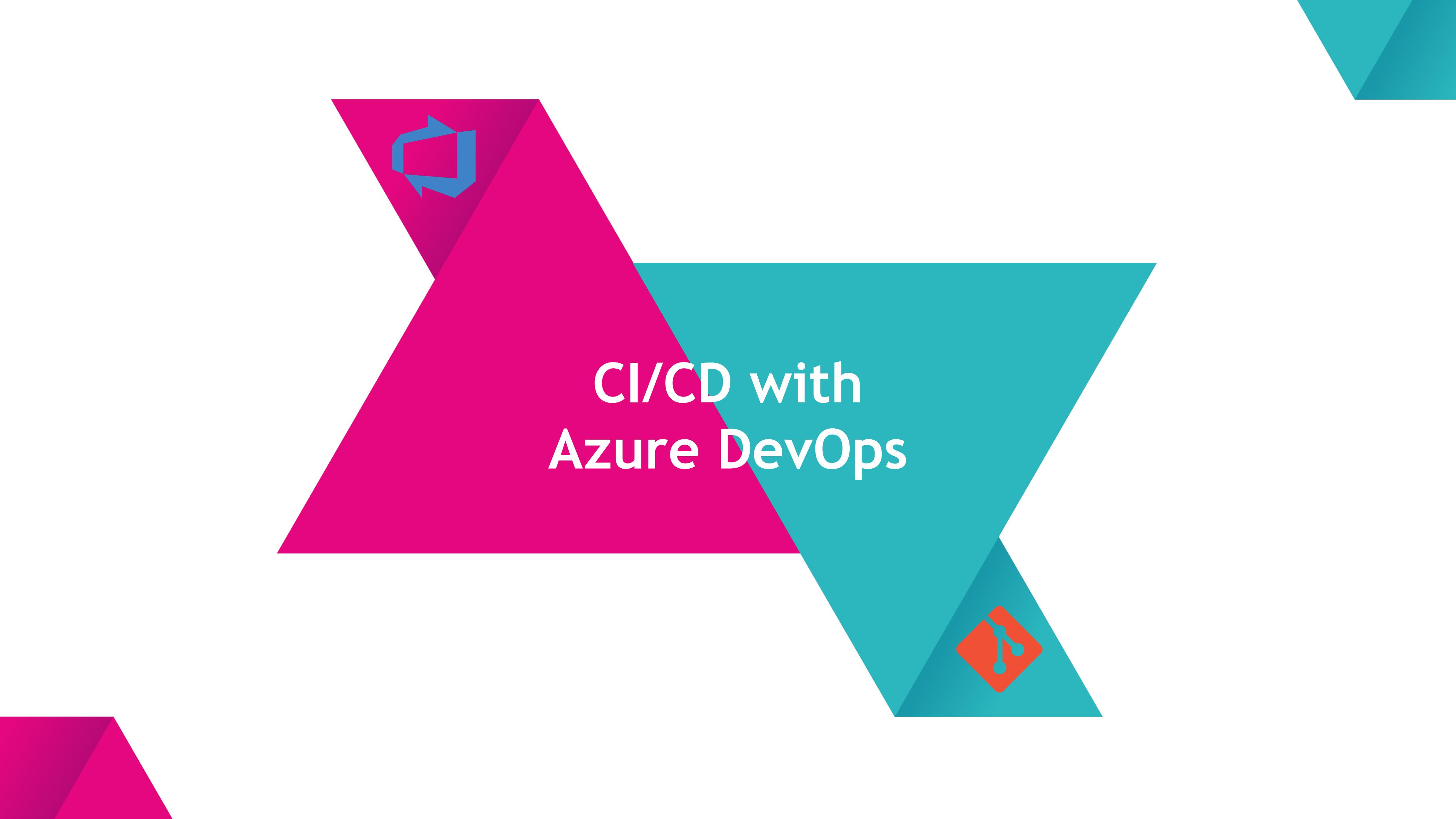


ARM
Enable Azure Defender for
SQL &
Vulnerability Assessments



INSPARK

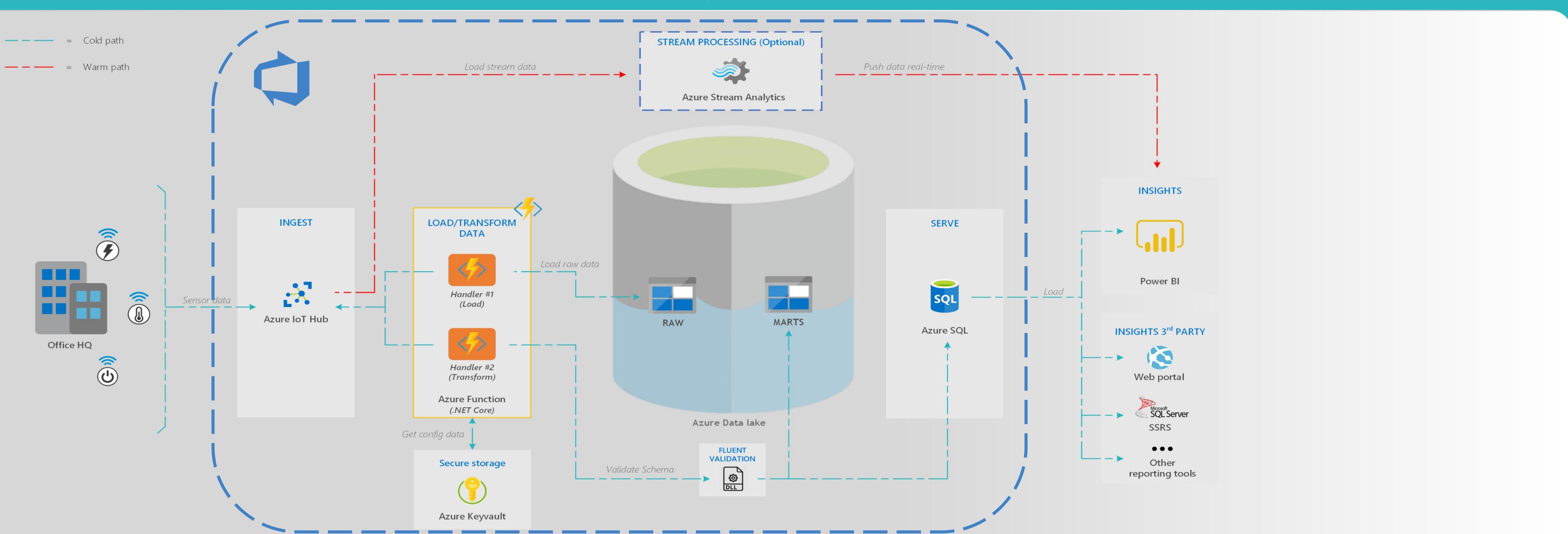
Neextapp: Provisioning infrastructure using Azure DevOps



CI/CD with Azure DevOps



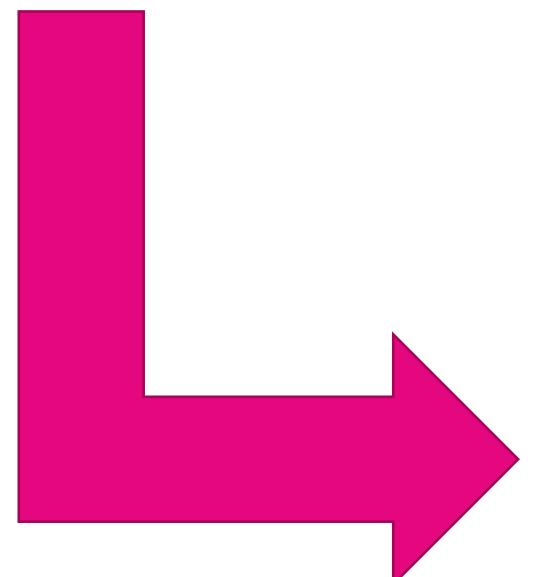
Use Azure DevOps when deploying your Azure Data Solution (Demo)



#1 Commit Terraform configuration files

Commit early created configuration files to Azure DevOps

```
git add .
git commit -m "Azure IoT Environment"
git push
```



The screenshot shows the Azure DevOps interface with the 'Source control' view for the 'Demo-Infra-as-Code' repository. The repository structure is as follows:

- arm
- modules
- pipelines
- .gitignore
- .terraform.lock.hcl
- azure-pipelines.yml
- dataSolution.tfvars
- demo.tfvars
- main.tf
- variables.tf

A pink box highlights the 'Source control' section at the bottom right of the interface.

#2 Install Extensions

Install the following dependencies in Azure DevOps

Extensions

Installed Requested Shared

Security Browse marketplace

Terraform by Microsoft DevLabs
Install terraform and run terraform commands to manage resources on Azure, AWS and GCP.

Terraform by Peter Groenewegen
Build extension that enable you to run Terraforms on the build agent.

Zip and unzip directory build task by Peter Groenewegen
Build extension that enable you to create zips from directories and unzip a zip file into a directory.

Azure DevOps Extensions

#3 Create Pipelines : Build

The screenshot shows the Azure DevOps interface with the 'MSDN' repository selected. The left sidebar includes links for Overview, Boards, Repos, Files, Commits, Pushes, Branches, Tags, Pull requests, Pipelines, Test Plans, and Artifacts. The main area displays the repository structure under 'MSDN': 'Demo-Infra-as-Code' contains 'arm', 'modules', 'pipelines', '.gitignore', and '.terraform.lock.hcl'. A file named 'azure-pipelines.yml' is highlighted in blue, indicating it is the current focus.

```
trigger:  
- master  
  
pool:  
vmImage: 'windows-latest'  
  
variables:  
terraform_workspace: 'eventDemo'  
  
jobs:  
- job: "build_iot_environment"  
displayName: "build iot environment"  
  
steps:  
- checkout: self  
persistCredentials: true  
clean: true  
  
- template: /pipelines/templates/set_environment_variables.yml  
  
- template: /pipelines/templates/build_terraform.yml  
parameters:  
TF_SRC: '$(System.DefaultWorkingDirectory)'  
ARTIFACT_NAME: 'IoT'  
ARTIFACT_FILE: 'iot-$(Build.BuildNumber).zip'
```

trigger

pool

variables

job: build

job steps: build

#3 Create Pipelines : Release

Azure DevOps SjoerdBorneman0997 / MSDN / Repos

M MSDN +

- Overview
- Boards
- Repos**
- Files
- Commits
- Pushes
- Branches
- Tags
- Pull requests
- Pipelines
- Test Plans
- Artifacts

Demo-Infra-as-Code

- arm
- modules
- pipelines
- .gitignore
- .terraform.lock.hcl
- azure-pipelines.yml**
- dataSolution.tfvars
- demo.tfvars
- main.tf
- variables.tf

```
- deployment: deploy_iot_environment
  displayName: deploy iot environment
  dependsOn: build_iot_environment
  job: release

  environment: Demo Azure IoT Solution
  strategy:
    runOnce:
      deploy:
        steps:
          - task: TerraformInstaller@0
            inputs:
              terraformVersion: 0.13.5
  task: install terraform

  - template: /pipelines/templates/set_environment_variables.yml
  template: set env variables

  - task: ExtractFiles@1
    displayName: 'Extract files'
    inputs:
      archiveFilePatterns: '$(Pipeline.Workspace)/IoT/iot-$(Build.BuildNumber).zip'
      destinationFolder: '$(System.DefaultWorkingDirectory)/infrastructure'
  task: extract artifact

  - task: PowerShell@2
    displayName: 'initialize'
    inputs:
      targetType: 'inline'
      script: |
        terraform init
    workingDirectory: '$(System.DefaultWorkingDirectory)/infrastructure'
  task: initialize terraform

  - task: PowerShell@2
    displayName: 'set terraform workspace'
    inputs:
      targetType: 'inline'
      script: |
        terraform workspace select '$(terraform_workspace)'
    workingDirectory: '$(System.DefaultWorkingDirectory)/infrastructure'
  task: select terraform workspace

  - task: PowerShell@2
    displayName: 'generate execution plan'
    inputs:
      targetType: 'inline'
      script: |
        terraform plan --var-file .\dataSolution.tfvars -out changes
    workingDirectory: '$(System.DefaultWorkingDirectory)/infrastructure'
  task: generate execution plan

  - task: PowerShell@2
    displayName: 'apply execution plan'
    inputs:
      targetType: 'inline'
      script: |
        terraform apply changes
    workingDirectory: '$(System.DefaultWorkingDirectory)/infrastructure'
  task: deploy environment
```

#4 Trigger pipeline

Commit a change to the environment to trigger the build & release pipeline

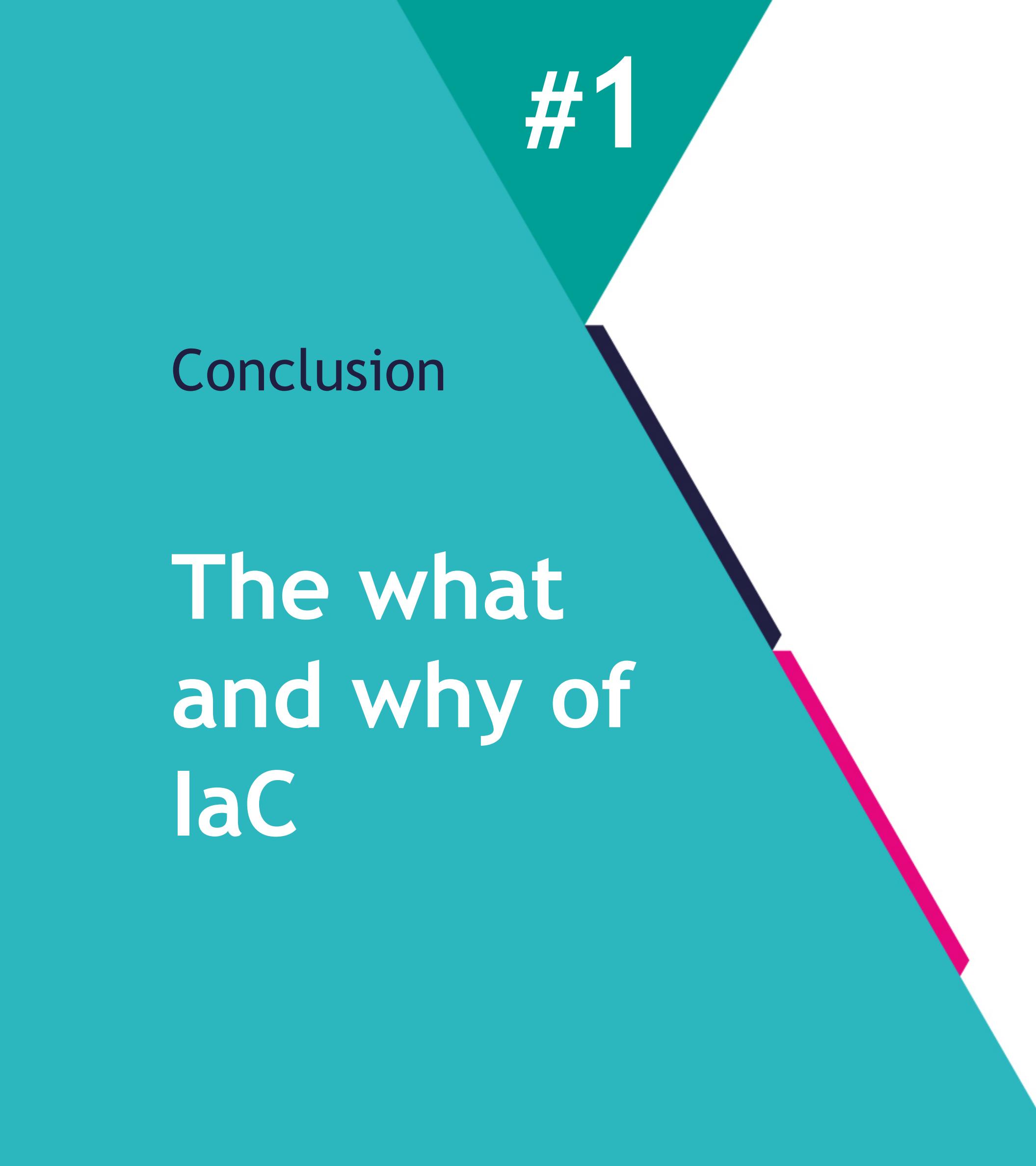
The screenshot shows the Microsoft Azure portal interface. At the top, there's a navigation bar with 'Jobs in run #20200929.2' and 'apply execution plan' buttons. Below the navigation bar is a search bar and a user profile section for 'Sjoerd.Borneman@insp... MSDN ENVIRONMENT'. The main content area is titled 'DEMO-EUW-IOT-SOLUTION-INFRA-AS-CODE-RG' and shows the 'Resource group' overview. On the left, there's a sidebar with 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', and 'Events' sections, followed by 'Settings' with options like 'Deployments', 'Security', 'Policies', 'Properties', 'Locks', 'Cost Management', 'Cost analysis', 'Cost alerts (preview)', 'Budgets', and 'Advisor recommendations'. The main pane displays the 'Essentials' section with details about the subscription (change), ID, and location (West Europe). It also shows deployment status ('No deployments') and a list of resources. A red box highlights the 'Resources' section, which lists the following items:

Name	Type	Location
demoeuwsolutionappsplan01	App Service plan	West Europe
demoeuwsolutiondl01	Storage account	West Europe
demoeuwsolutionfnc01	Function App	West Europe
demoeuwsolutioniot01	IoT Hub	West Europe
demoeuwsolutionsa01	Stream Analytics job	West Europe
demoeuwsolutionsql01 (demoeuwsolutionsqlsvr01/demoeuwsolutionsql01)	SQL database	West Europe
demoeuwsolutionsqlsvr01	SQL server	West Europe
demoeuwsolutionstg01	Storage account	West Europe
demoeuwsolutionvault01	Key vault	West Europe

At the bottom, there are buttons for 'Finalize build', 'Report build status', and a log window showing deployment logs.



Recap

A large teal triangle is positioned on the left side of the slide, pointing towards the center. It has a dark teal border and a magenta diagonal line running from its top-right corner to its bottom-left corner.

Conclusion

The what
and why of
IaC

#1

Terraform & ARM Templates

- Predictability & repeatability
- Software-defined infrastructure
- Speed & flexibility
- Managed & Versioned

How to deploy an IoT Data Solution with IaC

Conclusion

#2

Deploy with Terraform & ARM

- Terraform commands
- Terraform expressions
- Terraform variables
- Generating plan & deployment

How to deploy using Azure DevOps

Conclusion

#3

Azure DevOps

- Committing Terraform to Azure DevOps
- Source control
- Pipelines
- Build & Release

Thank You!



Email:

sjoerdborneman@hotmail.com



LinkedIn:

<https://www.linkedin.com/in/sjoerdborneman/>



Twitter:

<https://twitter.com/sjoerdborneman>