Azure or Nothing Part 1

Pipelines VBScript-style

Onkobu Tanaake

2018-09-20T10:00:00

Azure started years ago with split build- and release-pipelines. Both were created as point-and-click adventure. Abstract terms like agents, stages and tasks are mixed and brought into an order. It is easier for someone starting with Azure to repeatedly point and click to achieve the same sequence of actions than copying: there is no copy-feature. Surely you can clone a task but only within an agent's scope. Mixing Windows- and Linux-environments to combine tooling like Powershell and Kubernetes was impossible without repeating yourself.

Later so called task groups were added. They're still point-and-click but are re-usable building blocks with parameters. Fetch a Docker image, have some security checks, hand over to Helm, check some post conditions and for 3 different environments: use the same task group 3 times.

After this worked out YAML pipelines were introduced and Build Pipelines were re-labelled just Pipelines. Release Pipelines still remain as the one and only classic pipelines. A migration from already created classic pipelines to YAML variants is impossible. Surely a button exists to show the fancy UI's representation of a task as YAML. But this rendering is incomplete and sometimes wrong. Some script tasks use the property *arguments* others use *scriptArguments* and why not add a third? True YAML- pipelines have the same property but spelled different, e.g. *scriptarguments* or simply switched.

In addition you can't migrate your task groups to the new YAML templates. YAML templates come in 3 flavors: stage-, job- and task-templates. How should a task group be converted? Also parameters of task groups are simply ignored when rendering their YAML. At least you get the set of tasks nearly right. Again some properties are named different in YAML-pipelines or have differences in case.

But my topmost inconvenience is the deviation from the YAML standard. You can't have anchors and references. Instead you get a threefold pattern replacing with subtle errors lurking in a muddy pit to rip off limbs. First of all there is a regulare variable expression like `property: $( variableName )`. This is fine in a pipeline but not between pipelines and (imported) templates. To transfer something from a pipeline to a re-usable building block `property: ${{ parameters.variableName }}` is necessary. And in addition Bash-scripts can access variables and paramters, too, through the environment. But you need to spell it differently: all uppercase and '.' is replaced with '_' – convinient?

Second you can also have sort of eval-pendant with square brackets. You can also access variables programmatically, so that a variable's content is the variable to lookup. Third you have to use a strange de-referencing through either

`${{ parameters.name}}` or `${{ variables.name }}` if you fell into some precedence corner case because of name clashes. (A pipeline imports templates that set variables independent of each other to transfer outcomes to later stages.)

And now the most fucked up issue I ever made up with[1]: you can't have certain properties of YAML pipelines stuffed with variables nor parameters. For example azureSubscription in a task that is part of a template can neither be filled from a variable nor a parameter. There is a delicate detail about pipelines: two-pass-compilation. The first pass evaluates resource- and security checks and cannot evaluate variables reliably. But the azureSubscription-property is THE permission carrier. Thus trickery with a template that has a task wich requires an azureSubscription cannot refer to ${{ Environment.Name }}. You get a simple error upon pipeline start: *Azure subscription ${{ Environment.Name }}…cannot be found*. For me it seems like an easy task to also peek into the variable store and resolve the system variable `Environment.Name`. But I assume it reveals other more delicate issues. Other quirks and issues:

- A job template if imported cannot have additional tasks before or after
- Deployment jobs cannot refer to output variables
- And jobs can only access output variables across stages with complicated de-referencing of associative arrays/ hash maps
- Pipelines can relate to other pipelines but can't exchange data like build number or branch that triggered the build
- Other versioning concepts like Maven's <version>-tag are hard to integrate (require a script that transfers the value as output variable)