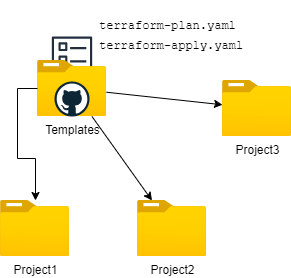
**Simplifying Terraform Pipelines with Azure DevOps Templates**

As part of my job, I’m always looking for ways to optimize the processes for the companies I’m working for. In a infrastructure deployment, combining the power of Azure DevOps with Terraform is a game-changer. In this post, I will show you a cool technique: using Azure DevOps templates to make Terraform pipelines to reduce code repetition. Let’s see how it works!

# What are Azure DevOps Pipelines templates?

Templates are reusable definitions for your CI/CD (Continuous Integration/Continuous Delivery) pipelines. They allow you to define a pipeline configuration once and then reuse it across multiple projects or repositories, making it easier to maintain consistent build and deployment processes.



# Building a Terraform Pipeline Template

To illustrate the concept, let’s explore a Terraform pipeline template that orchestrates the planning and deployment of infrastructure. This template leverages the power of Azure DevOps to streamline the process, making it more manageable and repeatable. This is a template for the PLAN operation:

# terraform-plan-operation.yaml  
  
parameters:  
- name: connectionName  
 type: string  
  
- name: environmentName  
 type: string  
 default: 'dev'  
  
# ... (Other parameters)  
  
steps:  
 - task: TerraformInstaller@1  
 # ... (Terraform installation configuration)  
  
 - task: TerraformTaskV3@3  
 # ... (Terraform initialization configuration)  
  
 - task: Bash@3  
 # ... (Terraform workspace creation)  
  
 - task: TerraformTaskV3@3  
 # ... (Terraform workspace selection)  
  
 - task: TerraformTaskV3@3  
 # ... (Terraform plan configuration)

Another example could be a template for the apply operation:

# terraform-apply-operation.yaml  
  
parameters:  
- name: connectionName  
 type: string  
  
- name: environmentName  
 type: string  
 default: 'dev'  
  
# ... (Other parameters)  
  
steps:  
 - task: TerraformInstaller@1  
 # ... (Terraform installation configuration)  
  
 - task: TerraformTaskV3@3  
 # ... (Terraform initialization configuration)  
  
 - task: Bash@3  
 # ... (Terraform workspace creation)  
  
 - task: TerraformTaskV3@3  
 # ... (Terraform workspace selection)  
  
 - task: TerraformTaskV3@3  
 # ... (Terraform apply configuration)

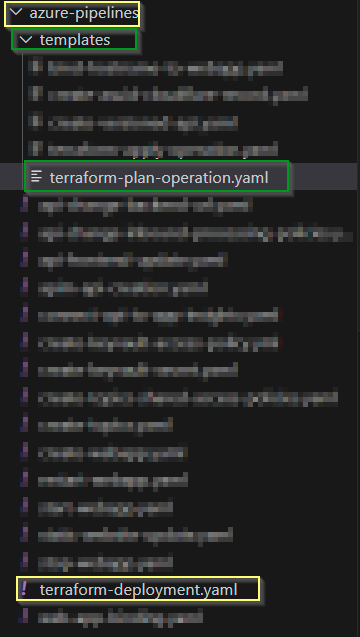
Why is that useful? Well, in our case, imagine if you need to change the Terraform workspace every time you get a new azure devops agent in place. Isn’t better if we could templatize that set of commands instead of repeating them over and over again?

# Implementing the Template in a Pipeline

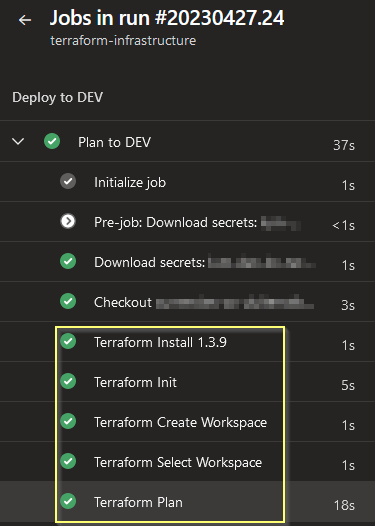
Now, let’s demonstrate how this template can be utilized within a pipeline.

# terraform-deployment.yaml  
  
stages:  
 - stage: "DEV"  
 displayName: "Deploy to DEV"  
  
 jobs:  
   
 - job: "PlanToDev"  
 displayName: "Plan to DEV"  
  
 pool:  
 vmImage: "ubuntu-latest"  
 variables:  
 - group: terraform-secrets  
  
 steps:  
 - template: templates/terraform-plan-operation.yaml  
 parameters:  
 connectionName: 'sc-dev-environment'  
 environmentName: "$(environment)"  
 # ... (Other parameters)  
  
 - template: templates/terraform-apply-operation.yaml  
 parameters:  
 connectionName: 'sc-dev-environment'  
 environmentName: "$(environment)"  
 # ... (Other parameters)  
  
- stage: "PRD"  
 displayName: "Deploy to PRD"  
  
 jobs:  
   
 - job: "PlanToPRD"  
 displayName: "Plan to PRD"  
  
 pool:  
 vmImage: "ubuntu-latest"  
 variables:  
 - group: terraform-secrets  
  
 steps:  
 - template: templates/terraform-plan-operation.yaml  
 parameters:  
 connectionName: 'sc-prd-environment'  
 environmentName: "$(environment)"  
 # ... (Other parameters)  
  
 - template: templates/terraform-apply-operation.yaml  
 parameters:  
 connectionName: 'sc-prd-environment'  
 environmentName: "$(environment)"  
 # ... (Other parameters)

In the steps, template, don’t forget to point to your own template file. This is how my files are organized:



If I run this now, all the steps that are declared in the template will be executed and you’ll be able to see it in the logs:



# Benefits and Takeaways

Utilizing Azure DevOps templates for Terraform pipelines offers several advantages:

1. **Consistency**: Templates ensure that the same configuration and steps are applied consistently across various pipelines and different environments, reducing the risk of errors due to manual configuration.
2. **Reusability**: Templates are reusable components that can be easily integrated into multiple pipelines, promoting efficient use of resources and reducing duplication of effort.
3. **Efficiency**: By encapsulating complex tasks in templates, the pipeline definition becomes cleaner and easier to manage, enabling faster onboarding and reducing the learning curve for new team members.

# Final Considerations

In conclusion, integrating Azure DevOps templates with Terraform is a potent strategy to optimize and efficiently manage the infrastructure. By adopting this approach, you can use the power of templating to achieve consistency, efficiency, and scalability in your DevOps practices in general.