ARM: Azure resource manager templates, nested loops – a commentary

What is Azure Resource Manager, ARM?

Arm is a Microsoft Azure provided managed service that enables an automation designer to define their design intent, expressed as templates, using arm's automation language, currently azureRM transitioning to Az.

Arm enables a designer to express their intent as to the selection, configuration, and assembly sequence of computing resource objects as specified by arm automation resource templates.

For example an automation designer's intent to design the resource objects and their necessary sequencing in order to install 0toN virtual machine extension resource objects on one to n existing virtual machine resource objects.

When is this automation used?

This automation is used when an automation designer needs to provision multiple provider supplied, virtual machine extensions on multiple existing, running, azure virtual machines. For example to provision Microsoft supplied antiMalware and/or bgInfo virtual machine extensions on existing virtual machines.

An automation designer uses azure resource manager's (arm's) declarative template language to implement an automation's design as the specification of each automation resource, and the order in which automation resources are deployed or provisioned.

When initiated, each resource specification is rendered, by arm, as a resource object, in an azure subscription namespace. The azure subscription namespace is secured by azure active directory and observable through the azure portal.

Design: Activity composition

An automation designer expresses an automation's design as the composition and sequencing of activities that add and/or change the arrangement of resource objects in the automation's namespace, in an azure subscription. Activities can be composed and are implemented using azure tasks. Arm tasks are not composable, arm templates are.

The activities in this automation install 0 to n virtual machine extensions, on 1 to n virtual machines. The automation is parameterized by two sets, a virtual machine names set, and a virtual machine extension names set. Both parameters are structured as arrays of strings.

Design: The automation's anatomy

This automation has three levels of activities, the first two levels have iteration controls and are composed with the unique, provider supplied virtual machine extension resource specifications, at the third level.

The activity compositions are:

- Level 1, the 'user' initiated activity, iterates over the virtual machine names in the array vmNames. A resource condition statement stops composition with level 2 when array providerVmExtensionNames is empty.
    - See template install-0toN-pVmExts-on-1toN-vms.json

- Level 2 activity, is initiated from level 1, and iterates over the designer nominated virtual machine extension names in the providerVmExtensionNames array. See template install-0toN-pVmExts.json
- Level 3, is initiated from level 2 and installs each virtual machine extension specification resource on a nominated virtual machine.
    - See provision-pVmExt-antiMalware.json or provision-pVmExt-bgInfo.json

Each level 3 activity is a provider supplied, virtual machine extension resource specification used to provision an antimalware, or bgInfo vm extension on to the already installed virtual machine resource objects tpt-ar-20. tpt-ar-21.

Design: Activity composition and nested iteration schematic

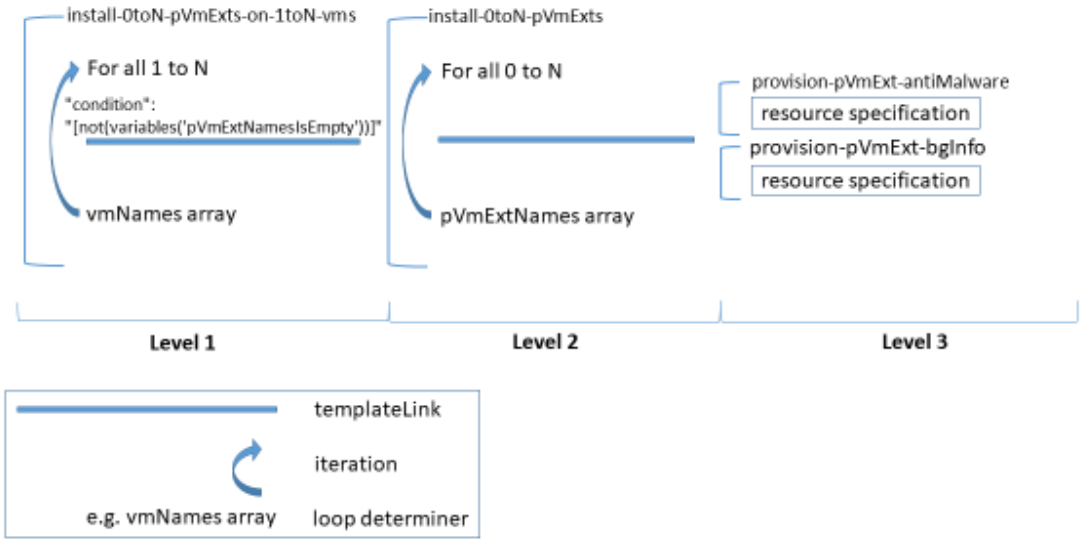

Fig 1, Shows activity composition, over two levels implemented by nested template links, activity iteration, and the empty array condition at level 1, that stops composition with level 2.

Implementation: Composition control, the challenge

Arm's declarative, template language enables an automation designer to implement template structures comprised of individual arm templates, corresponding to each design activity, to control the sequence (i.e. order), the number and, the location - in a subscription's namespace - of the resource objects comprising an automation.

For example the designed activity install-0toN-pVmExts-on-1toN-vms included in this automation, uses a three level nested template structure with four templates, two at level 3. The template structure corresponds to the designer's arrangement of the automation's activities.

The happy path through the three level nested and iterated template structure emerges when the virtual machine names array and the virtual machine extension names array both have members.

The level 1 template fails when the virtual machine names array, vmNames, is empty. The arm engine, throws a template validation error:

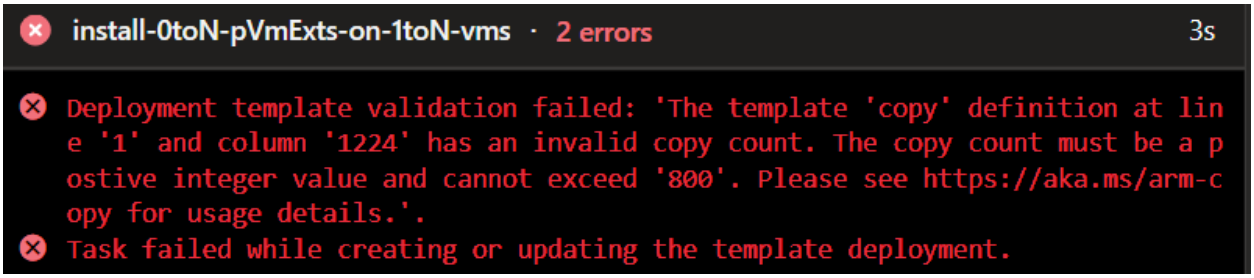

Fig 2, Shows an invalid template error message, caused by an empty array evaluated by a copy element's count property.

Inserting a resource condition element does not prevent the error.

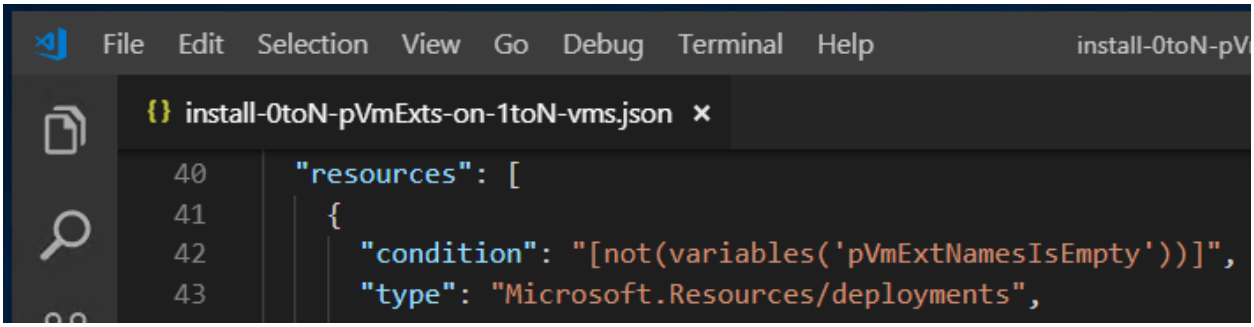

Fig. 3, Shows a resource condition element in template for level 1 composition activity.

Implementation: Composition control, the diagnosis

It appears that in templates nested with a resource template link element, the resource condition element (above), is applied by arm's template validation engine, after the engine has evaluated a resource copy element's count property.

As a consequence the copy element's count property is evaluated before the resource condition element is applied, and arm returns the above error. For example, as in template install-0toN-pVmExts.

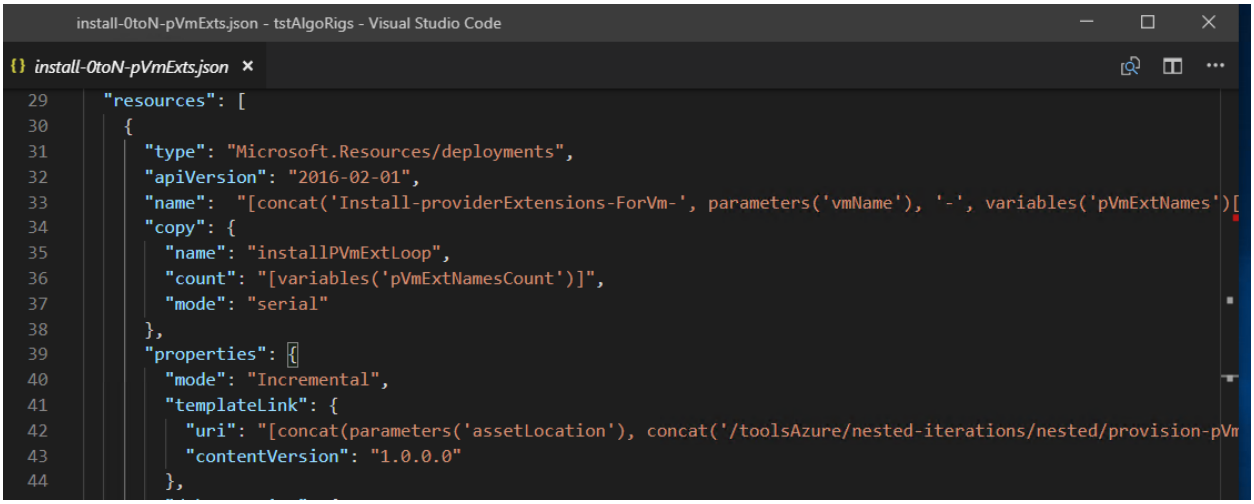

Fig 4, The resource copy element, count property in template install-0toN-pVmExts is evaluated before the condition element in template install-0toN-pVmExts-on-1toN-vms.json is applied. when array providerVmExtensionNames is empty. Note a term in the URI concat() function is the container sas token.

That situation is avoided in the install-0toN-pVmExts-on-1toN-vms template because that template requires the array vmNames to have at least one member, and so can not be empty, otherwise what's the point.

However the virtual machine extension names array, i.e. providerVmExtensionNames, can be empty, in which case arm throws a similar error, in the install-0toN-pVmExts template.

In summary, using a resource condition element in a deployment template's to avoid the evaluation of that resource's, copy element count property, didn't work for me.

Implementation: Composition control, the workaround

The preferred objective being to pass activity install-0toN-pVmExts-on-1toN-vms either an empty providerVmExtensionNames array or an array with members and have the activity composition complete successfully.

This automation design works around the above situation – i.e. the designer's need to stop resource iteration and activity composition when an empty argument array is detected -- by spoofing arm's template validation.

The install-0toN-pVmExts template implements two variables, pVmExtNames and pVmExtNamesCount, each variable is conditioned by a template expression if() function that sets the template variable's value differently depending on the presence or absence of virtual machine extension array members.

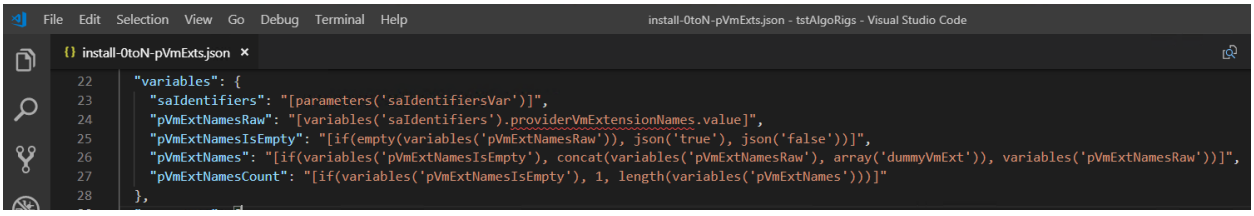

Fig 5. Variables section of template install-0toN-pVmExts.json showing variables used to spoof template validation.

Note: template variable pVmExtNamesCount is assigned int 1 when the array is empty.

- "pVmExtNamesIsEmpty": "[if(empty(variables('pVmExtNamesRaw')), json('true'), json('false'))]",
- "pVmExtNames": "[if(variables('pVmExtNamesIsEmpty'), concat(variables('pVmExtNamesRaw'), array('dummyVmExt')), variables('pVmExtNamesRaw'))]",
- "pVmExtNamesCount": "[if(variables('pVmExtNamesIsEmpty'), 1, length(variables('pVmExtNames')))]"

Note, the concat() function in the template variable vmExtNames assignment returns an array, and is passed two arguments, each structured as a strings array.

The template variable values assigned in template install-0toN-pVmExts, when array pVmExtNames is empty, are irrelevant as the template install-0toN-pVmExts, is never initiated.

The resource condition element, in the template install-0toN-pVmExts-on-1toN-vms template, stops the activity composition sequence at level 1, and the level 2 template install-0toN-pVmExts is never initiated when the pVmExtNames array is empty.

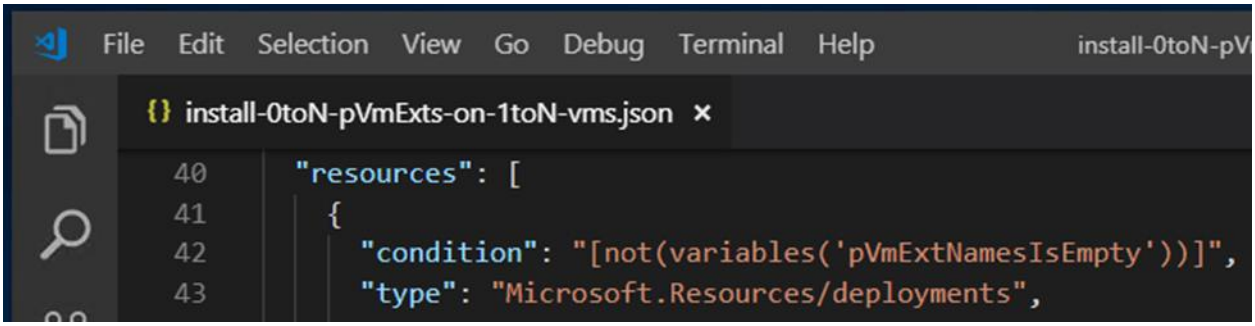

Fig 6, Shows the level 1 template's resource condition element showing the condition statement that stops composition before level 2 is initiated.

Verify: Automation input parameters

In addition to the usual asset location and shared access signature, required for nested templates by arm, the arrays controlling composition are shown here:

Fig 7. Shows template parameter arguments for template install-0toN-vmExts-on-1toN-vms.

Verify: Automation template configuration

Note: Templates for the four activities -- install-0toN-pVmExts-on-1toN-vms, install-0toN-pVmExts, provision-pVmExt-bgInfo, provision-pVmExt-antiMalware -- described above are available at: https://github.com/TecProTools/azure-arm-nestedLoops

For provision testing without existing virtual machines, set the value of template variable spike to true, in each provider supplied virtual machine extension template e.g. provision-pVmExt-antiMalware. Doing so prevents the virtual machine extension resource being provisioned and so stops the arm engine looking for the previously installed virtual machine resource objects, set in parameter vmNames, of template install-0toN-pVmExts-on-1toN-vms.

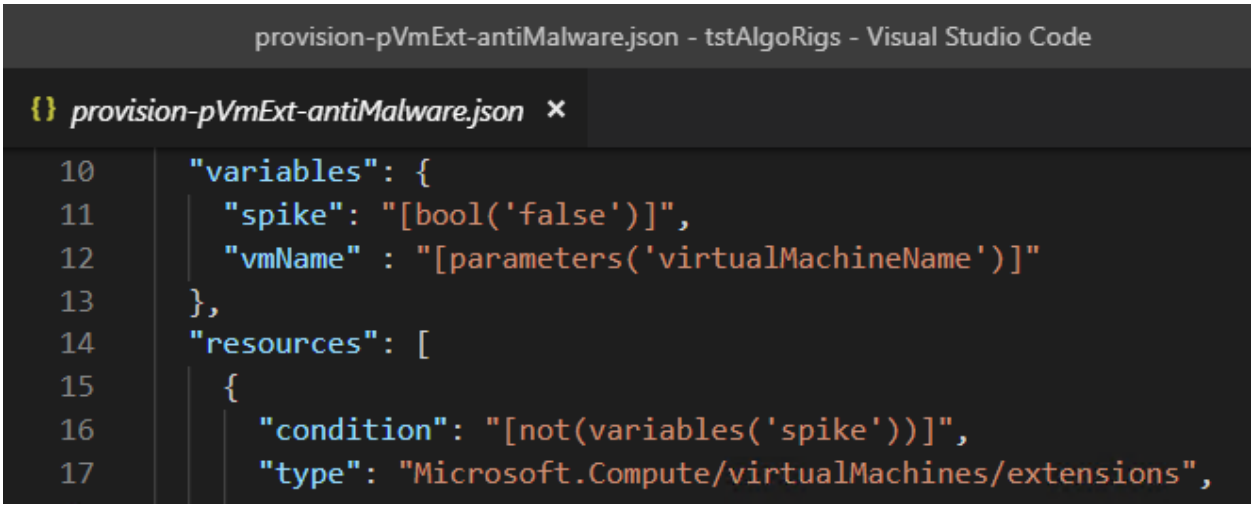

Fig 8. Shows template variable spike, set to false, and corresponding resource condition.

To demonstrate the interaction between a resource condition and a resource copy element at different activity composition levels, set the argument of parameter providerVmExtensionNames to empty.

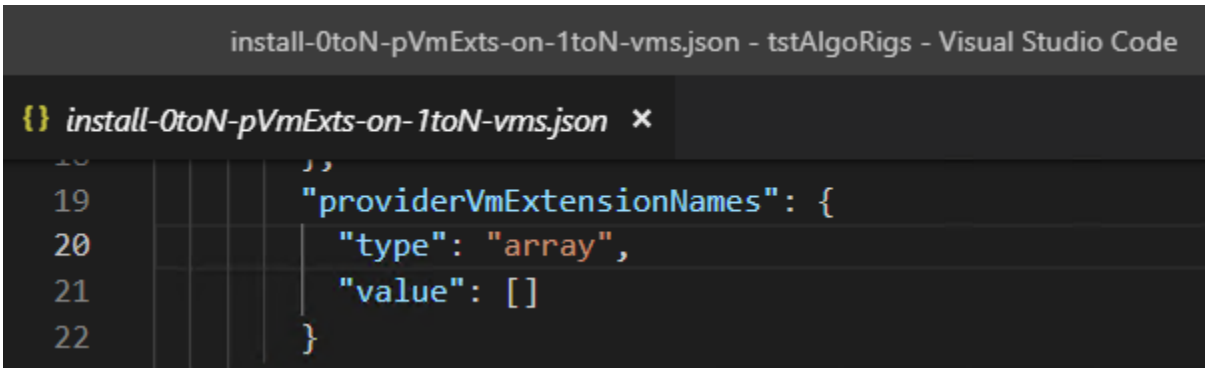

Fig 9, Shows the argument of parameter providerVmExtensionNames set to empty.

The release pipeline action install-pVmExts-on-vms, can be initiated from either powershell or an azure release pipelines blade via the azure devOps portal -- was vsts portal.

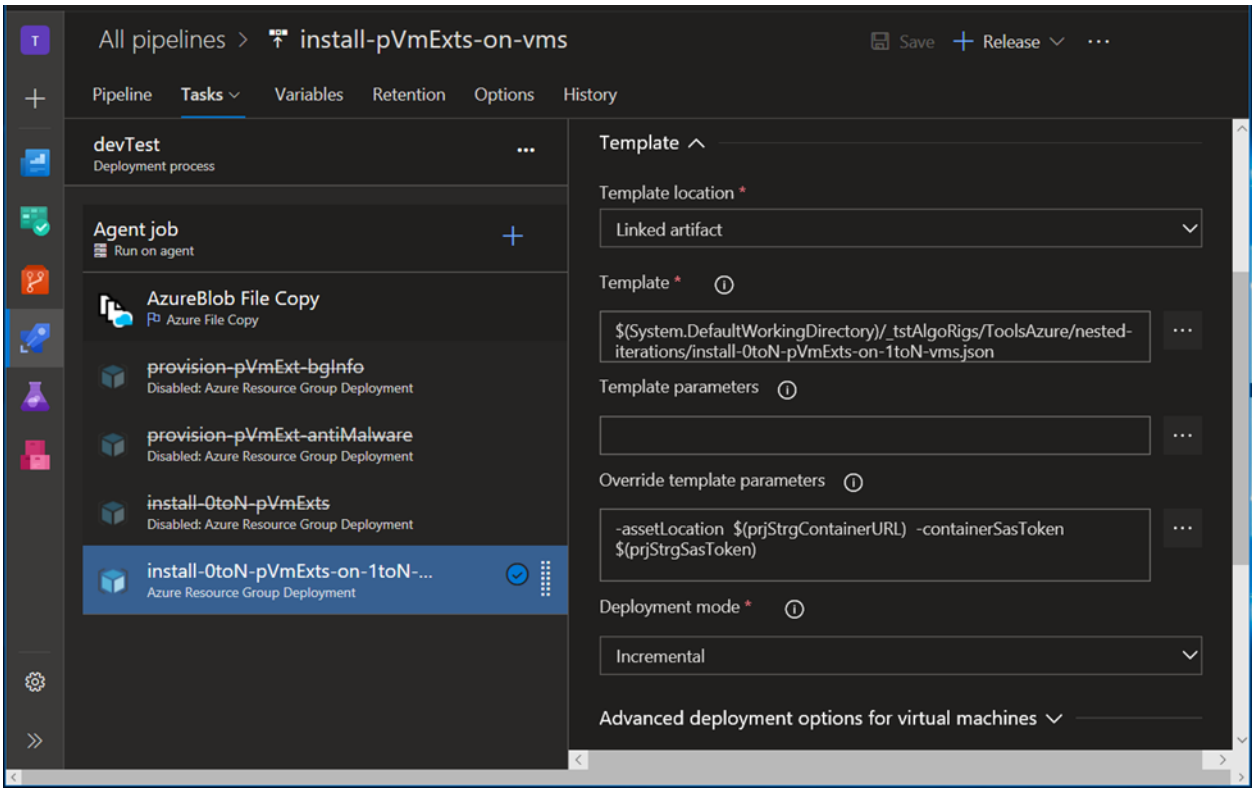

Fig 10. Shows the azure devops (azdo) release pipeline action, install-pVmExts-on-vms, comprised of five tasks, three disabled. Tasks AzureBlob File Copy and install-0toN-pVmExts-on-1toN-vms, provision multiple pVmExts resources on 1toN not necessarily existing virtual machines.

The AzCopy activity requires an existing azure storage account, and a shared access signature secured blob collection container.

On successful completion of the arm release pipeline action, install-pVmExts-on-vms,

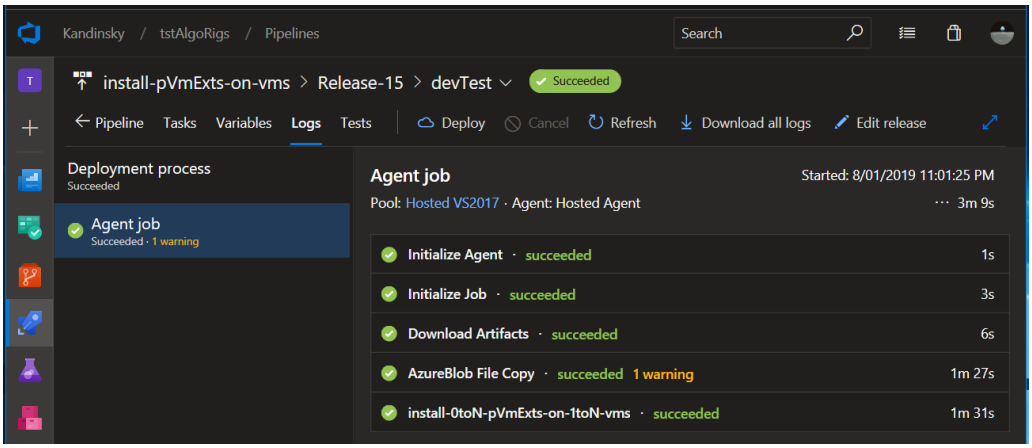

Fig 11, Shows the successful completion of the azdo release pipeline action, install-pVmExts-on-vms.

Verify: Automation state changes observable through azure portal

For provisioning, the virtual machine resource objects identified by the vmNames argument's values must be created prior to initiation of activity install-0toN-pVmExts-on-1toN-vms.

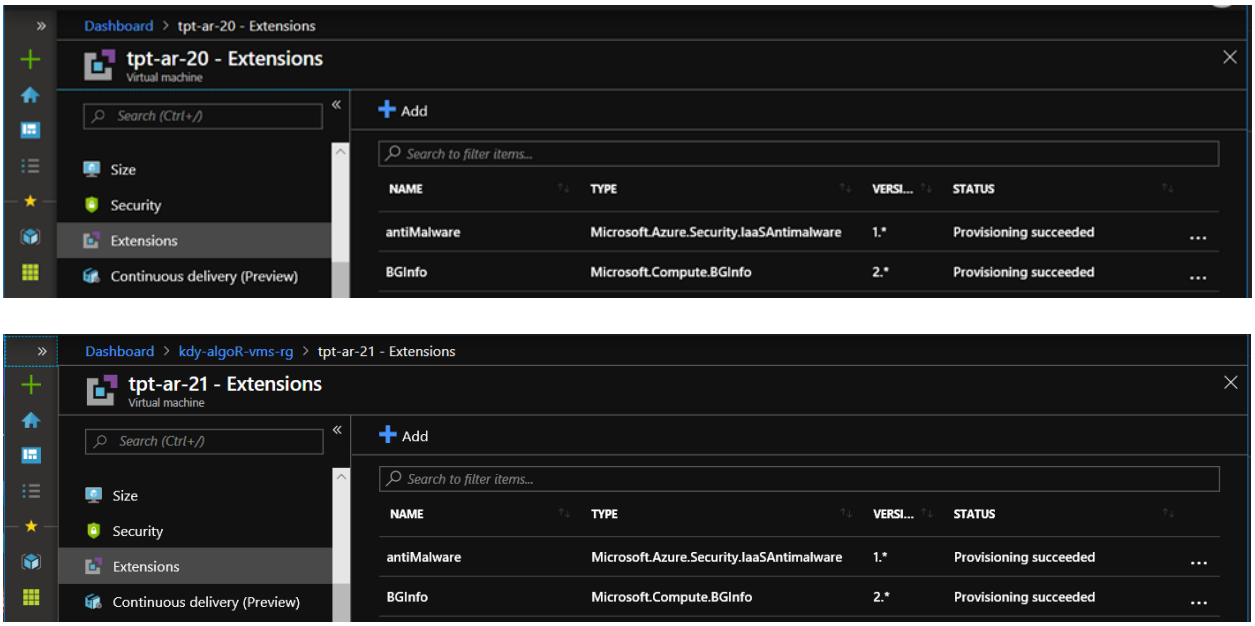

Fig 12. Shows the nominated, existing, virtual machine resource objects with successfully provisioned, Microsoft supplied, virtual machine extension resource objects.