QK Terraform Module Cheat Sheet

by Marco Ponton (mponton) via cheatography.com/173169/cs/36481/

Main Reasons to Create a Module

Enforce standards, settings, and conventions when deploying resources.
Offer a larger building block to facilitate resource deployment.
Implement logic to simplify users' life and improve Terraform adoption.

Naming Convention

BitBucket Project Key	We recommend you use a key that starts with `TF` and add an acronym to identify the main provider/platform (e.g. `TFAZ` = Azure, `TFGC` = GCP, `TFVR` = vRealize Automation, `TFVS` = vSphere, ...).
Repository	All repositories/modules **MUST** be named using the `terraform-<provider>-<name>` convention (e.g. `terraform-vsphere-windows-vm`, `terraform-azure-rm-keyvault`, ...).

Repository Structure

`/`	-	Root of the module
`/examples`	Required	Examples of how to use the module
`/images`	Optional	If your documentation includes images, put them in this folder
`/tests`	Optional	If you have Terratest files put there in this folder

Base Module Files

`.gitignore`	Required	Git ignore (see also `Security` section)
`.terraform-docs.yml`	Required	`terraform-docs` configuration
`.tfsec/config.yml`	Optional	If `tfsec` is enabled for the repository, it should have a configuration file here
`locals.tf`	Optional	If you use Terraform locals, you should put them in this file
`main.tf`	Required	Main module code
`outputs.tf`	Required	Definitions of module outputs should be put in this file
`provider.tf`	Required	Terraform providers configuration
`README.md`	Required	Main module documentation (see also QK's README.md Cheat Sheet)
`variables.tf`	Required	Module variables

If you prefer to organize your code in different `.tf` files, you can do so, but often a single `main.tf` is enough.

Coding Conventions & Style

Always add comments when useful. This is particularly important to explain reasons behind a specific design, choice of resources or values, edge cases, known bugs, issues or limitations in the code, etc. Do not add comments just for the sake of it (e.g. `# Create resource foobar`), add comments when the code is not enough.
Use underscores (_) not dashes when naming resources (e.g. `vm_image_name` not `vm-image-name`)
When naming variables, check existing modules and re-use existing names and patterns. For example, if module X uses `google_zone` and you also need to specify the zone, use the same name, not `gcp_zone`. Same for patterns, if the most common one is `google_*` use that pattern, not `gcp_*`.
Be consistent when naming resources and variables (e.g. don't name something `google_foo` only to name the next thing `gcp_bar`).
Remember that once your module is published, renaming resources and variables will become a breaking change and may impact existing projects negatively, so try to do it right the first time or chances are we'll need to live with it.
Validate all user inputs (variables) when possible. Follow the "fail early" mantra, do not let user create 13 resources only to fail on the 14th when you can prevent it.
When validating variable inputs, use multiple `validation` blocks when doing different checks. This allows you to provide more meaningful error messages.

QK Terraform Module Cheat Sheet

by Marco Ponton (mponton) via cheatography.com/173169/cs/36481/

Coding Conventions & Style (cont)

When it makes sense, provide useful defaults for variables.

Start the description of optional variables by "**OPTIONAL:** " followed by the description.

Do not use *explicit* dependencies (`depends_on`) unless necessary and you understand the impacts. Always use *implicit* dependencies instead.

Version Constraints

Your `provid er.tf` file **MUST** include version constraints.

Version constraints of each provider should usually be locked to the major version of the provider (e.g. `version = " >= 2.0, < 3.0"`) unless fully tested with multiple major versions".

If your code requires a specific minor of fix version, use the correct constraint and document why (e.g. `Version 1.2.0 or newer is required to prevent issue when updating resources of type XYZ`).

Semantic Versioning

Terraform modules **MUST** follow Semantic Versioning

Always pay attention to the changes you make to a module! Terraform can be very finicky and what may look like a simple change may trigger a destroy operation in existing project code.

Follow Semantic Versioning **rules**, do not fall for the "small changes" should not change the major version number mentality. For example, if you fix a typo in a variable name, any existing code using this variable will break, as such, small change = breaking change = new major version.

Project Upgrade Paths

When introducing breaking changes, always think about the best implementation for an upgrade path. We recommend you also create a `upgrad e-v X-t o-vY.md` (for example: `update -v2 -to -v3)md` file with instructions on how to upgrade existing projects to the new major module version from the last major version. This is particularly important if the breaking change may introduce data loss (trigger a Terraform destroy operation).

Documentation

Your module **MUST** include proper documentation. See QK's README.md Cheat Sheet for details.

Documentation **MUST** be part of the repository and **MUST** be written in Markdown. Feel free to split documentation into multiple files if needed, but always make the `README.md` file the main point of entry.

We also recommend you review some of QK's GCP VM modules (e.g. Windows VM for some sample README files.

terraform-docs

Always ensure you have proper `.terra - for m-d ocs.yml` in your repository and run `terraf orm -docs` after making changes to your module to update documentation

Look into one of QK's latest Terraform module for a recent copy of the `.terra - for m-d ocs.yml` file.

Security

When one or more providers in your module are supported by the `tfsec` tool, you should create a base `.tfsec /co nfi - g.yml` and run the tool against your module to validate it. If any issues are found, fix them or if not possible, add the proper exceptions to the `config.yml` file and document them.

⚠ Always exclude `.tfvars`, state files, and any secrets using `.gitignore` before you commit code.

Releases

Always tag official releases with a proper `vX.Y.Z` semantic version tag.

Always ensure your README and any other documentation is up-to-date before publishing a new release.

Always provide 2-3 up-to-date working examples with your modules. Exception of any authentication and access, these should work out-of-the-box for any user that wants to deploy them.

Tips

If your module relies on other modules, always lock their version using either version constraints (if using registry) or `git` URL `ref` (e.g. `?ref=v 1.2.3`)

When creating examples for your module, provide at least one `minimal` and one `complete` example.

Include all necessary files in your examples so they can be used as a starting point for a new project. For example, include a `.gitignore` even if not needed for the example to be functional.