INSPARK

Infra-as-Code for your Data Solution in Azure

Terraform & Azure Resource Manager (ARM)

13.10.2020 Sjoerd Borneman

Sjoerd Borneman

Data Engineer/Consultant at **InSpark**

Email: sjoerdborneman@hotmail.com

LinkedIn: https://www.linkedin.com/in/sjoerdborneman/

Twitter: https://twitter.com/sjoerdborneman

INSPARK

Objectives
Presentation

Goal?

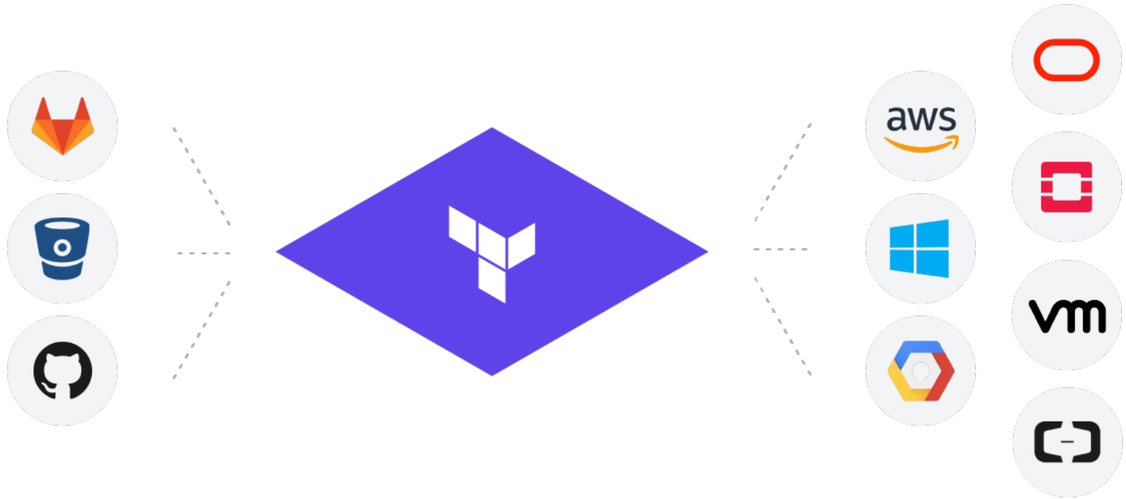
- You know what Infrastructure-as-Code (IaC) is
- Basic understanding of Terraform
- Basic understanding of Azure ARM Templates
- Knowing how to deploy an Azure Data Solution using IaC
- The use of Azure DevOps when deploying an Azure Data Solution using IaC

INSPARK

INSPARK

Infra-as-Code Tools

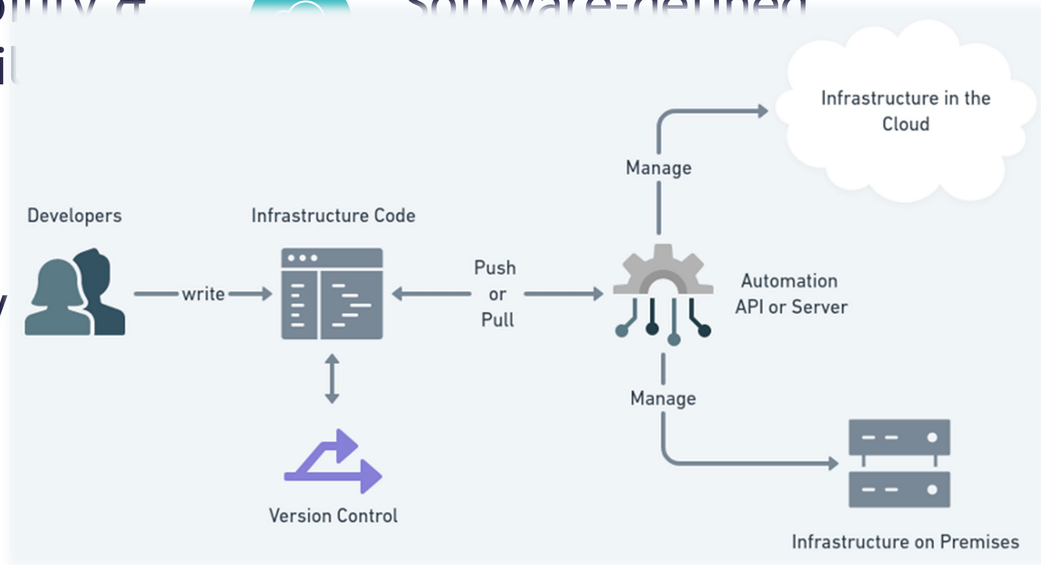
Why Infrastructure-as-Code?



predictability &
repeatabi

Software-defined

speed &
flexibility



HashiCorp
Terraform

Infrastructure-as-Code Technologies

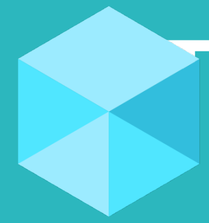
Azure Resource Manager Templates

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

```
{
    "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "serverName": {
            "type": "string",
            "metadata": {
                "description": "The name of the database server."
            }
        },
        "databaseName": {
            "type": "string",
            "metadata": {
                "description": "The name of the database."
            }
        },
        "storageEndpoint": {
            "type": "string",
            "metadata": {
                "description": "The endpoint of the datalake."
            }
        },
        "emailAdresses": {
```

Parameters

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "",
  "apiProfile": "",
  "parameters": {  },
  "variables": {  },
  "functions": [  ],
  "resources": [  ],
  "outputs": {  }
}
```

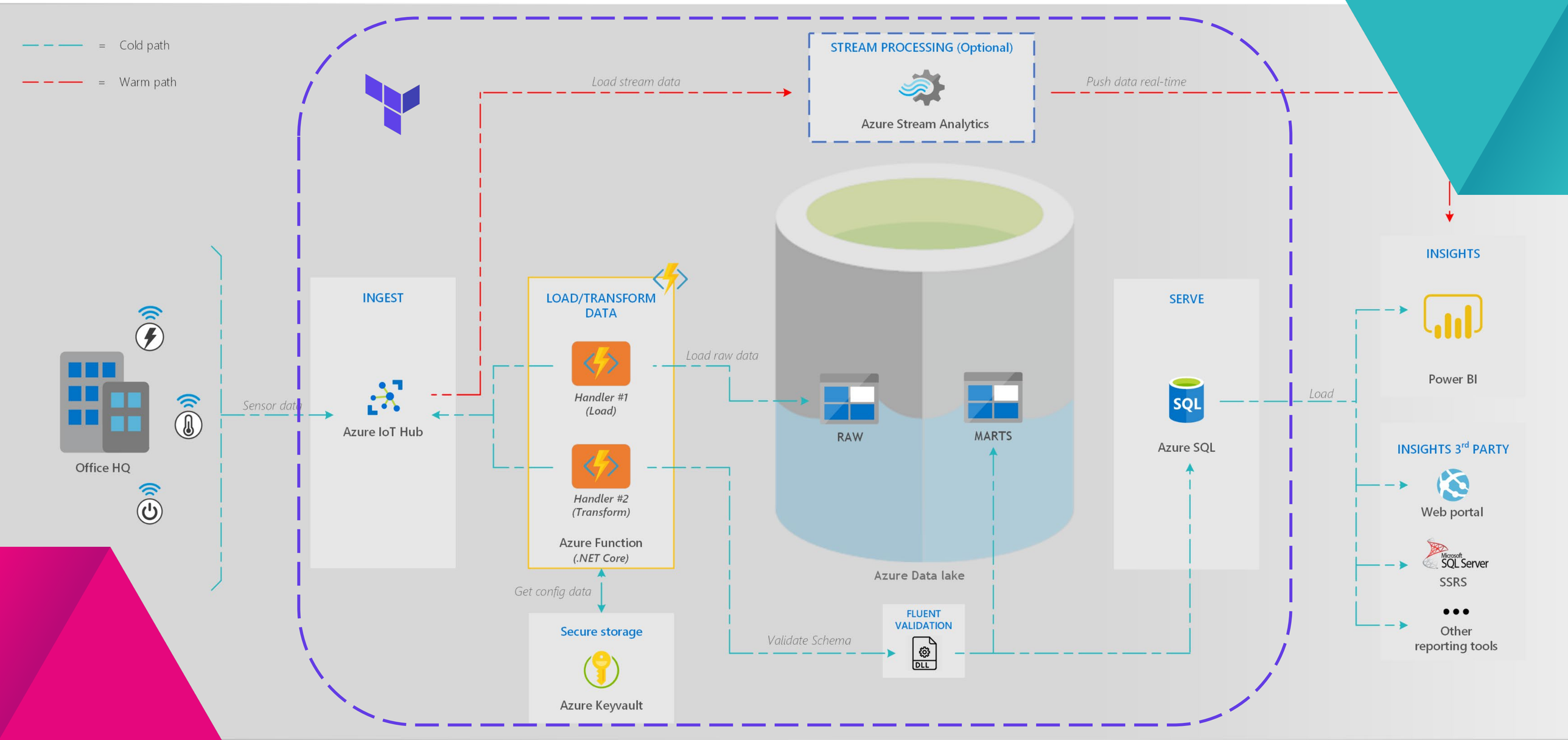
```
                "isEnabled": true,
                "emailSubscriptionAdmins": true,
                "emails": "[split(parameters('emailAdresses'),',')]"
            }
        },
        "dependsOn": [
            "[concat('Microsoft.Sql/servers/', parameters('serverName'), '/databases/', parameters('databaseName'), '/securityAlertPolicies/default')]"
        ]
    }
]
}
```

INSPARK

Next app:

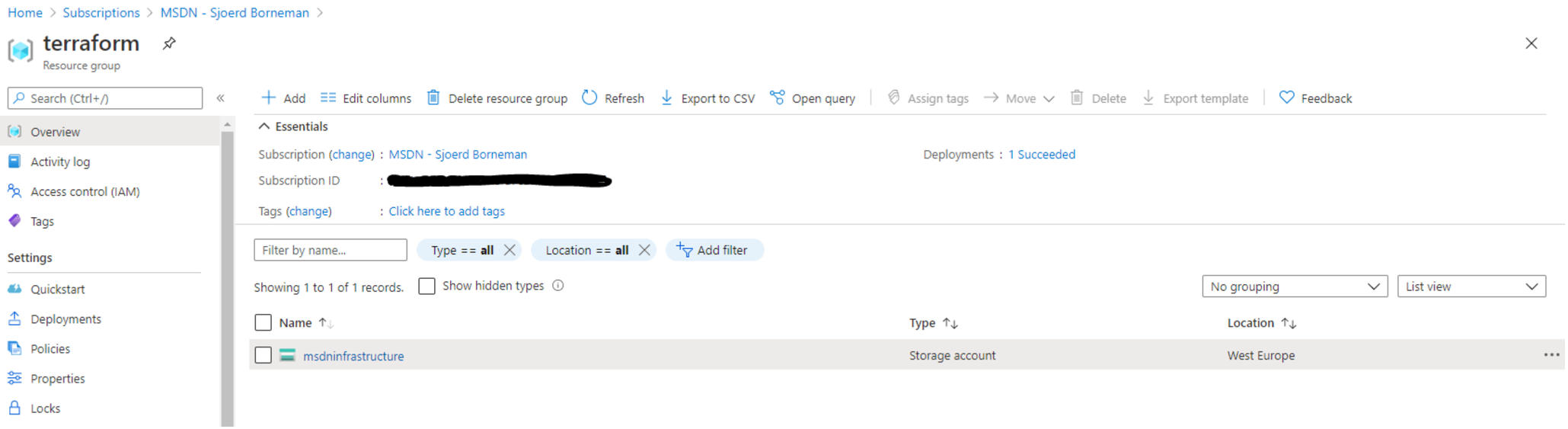
Provisioning with Terraform & ARM

Azure IoT Data Solution



#1 Setup terraform solution

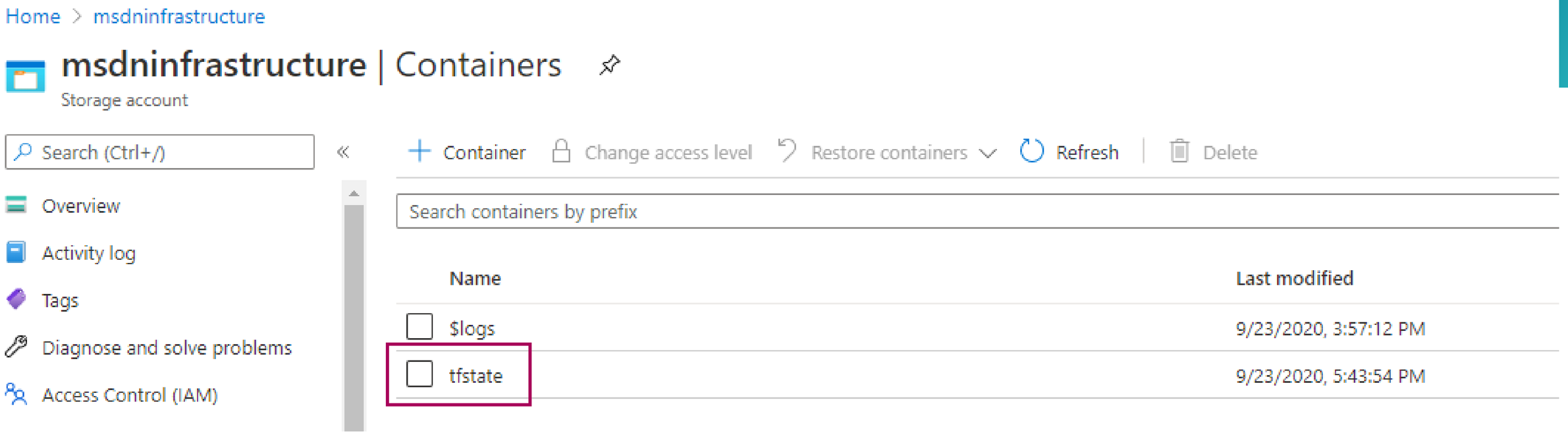
1. Create a storage account in Azure to host your Terraform state files



But what are state files...? 🤔

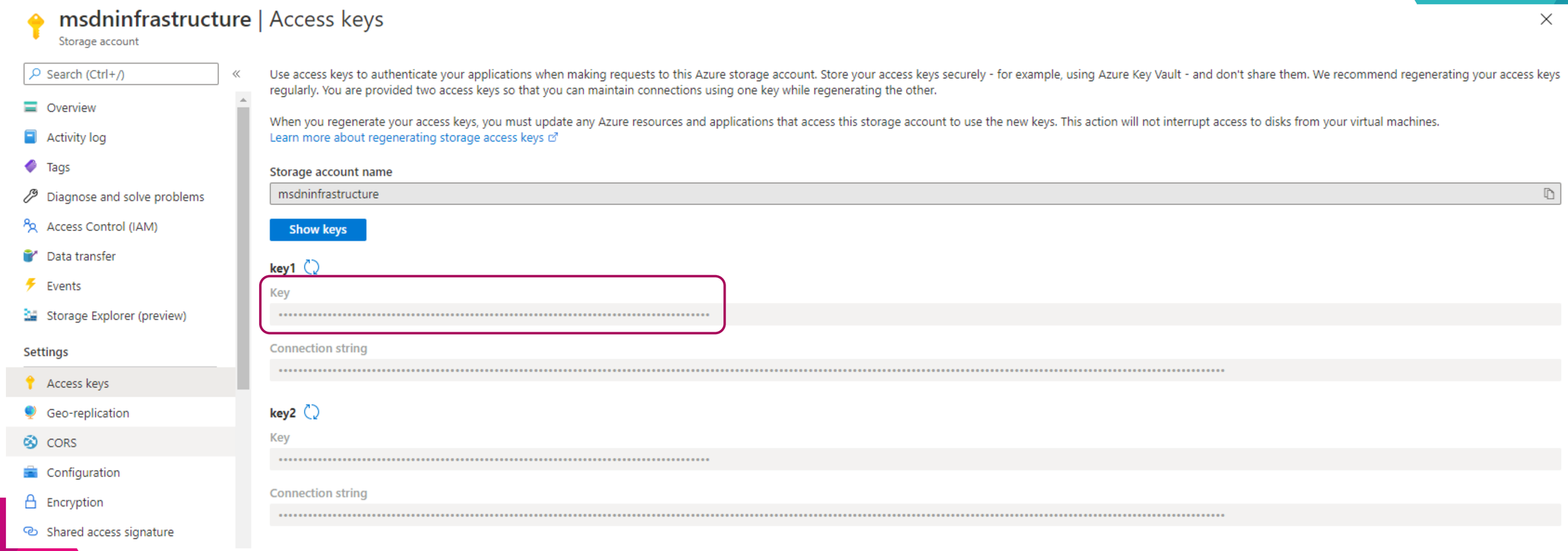
#1 Setup terraform solution

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



#1 Setup terraform solution

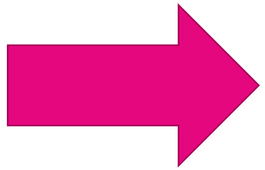
3. Save the access key of the storage account

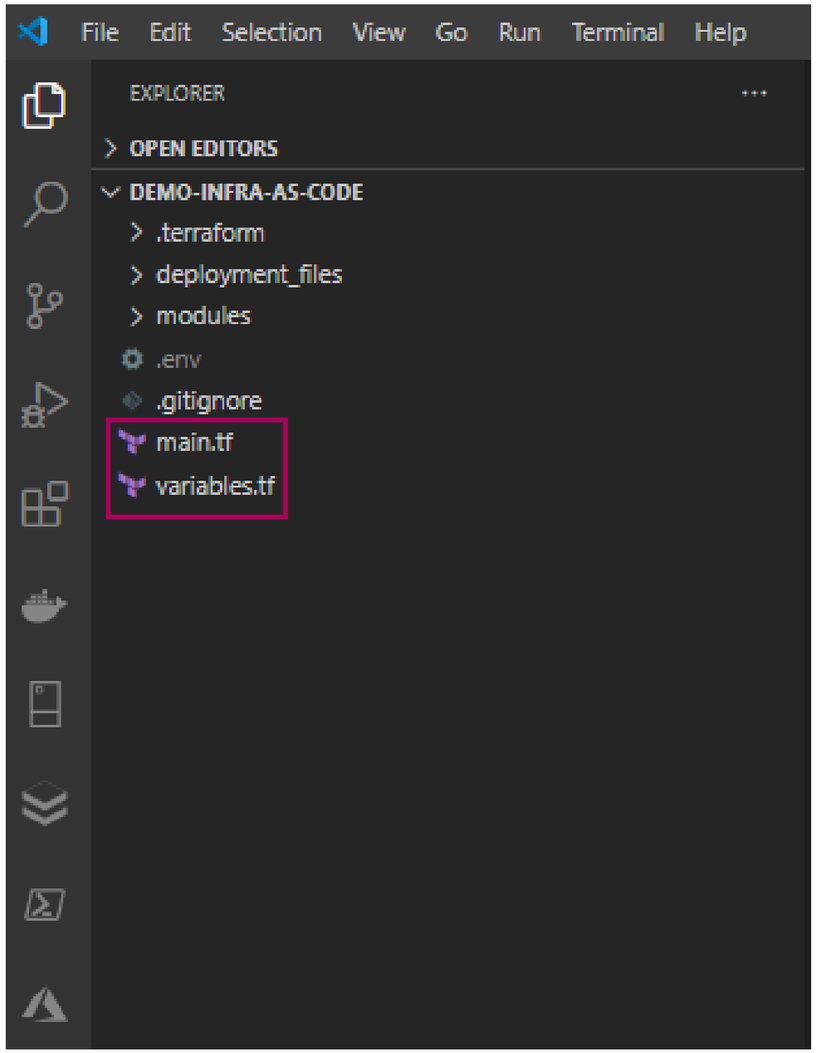


#1 Setup terraform solution

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)

main.tf .env

.env

```
## 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>"

```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

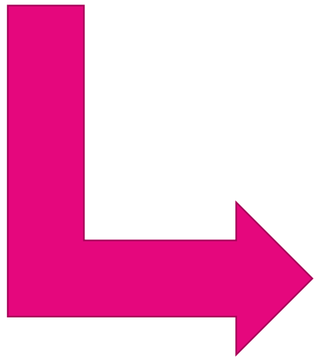
```
PowerShell 7.1.0
Copyright (c) Microsoft Corporation.

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
```



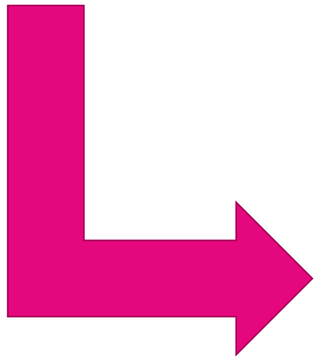
```
PS C:\Azure DevOps\evenementen\Demo-Infra-as-Code> terraform init

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
```



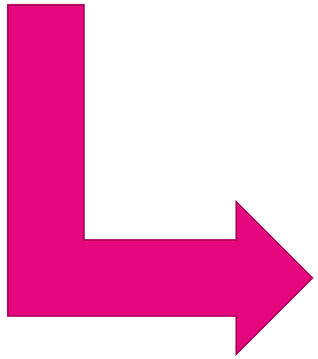
```
OUTPUT  TERMINAL  DEBUG CONSOLE  PROBLEMS

PS C:\Azure DevOps\evenementen\Demo-Infra-as-Code> terraform workspace list
* default
```

#1 Setup terraform solution

7. Initialize the working directory and create a terraform workspace

   Run the following commands:

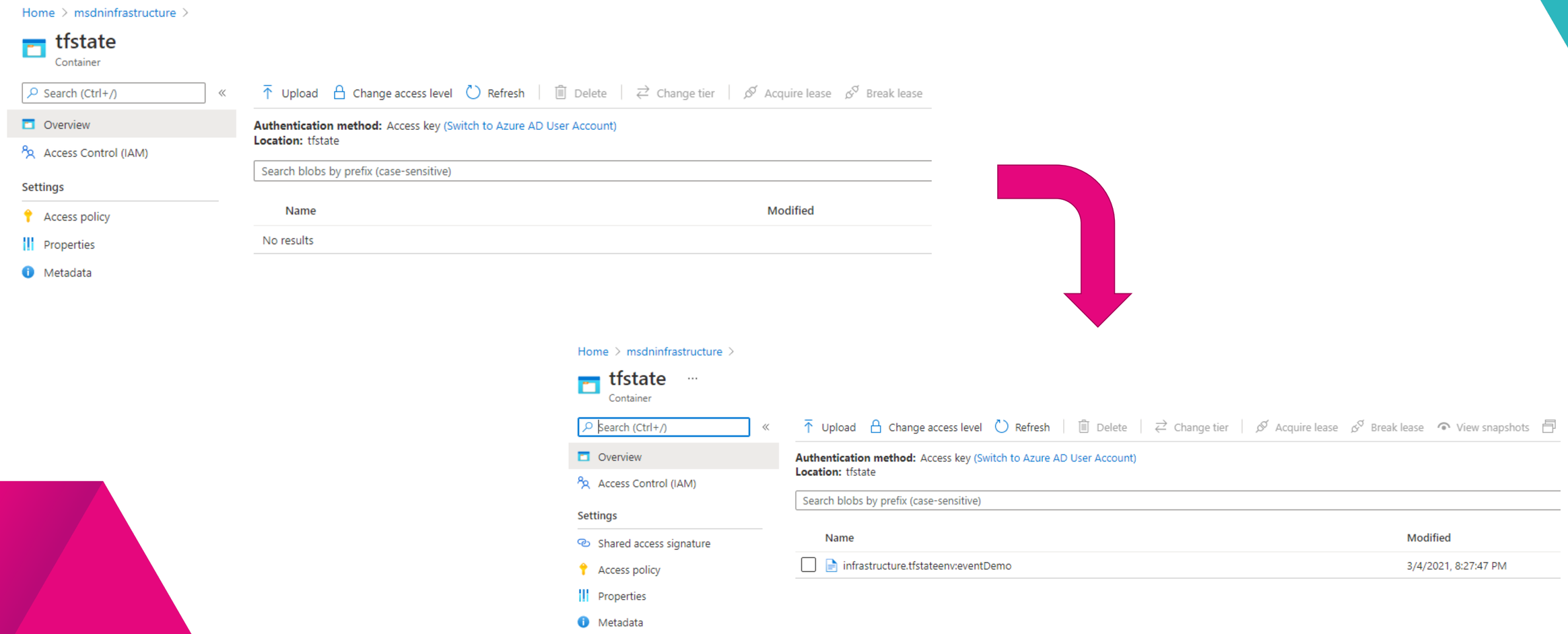
```
terraform workspace new eventDemo
terraform workspace list
```



```
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



#2 Create a variables.tf & .tfvars file

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

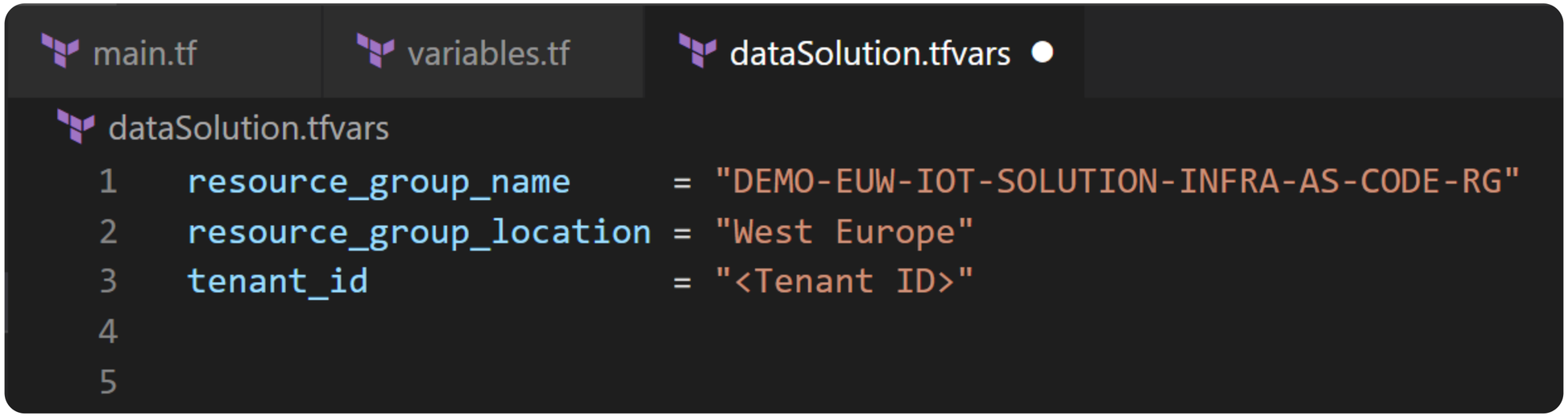
main.tf variables.tf

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:



#3 Define the Azure IoT data services
: Azure Resource Group

main.tf

```
# 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%ssolution%s01"
}

# 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
}
```

Terraform backend type

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

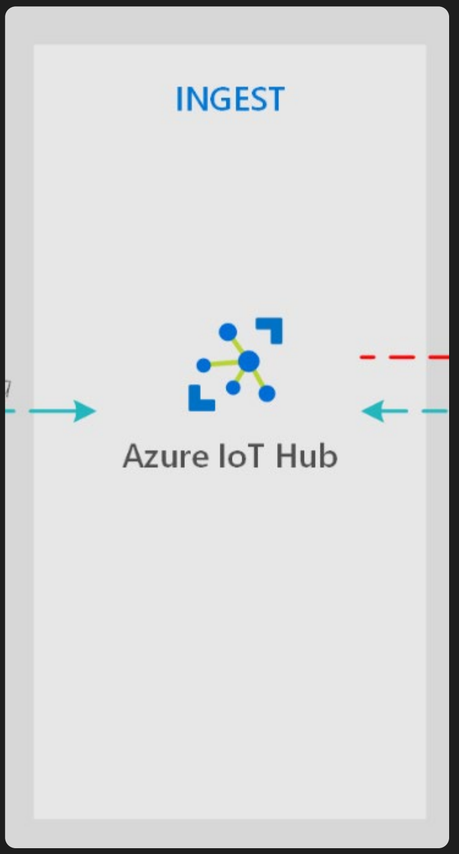
main.tf

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

# 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"
  }
}

locals {
  resource_name_prefix    = "demo%ssolution%s01"
}
```

Stock Keeping Unit

IP Rules

Resource tags



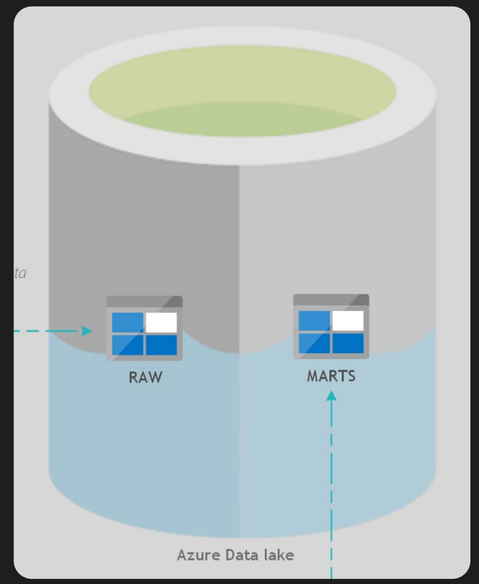
#3 Define the Azure IoT data services
: Azure Data lake Gen2

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.iot_solution_rg.name
  location                 = azurerm_resource_group.iot_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

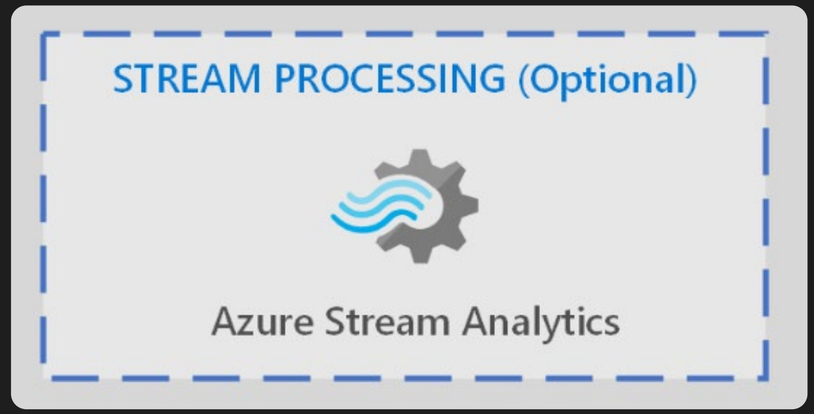


#3 Define the Azure IoT data services
: Azure Stream Analytics

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"
  }

}
```

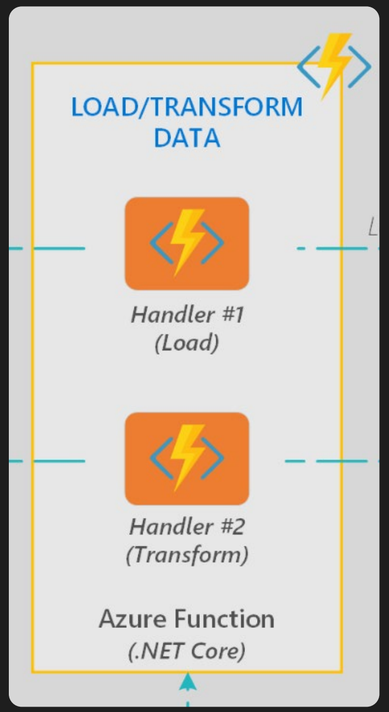
ASA Query



#3 Define the Azure IoT data services
: Azure Function (storage)

main.tf

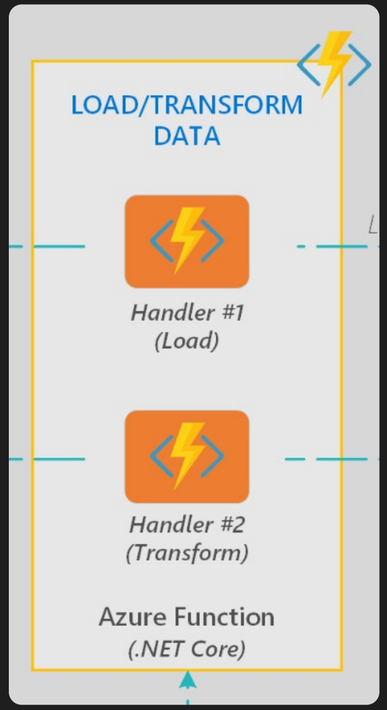
```
# create a storage account for the azure function in Azure with the necessary configurations.
resource "azurerm_storage_account" "functions_storage" {
  name                     = format(local.resource_name_prefix, "euw", "stg") # ← demoeuwsolutionstg01
  resource_group_name      = azurerm_resource_group.iot_solution_rg.name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



#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"
  }
}
```



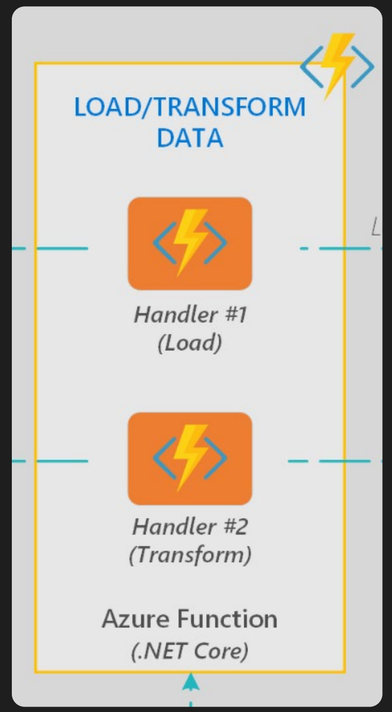
#3 Define the Azure IoT data services
: Azure Function

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



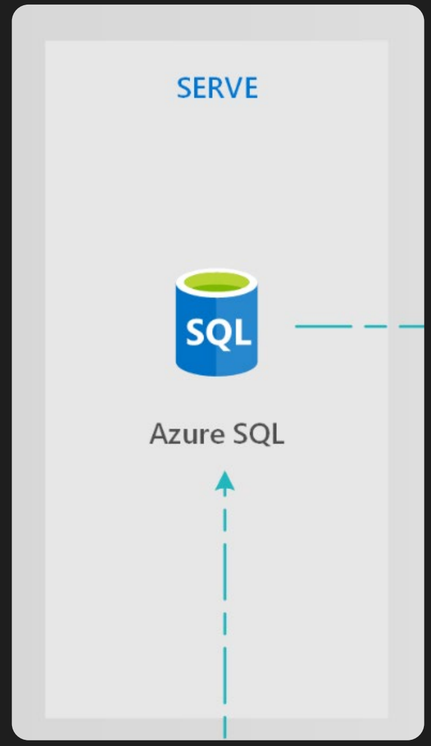
#3 Define the Azure IoT data services
: Azure SQL Server

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"
  }
}
```

SQL Admin

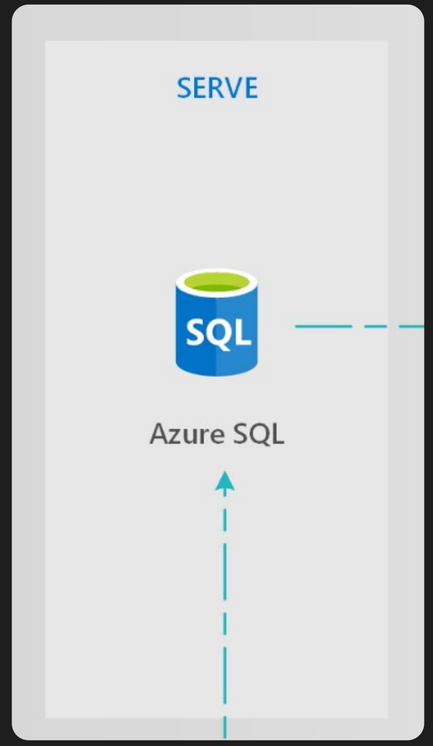
Managed Identity



#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

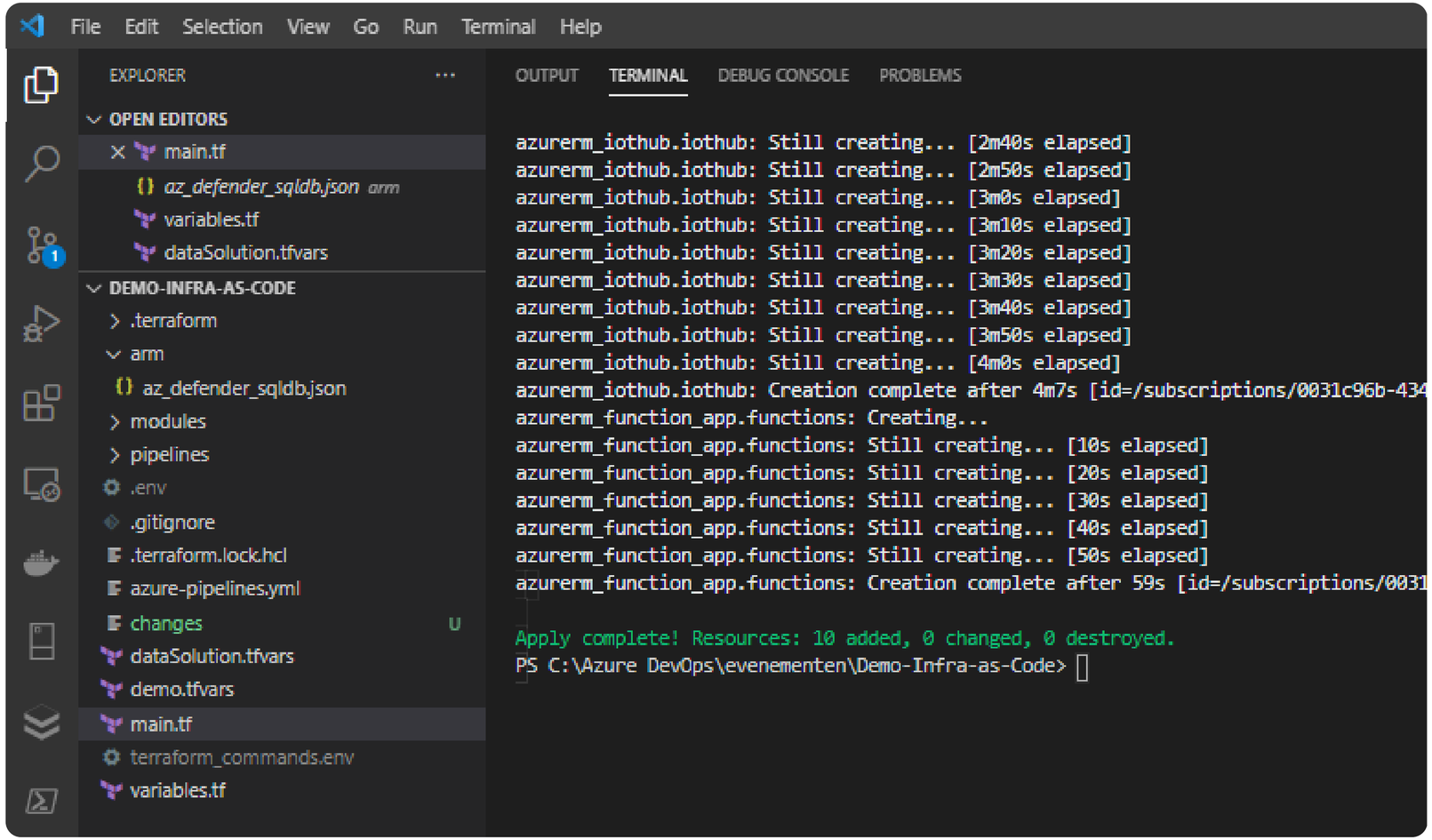
Run the *terraform plan* command to generate and save the execution plan for your configuration

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

#6 Deploy configuration

Run the *terraform apply* command to apply your configration

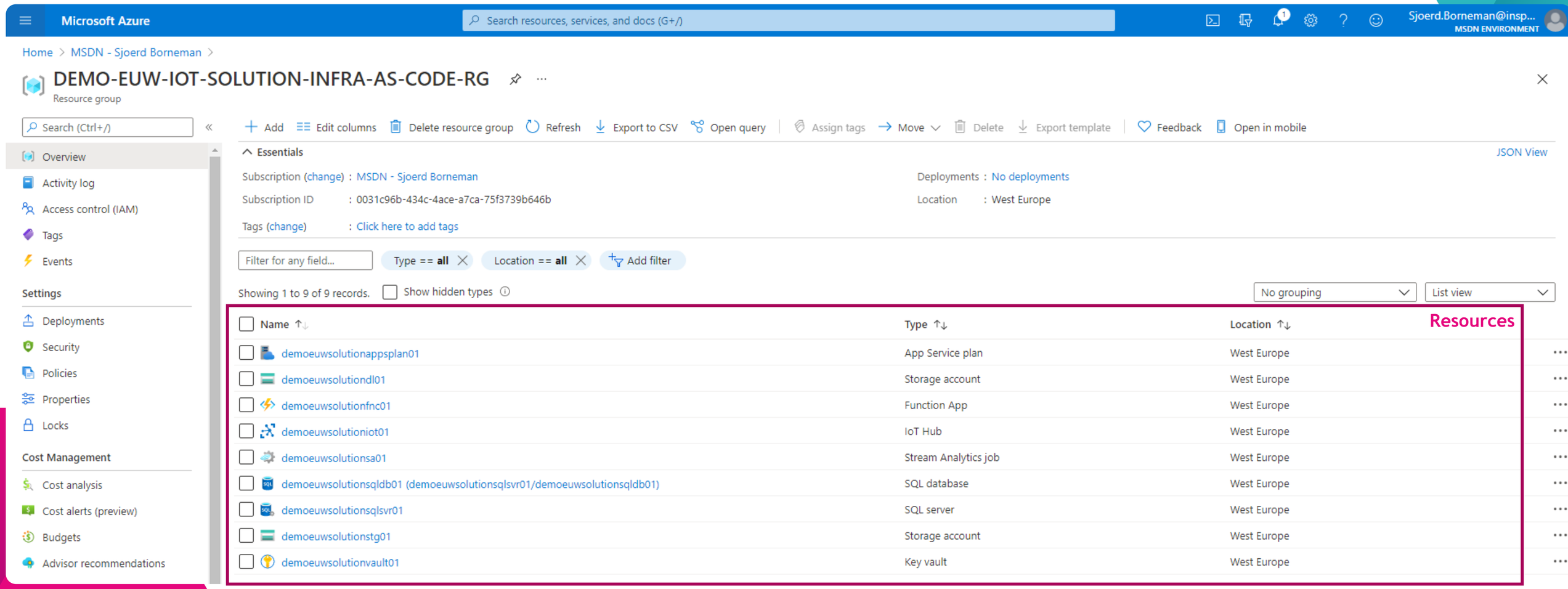
```
terraform apply changes
```

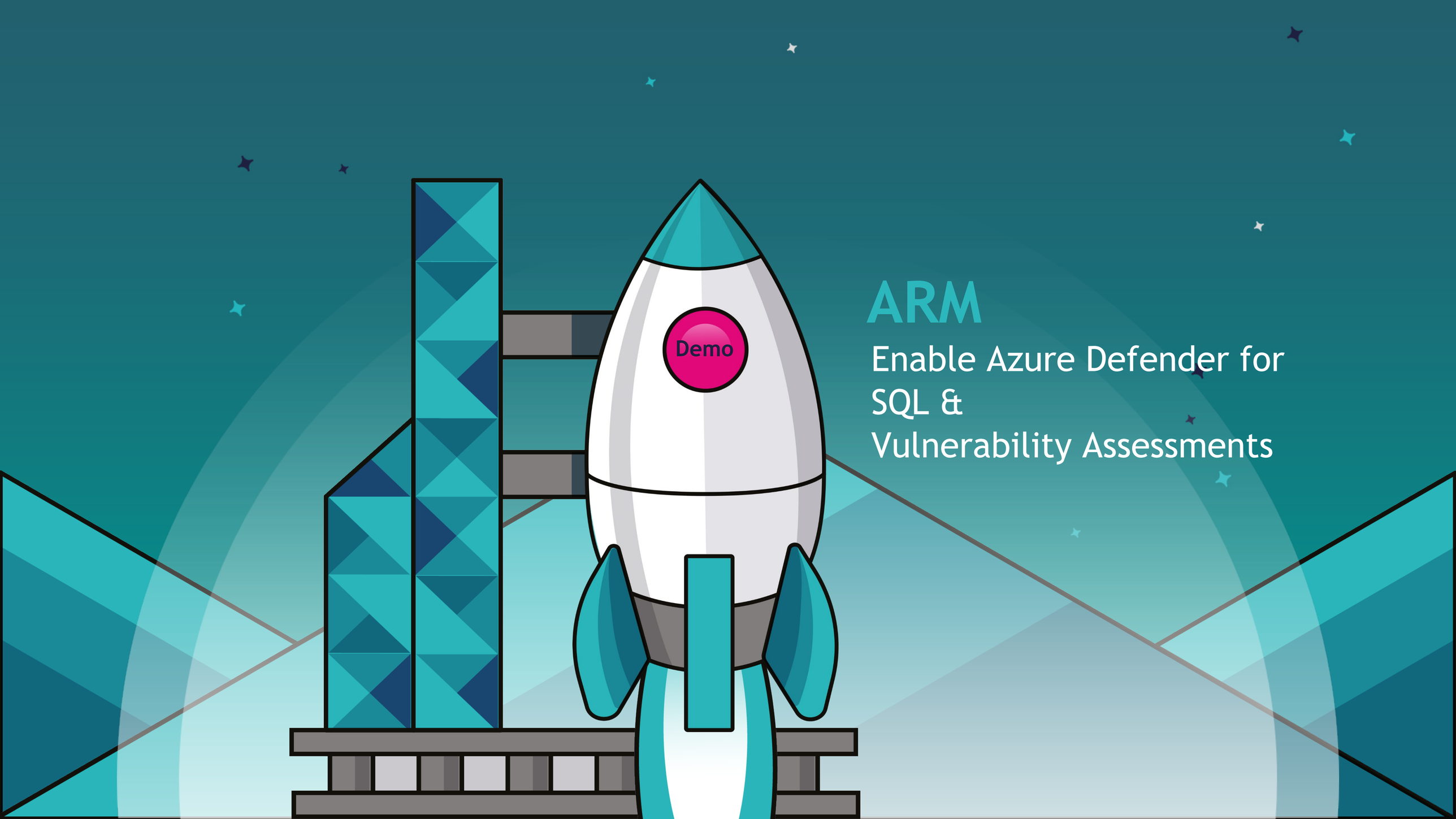


#6 Deploy configuration

Run the *terraform apply* command to apply your configration

```
terraform apply changes
```





ARM

Enable Azure Defender for SQL & Vulnerability Assessments

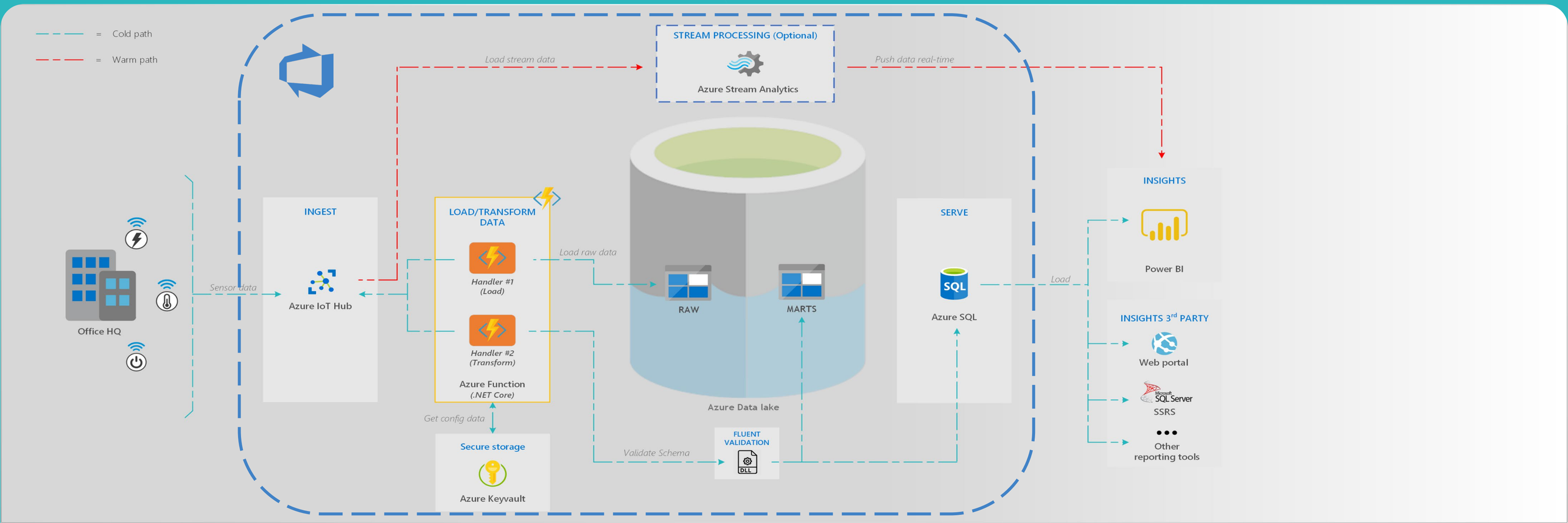
INSPARK

Next.js app:

Provisioning infrastructure using Azure DevOps

CI/CD with Azure DevOps

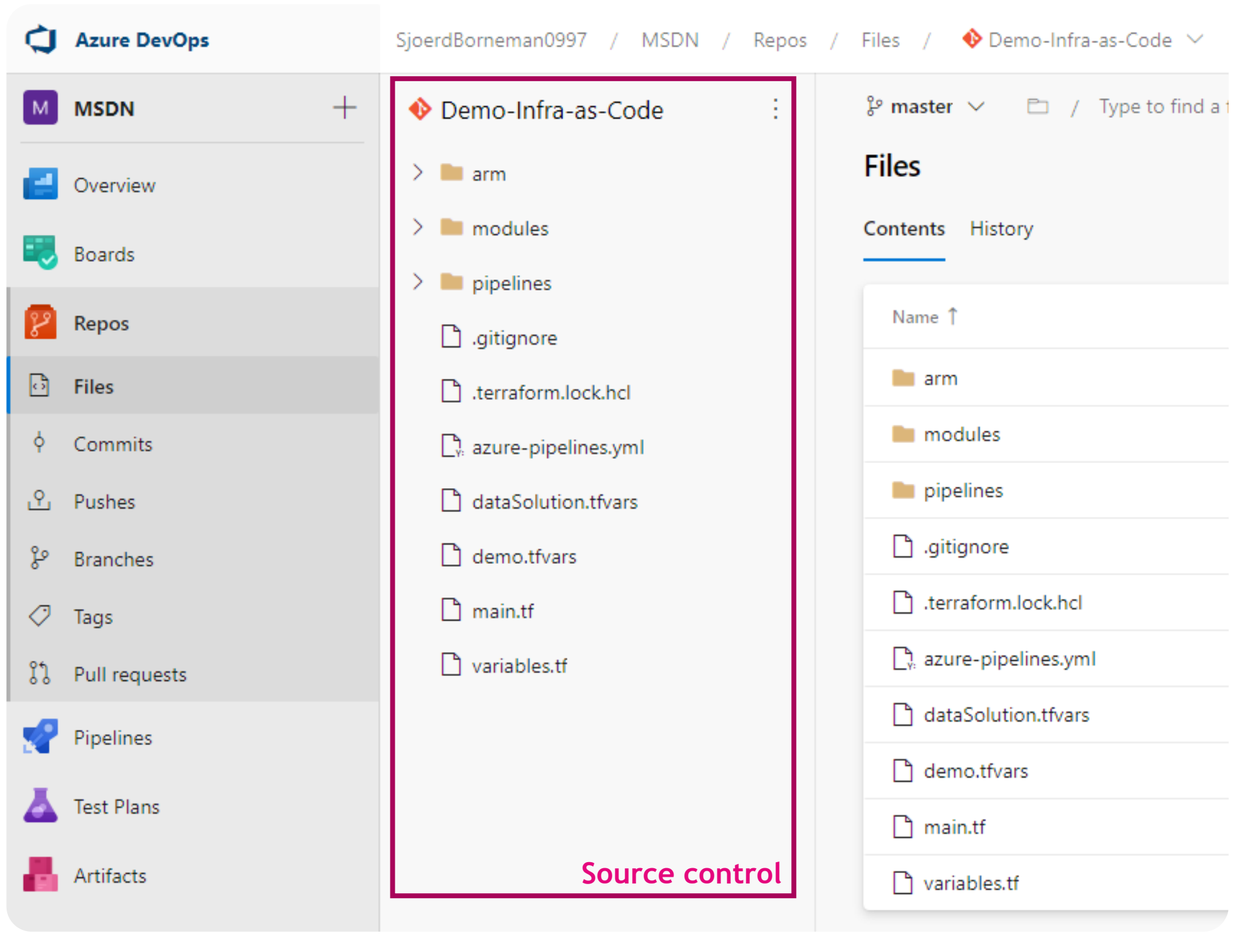
Use Azure DevOps when deploying your Azure Data Solution (Demo)



#1 Commit Terraform configuration files

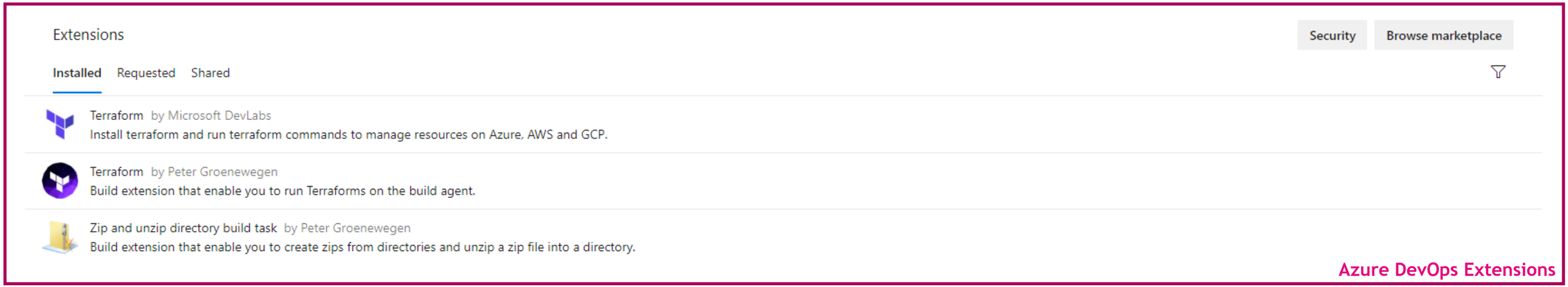
Commit early created configuration files to Azure DevOps

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



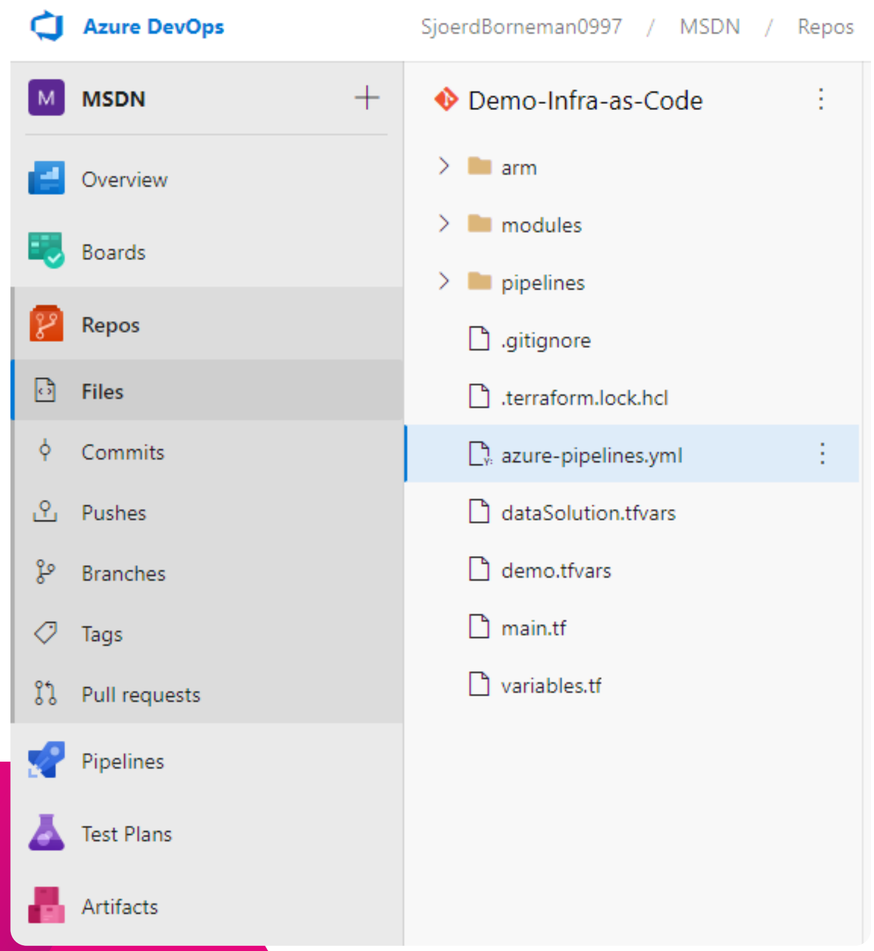
#2 Install Extensions

Install the following dependencies in Azure DevOps



Azure DevOps Extensions

#3 Create Pipelines : Build




```yaml
trigger:
- master
```
trigger

```yaml
pool:
  vmImage: 'windows-latest'
```
pool

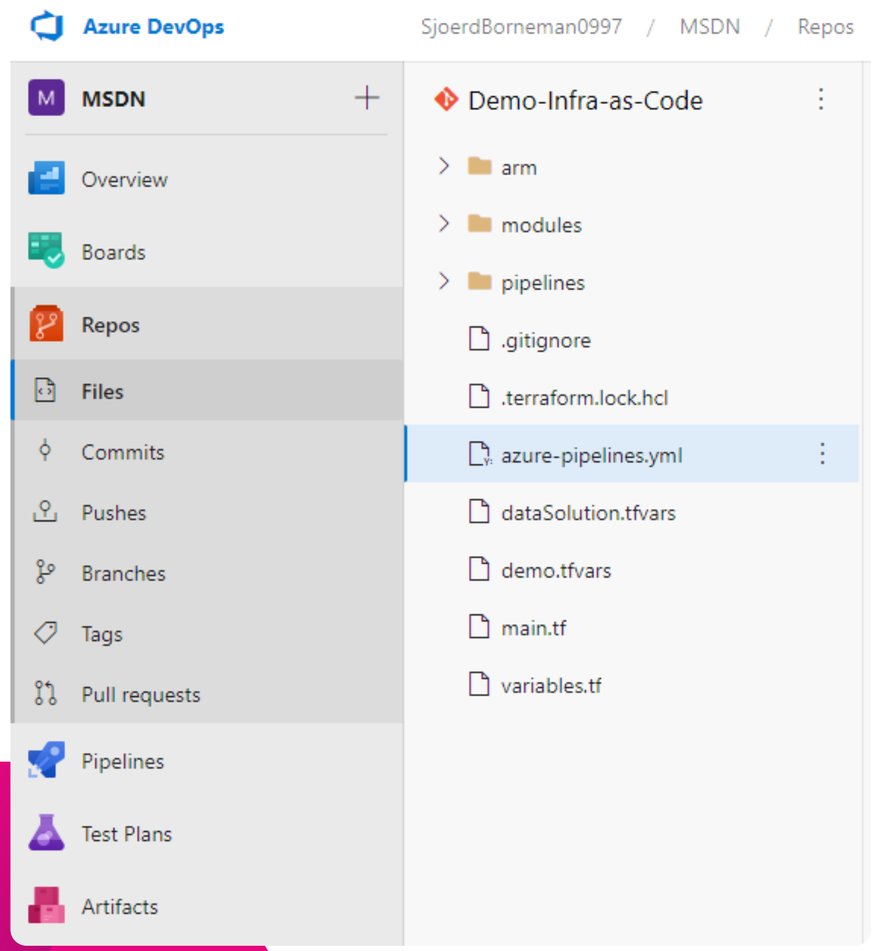
```yaml
variables:
  terraform_workspace: 'eventDemo'
```
variables

```yaml
jobs:
- job: "build_iot_environment"
  displayName: "build iot environment"
```
job: build

job steps: build

```yaml
  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'
```

#3 Create Pipelines : Release




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

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

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

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

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

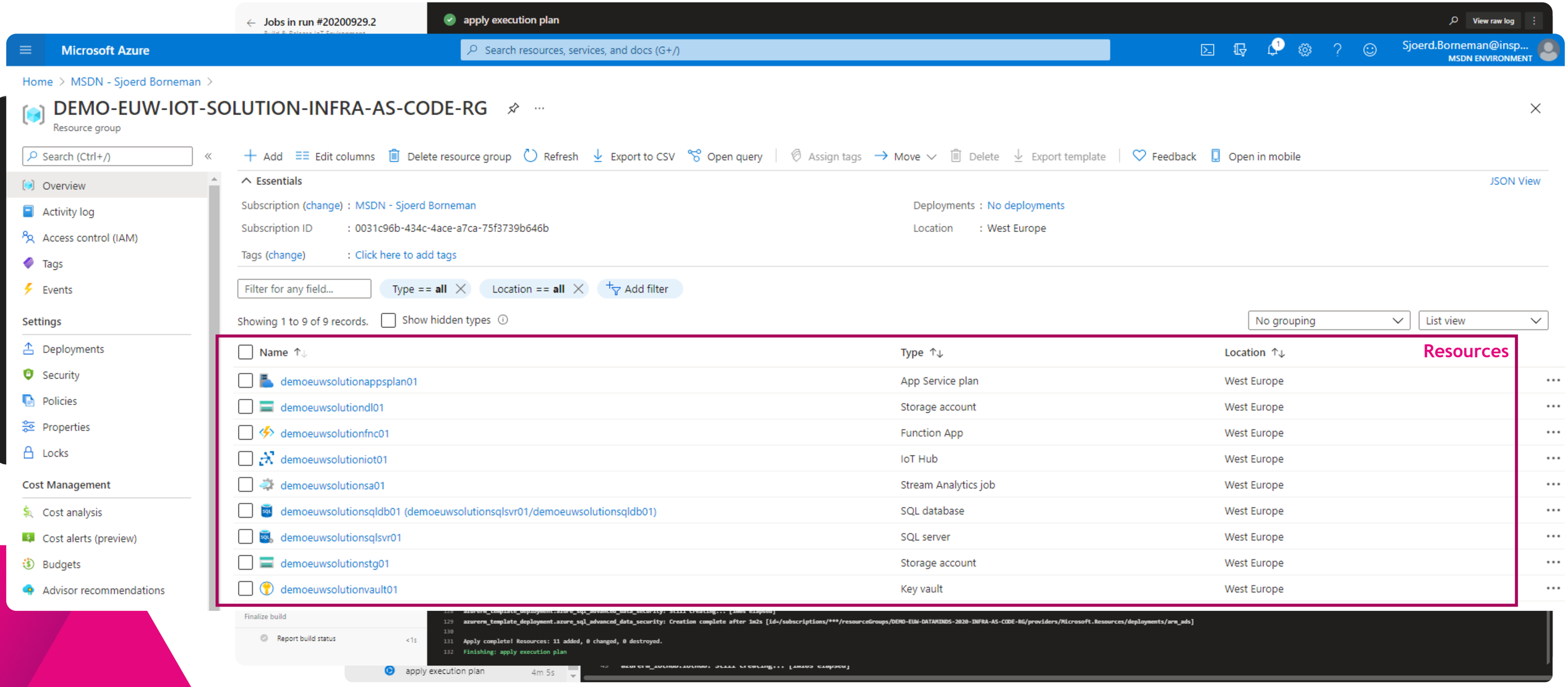
```yaml
        - task: PowerShell@2
          displayName: 'set terraform workspace'
          inputs:
            targetType: 'inline'
            script: |
              terraform workspace select '$(terraform_workspace)'
            workingDirectory: '$(System.DefaultWorkingDirectory)/infrastructure'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

```yaml
        - 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

```yaml
        - 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 pipline



INSPARK

Recap

#1

Conclusion

The what and why of IaC

Terraform & ARM Templates

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

INSPARK

Conclusion

How to deploy an IoT Data Solution with IaC

#2

Deploy with Terraform & ARM

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

INSPARK

Conclusion

How to deploy using Azure DevOps

#3

Azure DevOps

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

INSPARK

dataMinds

Thank You!


Email: sjoerdborneman@hotmail.com


LinkedIn: https://www.linkedin.com/in/sjoerdborneman/


Twitter: https://twitter.com/sjoerdborneman

INSPARK