Accelerators.wiki
Filter pages by title
Creating Jumpo...
DeploymentEngine
Log/Log & Prove...
ProvisioningEngine
Multistaging Pi...
Multi-environm...
SonarCloud Int...
Templates
PowerShell
New page

ProvisioningEngine

Lai Kumar Reddy T · Apr 29

Follow Edit

Contents

- Introduction
- High-Level Diagram
- Technical Overview
- How to use Provisioning Engine
- Consumer pipeline description
- Variables-Description
  - Variable-file.yml
- Common- infra -deploy Description
  - Common-Infra-deploy.yaml
- Validate-deploy-terraform Description
  - Validate-Copy-template.Yaml
- Validate-deploy-tfe Description
  - Validate-deploy-tfe.yaml
- Execute-powershell-template.yaml
- Validate-deploy-terraform Description

Introduction

Provisioning Engine is a tool which can deploy the services in Azure DevopsUsing the templates like (Arm,Bicep, terraform) with pre deploy and postdeploy.

High-Level Diagram

- **Source Code**: Source Code mainly YAML files has been managed in Azure DevOpsADO.
- **Build Agent**: Depending on the Solution, customerUser need to select agent with specific configuration. For Demo purpose we have used Microsoft hosted windows agent.

How to use Provisioning Engine

As per tool design,provisioning engine as generic tool is separated out and its code(YAML scripts) has checked in to ProvisioningEngineRepo.

From Customer point of view, customer will have to make changes in yaml files their repository and extend pipeline to use Provisioning Engine.

Note: For reference we have kept standard customer yaml based pipeline(yaml) which can be used by customer to create customer pipeline from their repository.

If customer's scenario is matching with standard templates then they can just take these file and add to their repository.

Consumer-pipeline description

```
name: $(Build)

trigger:
  batch: true
  branches:
    include:
      - 'master'

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

The **name: $(Build)**: Build/Version number is auto incrementing number assigned to pipeline once executed. This number will be decided by project team which fits for the purpose. Refer Link for more combinationsvariables

The **trigger** is used to run the pipeline automatically. Azure Pipelines supports various types of actions. Depending on your pipeline type.

If you set **batch: true** when a pipeline is running, the system waits until the run is completed, then starts another run with all changes that have not yet been built.

The **branches** indicates which branch should trigger the pipeline to run.

The **include** is a keyword used to specify additional files to include in the pipeline configuration.

The **agentpools** Azure Pipelines provides a predefined named as Microsoft-hosted agents and Self-hosted agents .This agent will depends on solutions.

Properties	Description
batch	(boolean)Whether to batch changes per branch.
branches	(includeExcludeFilter)Branch names to include or exclude for triggering a run.

The Resources is defined list of repositories resources referenced by the pipeline. The repositories represents list of repository referenced by the pipeline.

```
resources:
  repositories:
    - repository: ProvisioningEngine
      name: Accelerators/ProvisioningEngine
      type: git
      ref: master

    # - repository: consumerTemplateRepo
    # name: Customer/CustomerTemplates
    # type: git
    # testing tag
    # ref: master

    - repository: ProvisioningEngineTemplates
      name: Accelerators/ProvisioningEngineTemplates
      type: git
      ref: master
```

properties	Description
repository	[String]:Required as first property. Alias for the specified repository. Acceptable values:[,A-Za-z0-9].
name	[string]The name of the triggering repository ?
type	[string]The type of the triggering repository ? .Git : Git repository hosted on an external server. Type of repository: git, github, github enterprise, and bitbucket.
reference	[String]:Required as first property. Alias for the specified repository. Acceptable values:[,A-Za-z0-9].

Type
Azure pipelines support the following values for the repository type: git, GitHub, and bitbucket. The git type refers to Azure Repos Git repos. If you specify type: git, the name value refers to the name of an Azure Repos Git repository.
Note: If your pipeline is in the same Azure DevOps project as the repository, for example a repository named tools, you reference it using name: tools.
If your pipeline is in the same Azure DevOps organization as the repository, but in a different Azure DevOps project, for example a project named Tools Project you must qualify the repository name with the project name: name: Tools Project/tools.

Variables-Description

```
variables:
  - template: /consumer-project/azure-pipelines/vars/variables.yaml@self
  - template: /consumer-project/azure-pipelines/vars/common.yaml@self
```

The Variables file pairing from Consumer Project in the Same Repo.
In variable file :

Variable-file.yml

```
variables:
  - azureResourceManagerConnection: TobeUpdatedLater
  - subscriptionId: TobeUpdatedLater
  - location: TobeUpdatedLater
  - resourceGroupName: TobeUpdatedLater
  - rgLocation: TobeUpdatedLater
  - serviceCode: TobeUpdatedLater
  - deploy: TobeUpdatedLater
  - templateType: TobeUpdatedLater
  - servicetoDeploy: TobeUpdatedLater
  - templateRepositoryName: TobeUpdatedLater
  - provisioningEnvironment: TobeUpdatedLater

  - kkpService
  - appPlanUsedNumber: TobeUpdatedLater
  - appPlanIsHeadundant: TobeUpdatedLater
  - appPlanNumber: TobeUpdatedLater
```

Common- infra- deploy Description

Extend from a template with resources.You can a use extends to extend from a template in your Azure pipeline that contains resources. There are Several Condition in Common-infra-deploy file as Shown below;
Conditions
1.If run script is 'false' and not equal template type 'terraform'.
2.If run script is 'false' and equal template type 'terraform'.
3.If run script is 'false' and equal template type 'terraform'.

```
extends:
  template: /pipelines/templates/common-infra-deploy.yaml@ProvisioningEngine
  parameters:
    ...
```

This is the pattern folder that extended form the customer -pipeline templates.

Common-infra-deploy.yaml

There are so many stages with Conditions in this file. According to solutions it will go to the next step.
1.If run script is **Terraform** & **Deploy** and it is not equal to **template** & **terraform**.(ConditionsThen,It will go to the stage Validate and run the Validate-copy-template.yml or Black Condition is false.Then it will go the other condition.

```
stages:
  - ${{ if and(eq(parameters.runscript, false), ne(parameters.templatetype, 'terraform') )}}:

  - stage: Validate
    jobs:
      - template: job-templates/validate-copy-template.yaml
```

properties	Description
stage	[string] Required as first property. It is ID of the stage.For more reference ?
jobs	which makeup the stage.For more reference ?
template	Stage related information passed from pipeline when path of the template is given

Validate-Copy-template Description

This template pairing other template validate-deploy-tfe.yaml

- **Properties**:
  1. The **jobs** property defines number of job .Each job represents a specific stage in the pipeline, and is responsible for performing a particular set of tasks.Link for more reference ?
  2. The **steps** property refer the individual task that we need perform the stage of the pipeline .Each step in the pipeline performs a specific function.Link for more reference ?
  3. In this Solution they used each Condition with **split** it is used to deploy one or more services.
  4. **Template** is pairing other template path.

Validate-Copy-template.file

```
jobs:
  - job: Validate
    steps:
      - checkout: $(j2_parameters.repositoryToCheckout )
      - ${{ each service in split(parameters.services, ',') }}:
        - ${{ each template in parameters.templates }}:
            - template: /pipelines/templates/step-templates/validate-deploy-template.yaml
```

Validate-deploy-tfe Description

This file is used to validate and deploy services into Azure Resource Manager depends on the condition.

Validate-deploy-tfe.yaml

Tasks are the building blocks of a pipeline.

1. The first task will replace tokens in the template according to the parameters.
   There are two conditions
   1.If parameters.deploy.false , the template will not deploy into the Azure Resource group.
   2.If parameters.templateType.'terraf , The template type is terraform below task will performthe Azure CLI will performs:It will compile and convert to Arm and next task is Deploy templatesJson.It depends on deploymentmode is Validation. It will validate the template.

```
- ${{ if eq(parameters.templatetype, 'terraform') }}:
  - task: replaceTokens@6
    displayName: Replace Tokens
    inputs:
      targetFiles: $(j2_parameters.tmpdir)/*.j2
      tokenPrefix: '{{'
      tokenSuffix: '}}'
      writeFolder: '$(j2_parameters.tmpdir)'
      actionOnMissing: warn
      keepToken: true
      enableTransforms: false
      enableRecursion: false
      keepFoldersAndBlankFiles: false
      emptyValue: ''
      encoding: auto
      writeFlags: '-f -n -q'
      replacementMarkers: '{{ parameters.replacmentMarkers }}'
      keepRootFolder: true
      transformationRules: '-s -p -v -x -y -ea -f -t -ic -iv -iy -ef -tl -tr -tc -tv -ty -ecp -ecl -ecv -ecy -ecf -ecr -ecc -ecv -ecy -ecf -ecr -ecc -ecv -ecy -ecf -ecr -ecc
```

```
    displayName: AzurePowerShell :'${{ script.displayName }}'
    inputs:
      azureSubscription: '${{ parameters.azureResourceManagerConnection }}'
      ScriptType: 'FilePath'
      ScriptPath: '${{ script.scriptPath }}'
      ScriptArguments: '${{ script.scriptArguments }}'
      azurePowerShellVersion: 'LatestVersion'


- ${{ if or(eq(script.Type, 'PowerShell'), eq(script.Type, 'Core')) }}:
  - task: PowerShell@2
    displayName: 'Powershell: ${{ script.displayName }}'
    inputs:
      filePath: '${{ script.scriptPath }}'
      arguments: '${{ script.scriptArguments }}'

- ${{ if eq(script.Type, 'AzureCLI') }}:
  - task: AzureCLI@2
    displayName: 'Azure CLI: ${{ script.displayName }}'
    inputs:
      azureSubscription: '${{ parameters.azureResourceManagerConnection }}'
      scriptType: 'ps'
      scriptLocation: 'scriptPath'
      scriptPath: '${{ script.scriptPath }}'
      arguments: '${{ script.scriptArguments }}'
```

- **Third condition**
  This is the third condition if runscript is 'false' and it is equal to template type 'terraform',if this Condition is 'true',then it will go to the provisioning stage and deploy the environment once. and it will go to the Validate -terrform file.
  Common-infra-deploy.file

```
- ${{ if and(eq(parameters.runScript, false), eq(parameters.templatetype, 'terraform') )}}:

  - stage: Provisioning_${{ parameters.provisioningEnvironment }}
    jobs:
      - deployment: Deploy${{ parameters.provisioningEnvironment }}
        environment: ${{ parameters.provisioningEnvironment }}
        workspace:
          clean: all
        timeoutInMinutes: 90
        strategy:
          runOnce:
            deploy:
              steps:
                - checkout: ${{ parameters.repositoryName }}
```

In this there is a Condition if parameter.ExecutePowershell is true and not equal to length of pre deployScripts is '0'then it will go the template path. and it will go the execute-powershell-template,and pre-deploy the the scripts .

```
- checkout: ${{ parameters.repositoryName }}
- ${{ if and(eq(parameters.executePowershell, true), ne(length(parameters.preDeployScriptsList), 0) ) }}:
  - template: step-templates/execute-powershell-template.yaml
    parameters:
      azureResourceManagerConnection: ${{ variables.azureResourceManagerConnection }}
      scriptsList: ${{ parameters.preDeployScriptsList }}
```

```
- template: step-templates/validate-deploy-terraform.yaml
  parameters:
    deploy: ${{ parameters.deploy }}
    templates: ${{ parameters.templates }}
    services: ${{ parameters.servicesToDeploy }}
    connection: $(azureResourceManagerConnection)

- ${{ if and(eq(parameters.executePowershell, true), ne(length(parameters.postDeployScriptsList), 0) ) }}:
  - template: step-templates/execute-powershell-template.yaml
    parameters:
      azureResourceManagerConnection: ${{ variables.azureResourceManagerConnection }}
      scriptsList: ${{ parameters.postDeployScriptsList }}
```

Validate-deploy-terraform Description

This file parsing from common -infra-deploy file. when Condition is satified.

```
steps:
  - task: TerraformInstaller@0
    inputs:
      terraformVersion: 'latest'

  - ${{ if eq(parameters.deploy, true) }}:
    - ${{ each service in split(parameters.services, ',') }}:
      - ${{ each template in parameters.templates }}:
        - task: TerraformTaskV3@3
          displayName: Terraform-init-${{ service }}
          inputs:
            provider: 'azurerm'
            command: 'init'
            workingDirectory: '$(System.DefaultWorkingDirectory)/${{ template.templatePath }}/${{ service }}'
            backendServiceArm: ${{ parameters.connection }}
            backendAzureRmResourceGroupName: ${{ template.resourceGroupName }}
            backendAzureRmStorageAccountName: 'terraformstatebkp'
            backendAzureRmContainerName: 'terraform'
            backendAzureRmKey: 'tf/terraform.tfstate'

        - task: replacetokens@5
          inputs:
            targetFiles: '$(System.DefaultWorkingDirectory)/${{ template.templatePath }}/${{ service }}/terraform.tfvars'
            encoding: 'auto'
            tokenPattern: 'default'
            writeBOM: true
            actionOnMissing: 'warn'
            keepToken: false
            actionOnNoFiles: 'continue'
            enableTransforms: false
            enableRecursion: false
            useLegacyPattern: false
            enableTelemetry: true

        - task: TerraformTaskV3@3
          displayName: Terraform-plan-${{ service }}
          inputs:
            provider: 'azurerm'
            command: 'plan'
            workingDirectory: '$(System.DefaultWorkingDirectory)/${{ template.templatePath }}/${{ service }}'
            environmentServiceNameAzureRM: ${{ parameters.connection }}
        - task: TerraformTaskV3@3
          displayName: Terraform-apply-${{ service }}
          inputs:
```

After Validate the terraform if parameter.excuetePowershell is true not equal to the length parameter.postdeployscriptsList is '0' then it will execute the power shell and post deploy the scripts.

```
- template: step-templates/validate-deploy-terraform.yaml
  parameters:
    deploy: ${{ parameters.deploy }}
    templates: ${{ parameters.templates }}
    services: ${{ parameters.servicesToDeploy }}
    connection: $(azureResourceManagerConnection)

- ${{ if and(eq(parameters.executePowershell, true), ne(length(parameters.postDeployScriptsList), 0) ) }}:
  - template: step-templates/execute-powershell-template.yaml
    parameters:
      azureResourceManagerConnection: ${{ variables.azureResourceManagerConnection }}
      scriptsList: ${{ parameters.postDeployScriptsList }}
```

14 visits in last 30 days

ST

Add a comment...