Terraform Templates

We have provided a set of templates that provide a quick start for teams starting out with terraform.

The design philosophy behind the templates is that:

1. Each terraform configuration we have should have a single purpose, e.g. setup a base environment, deploy a function app, etc
2. A single terraform configuration should be able to be released through multiple environments (dev through production) without any changes - any differences are obtained from environment specific variables

> This separation of the code into a number of different repositories means there is no one place an entire environment can be stood up from scratch, however as there is no point in creating resources such as function and logic apps unless the code is deployed along with it.

> I recommend placing a watch on the template project so that you are alerted of any changes as these may be something you need to consider doing in your projects.
>
> For example, we are investigating how to build in monitoring and alerting and will be updating the modules to meet compliance policies. The templates will be kept up to date with best practice.

> When recreating an environment using **terraform destroy** and then **terraform apply** you MUST run terraform apply (usually 3 times) until it comes back and says that everything is up to date.
>
> This mostly applies when using virtual networks, but is a good habit to get into. It's most likely to do with the way key vaults are not permanently deleted immediately.

Terminology

Name	Meaning
module	A reusable bit of terraform code that can be used by your configuration
configuration	The terraform script that builds your resources
environment	All the resources required for an application to function, that might be D365, SharePoint, Azure Resources, External application. To manage multiple environments in with our terraform configuration we are using terraform workspaces. Our templates lump all the resources for an environment into a single resource group, therefore: "azure resource group" = "environment" = "terraform workspace".
base environment	The base environment is the resource group and common resources other resources will consume. For example: • A function app requires an app service plan, app insights, storage account and key vault • These additional resources are considered part of the base environment because they are only created once, while multiple function apps (and other resources) will use them

Azure Infrastructure Template

The Azure Infrastructure template is designed for projects using D365 and Azure Functions and is intended as a starting point only. You will need to customise the template to suit the needs of your project. Even if you aren't targeting D365 there is a lot of value in using the template as a starting point.

Refer to the code to get a better understanding of the template.

The template contains the following folders:

Folder	Description

global_variables	A module that is used by all terraform configurations in your project to return information specific to the Azure subscription your configuration will create the resources in. The purpose of this module is to minimise the amount of code that is required across all configurations.
shared_resources	Creates a "shared" resource group containing resources that can be used across all environments in your subscription, this includes a key vault containing the credentials of the the service principal used by pipelines. This key vault is configured such that the secrets are editable by developers. As this builds subscription wide resources you will run this one in your test subscription, and one in your production subscription.
base_environment	The terraform configuration that creates the common resources required that other components such as Function and Logic Apps require. Secrets in the key vault will not be editable (except for your development environment), so all secrets that your environment requires must be configured by terraform. As secrets must not be stored in code these must be looked up elsewhere. A handy place to store these is in the shared key vault where a convention is to store them using a secret name in the format "<secret name>-<environment-name>".

Function App Template

The Function App template should reside in the same source repository as the function app. Our standard is to create a folder called Terraform in the root directory of the solution. If multiple configurations are required for the solution then these are separated out into subfolders under the Terraform folder.

This template creates a function app and registers the managed identity of the function app in D365 as an application user so that the functions can connection to D365 using a managed identity connection. The example also demonstrates attaching to service bus queues and registering webhooks and queues in D365.

When using this template remove the code that is not relevant to your needs and search and replace any bits of code containing "[UPDATE]".

Logic App Template

The terraform configuration for a logic app should reside in the same source repository as the logic app. Our standard is to create a folder called Terraform in the root directory of the solution. If multiple configurations are required for the solution then these are separated out into subfolders under the Terraform folder.

The logic app module is a little different to other modules in that it deploys an ARM template (still using terraform) describing the logic app. This ARM template is typically built using the logic app designer in Visual Studio, but can also be exported from Azure.

The example has an example ARM template in the same folder as the terraform - normally this would reside in separate project within the solution.

Files

These files will appear in all terraform configurations built from the templates.

File	Purpose
main.tf	Contains the backend configuration, provider declaration, and the resources you want configured. While terraform itself doesn't require a file named main.tf as it will read any .tf file, the release pipeline requires that the backend is configured in this file so that it can pickup the backend key value.
variables.tf	Contains a block of code that loads variables from a file in the env folder with the same name as the workspace. This avoids the need to pass in environment specific variables in the command line and potentially updating an environment with the wrong values. These values are added to a local variable call env and can be referenced using local.env.<variable name>.

.terraform.lock.hcl	Contains the versions of the providers used when developing the configuration. When running "terraform init" the provider version will be read from this file and installed, this ensures other developers and the release pipeline all use the same providers versions. To update the version run 'terraform init --upgrade" To view the current versions run "terraform version"
env folder	This contains the environment specific variables in yaml format. The name of the file (minus the suffix) must match the name of the workspace - this is case sensitive. For example if you have a file called test.yaml and want to run your terraform against that: "terraform workspace list" - will list your workspaces "terraform workspace select <name>" - will select the workspace "terraform workspace new <name>" - will create a new workspace Once selected you can run "terraform apply"