Azure arm virtual machines, nested loops, winRM and custom extensions

What is Azure Resource Manager, ARM?

Arm is Microsoft Azure's managed automation hosting service. Arm enables an automation designer to define their design intent, expressed as resource specifications using arm's declarative, template automation language.

Arm requires designers to express their design intent:

a) As the selection, implementation, deployment and provision of computing resource objects, together with
b) The assembly sequence of those computing resource objects, that is the assembly of compute, storage, and network resource objects.

Arm provides a mapping from compute, network or storage resource specifications to resource objects rendered by arm onto a resource object canvas, located in the automation owner's azure subscription namespace, visible through azure's secured resource management portal.

When is this automation used?

This automation is used when an automation designer intends to implement, install, and provision one or more virtual machine assemblies.

This design extends the automation described in the previous article https://dzone.com/articles/arm-azure-resource-manager-templates-nested-loops to include the installation of 1 to n virtual machine assemblies, and to provision winRM, initialize data disks, and configure an application log on each virtual machine. A virtual machine is assembled from a collection of resource objects: compute e.g. vm; storage e.g. data managed disks, and; network e.g. network interface card and public ip address.

Design: virtual machine assemblies requirements

The automation's design requirements are:

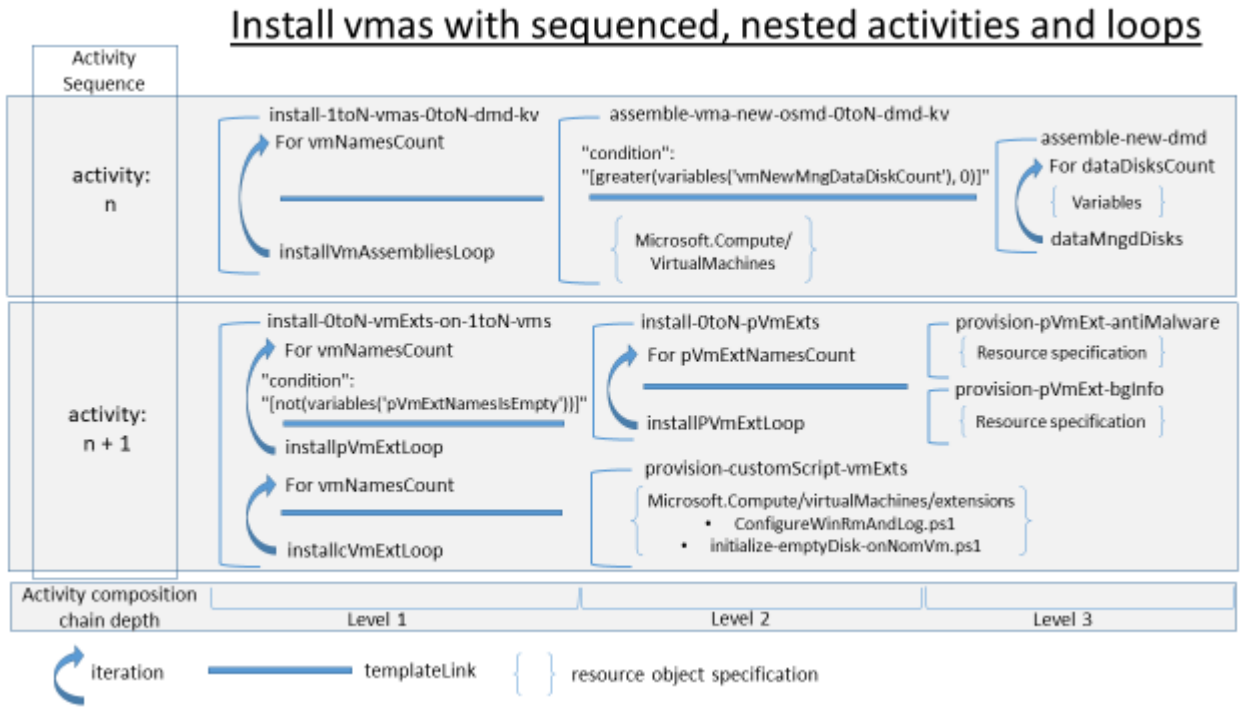
- Install two virtual machine assemblies.
- Attach two managed disks for data to each virtual machine.
- Retrieve virtual machine administrator password from a secure location, i.e. key vault.
- Use existing public ip (pip) address resource objects.
- Use existing network interface card (nic) resource objects.
- Attach nic to existing subnet and vnet.
- Provision each virtual machine with winRM.
- Provision each virtual machine with a windows event viewer application log.
- Initialize each virtual machine's data managed disks.
- Use a secure private storage location to store nested templates and custom script virtual machine extensions, i.e. not a public repo.

Design: activity composition

This design uses sequenced activities and a similar collection of loops and nested activities to that used by the automation described in a previous article. For details of activity composition and the nested control mechanism see that article.

This design uses two sequenced activities -- activity n and n+1, each having an activity composition chain three levels deep -- to install a virtual machine assembly, provision winRM, initialize data managed disks, and add the application log.

[fig1]



Design: activity n, install vm assemblies

Activity n, install-1toN-vmas-0toN-dmd-kv, installs the virtual machine assemblies. The number of virtual machine assemblies installed is determined using the copy loop count vmNamesCount, at level 1 in the activity composition chain.

The level 2 and 3 activities, assemble-vma-new-osmd-0toN-dmd-kv and assemble-new-dmd, are repeated for each virtual machine assembly using the copy statement in level 1.

The level 2 activity has two resource specifications. The first, a link to the level 3 activity, assemble-new-dmd, which is the resource specification for the two managed disks used for virtual machine for data storage.

The second resource specification assembles and installs a virtual machine resource object using the resource objects e.g. the nic, pip, disks, vm, named in the virtual machine resource specification.

The admin password for each virtual machine – this design uses the same password -- is stored in the key vault and retrieved by arm when the virtual machine resource object is rendered. For implementation details see the below, for activity n.

Design: activity n+1, install vm extensions

Activity n + 1, install-0toN-vmExts-on-1toN-vms, provisions proprietary e.g. bgInfo and custom virtual machine extensions e.g. ConfigureWinRmAndLog on each installed virtual machine.

Arm uses the Microsoft.Compute/virtualMachines/extensions resource object, and naming conventions, to provision both proprietary and custom virtual machine extensions.

This design uses separate resources to provision proprietary and custom script extensions. The activity install-0toN-pVmExts provisions the proprietary bgInfo and antiMalware extensions. And the activity provision-customScript-vmExts provisions the custom script ConfigureWinRmAndLog.ps1 extension.

For the installation of proprietary custom scripts see previous article.

Arm enables only one custom script virtual machine extension to be installed on a virtual machine. When virtual machine provisioning requires multiple scripts either the scripts are chained, as is done below to initialize empty disks, or the first custom script extension must be removed from the virtual machine and a new extension resource object installed.

The custom script ConfigureWinRmAndLog.ps1 installs the application log, configures winRM, and then chains to the script, initialize-emptyDisk-onNomVm.ps1, to initialize the data managed disks on the nominated virtual machine.

Design: resource groups and secure private storage

Arm requires designers to group resource objects in resource group objects. Arm's guidance to designers is to group resource objects with similar lifecycles in the same resource group object, see: https://docs.microsoft.com/en-us/azure/azure-resource-manager/manage-resource-groups-portal#what-is-a-resource-group

This design has two resource groups:

1. tpt-algoR-ops-rg, to group resource objects with long stable lifecycles, for example the automation's, key vault tpt-algoRigs-kv, and operations storage account tptalgorigsops10strgacc. The blob collection container, prjtplatesandscripts, used as the private store for automation templates and scripts, is located in the operations storage account.
2. tpt-algoR-vms-rg, to group components for virtual machine assemblies. The component resource objects for each virtual machine assembly are a network interface card, a public ip address, two data managed disks, a virtual machine os managed disk.

For the structure of the operations storage resource group see the implementation description below.

The resource specifications for nested activity levels 2 and 3 are implemented using azure's external template syntax described here: https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-linked-templates#external-template-and-external-parameters

The templates are located in the private storage account tptalgorigsops10strgacc, blobs collection prjtplatesandscripts. A private storage account is secured by a shared access signature, also known as a sas token.

When arm renders virtual machine assembly resource objects, a sas token is used to access the task templates at the secure private storage location. For details see, implementation activity n, below.

Implementation: activity n, install vm assemblies

Activity n, implemented by task template install-1toN-vmas-0toN-dmd-kv.json, installs each virtual machine assembly.

The admin password for each virtual machine is retrieved by arm, from the key vault resource object tpt-algoRigs-kv, and passed from the level 1 template install-1toN-vmas-0toN-dmd-kv.json as a secure string argument into the level 2 template, assemble-vma-new-osmd-0toN-dmd-kv.json.

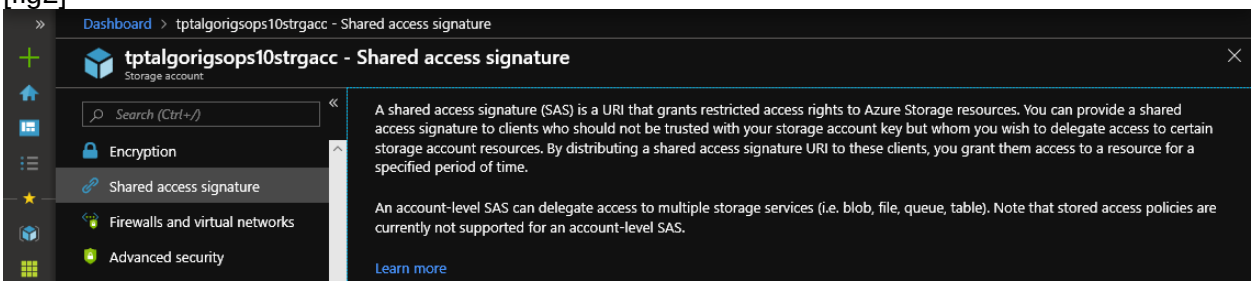
A task template implementing an activity runs ‘in’ a resource group object, as a consequence a reference to a resource object e.g. a key vault, located in a different resource group e.g. tpt-algoR-ops-rg is qualified by its resource group name.

The key vault is located in the operations resource group, not the virtual machine assemblies’ resource group, so the key vault, referenced using the resource object resource id e.g. kvId, includes the key vault’s resource group:

```
"kvId": "[resourceId(variables('kvRgName'), 'Microsoft.KeyVault/vaults', variables('kvName'))]"
```

The sas token that secures the private storage containing the nested templates and scripts is passed to task template install-1toN-vmas-0toN-dmd-kv.json, as a secure string argument and subsequently passed to the level two template assemble-vma-new-osmd-0toN-dmd-kv.json.

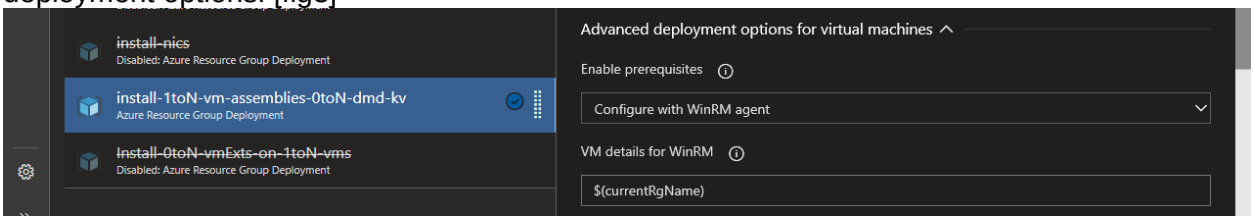
A sas token is retrieved from the azure portal here:
[fig2]



A virtual machine can be provisioned with winRM using either:

- The azure devOps release pipeline available through the azure devOps portal, or
- A custom script virtual machine extension as done here, see activity n+1 implementation, below.

To provision winRM using the azure devOps portal release pipeline, set the arm advanced deployment options. [fig3]



Note the “VM details for WinRM” is a release pipeline variable name holding the value of the resource group containing the virtual machines.

Implementation: activity n+1, install vm extensions

For a description about provisioning proprietary custom script virtual extensions e.g. bgInfo, see the earlier article.

To provision winRM using a custom script virtual machine extension specification, this design uses the level 2 template provision-customScript-vmExts.json, initiated from level 1.

Existing installed virtual machines are provisioned by the level 1 activity install-0toN-vmExts-on-1toN-vms implemented by task template install-0toN-vmExts-on-1toN-vms.json.

The level 1 template install-0toN-vmExts-on-1toN-vms.json receives the vmCustomScriptVmExtNames string array argument locating and naming powershell scripts used by the custom script virtual machine extension: [code1]

```
            "vmCustomScriptVmExtNames": [
"https://tptalgorigsops10strgacc.blob.core.windows.net/prjtplatesandscripts/compu
te/vmExtensions/vmExtScripts/winRM/ConfigureWinRM.ps1",
"https://tptalgorigsops10strgacc.blob.core.windows.net/prjtplatesandscripts/compu
te/vmExtensions/vmExtScripts/winRM/initialize-emptyDisk-onNomVm.ps1"
            ]
```

The vmCustomScriptVmExtNames strings array is passed to the level 2 template provision-customScript-vmExts.json which implements the custom script virtual machine extension resource specification, Microsoft.compute/virtualMachines/extensions.

The necessary level 2 extension resource specification properties are:

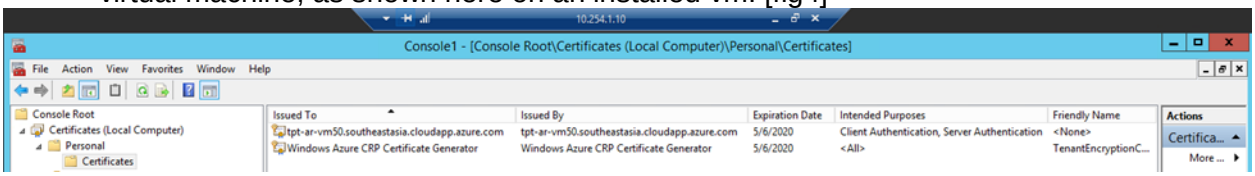
1. The template variable strgAccId value, is a resource Id, qualified by a resource group for the reason described above: [code2]

```
    "strgAccId": "[resourceId(parameters('opsRgName'),
'Microsoft.Storage/storageAccounts', variables('strgAccName'))]",
```

2. The template variable hostDNSNameScriptArgument value, [code3]

```
    "hostDNSNameScriptArgument": "[concat (variables('vmName'),
'.southeastasia.cloudapp.azure.com')]"
```

   is used as the subject of the self signed certificate that enables winRM access to each virtual machine, as shown here on an installed vm: [fig4]



3. "fileUris": "[variables('vmCustomScriptVmExtNames')]"
   The file uris is the strings array template variable sourced from template parameter vmCustomScriptVmExtNames as described above.
4. "commandToExecute"
   The commandToExecute property specifies the powershell script to provision the virtual machine. The path used for powershell -file argument matches the location of the ConfigureWinRmAndLog.ps1 script in the automation's private storage i.e. tptalgorigsops10strgacc/prjtplatesandscripts/compute/vmExtensions/.. . The script is copied to a similar location on the virtual machine. [code4]

```
      "commandToExecute": "[concat('powershell -ExecutionPolicy Unrestricted    -
file ./compute/vmExtensions/vmExtScripts/winRM/ConfigureWinRM.ps1
',variables('hostDNSNameScriptArgument'))]",
```

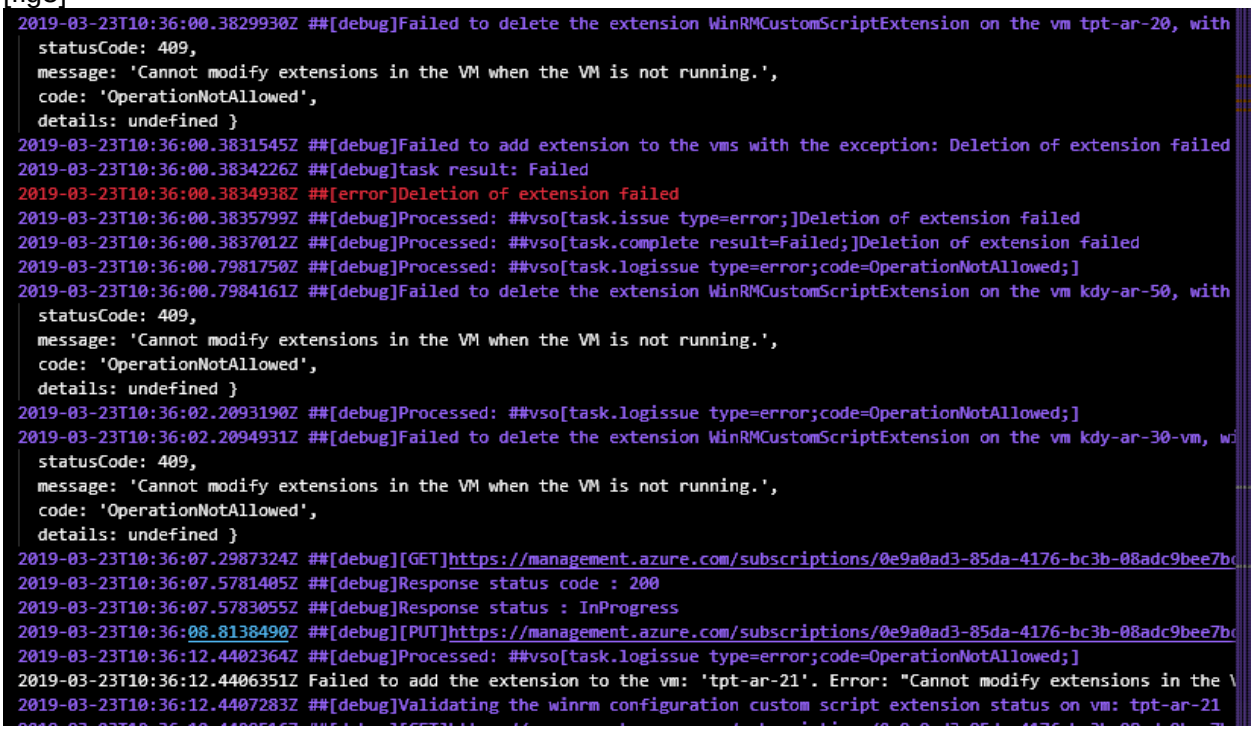
5. "storageAccountKey":

The virtual machine extension custom scripts are located in private storage .i.e. at tptalgorigsops10strgacc/prjtplatesandscripts. Arm uses a storage account key, not a sas token, to access the custom scripts used by the virtual machine extension object. [code5]

```
        "storageAccountKey": "[listKeys(variables('strgAccId'),
providers('Microsoft.Storage','storageAccounts').apiVersions[0]).keys[0].value]"
```

The azure template function listKeys returns the first storage account key used by the property, storageAccountKey. Note, this implementation assumes the first apiVersion.

All virtual machines in the nominated resource group need to be running when applying a vm extension, even those not targeted for change. The change applies to all virtual machines in the resource group. Otherwise this message appears in the log, 'Cannot modify extensions in the VM when the VM is not running.'.
[fig5]



Implementation: custom script vm extension, pathnames

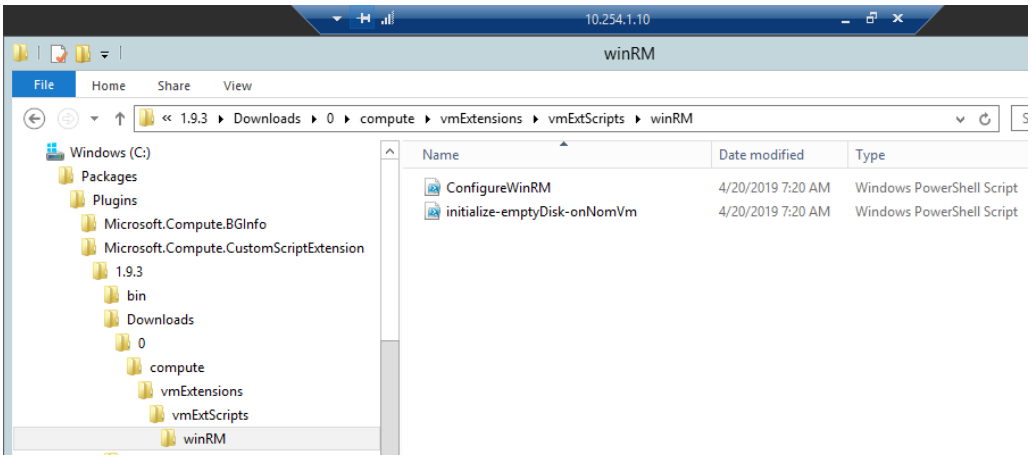
The file path used by a powershell 'dot source' statement in a custom script virtual machine extension to locate the script to chain to e.g.

./compute/vmExtensions/vmExtScripts/winRM/initialize-emptyDisk-onNomVm.ps1

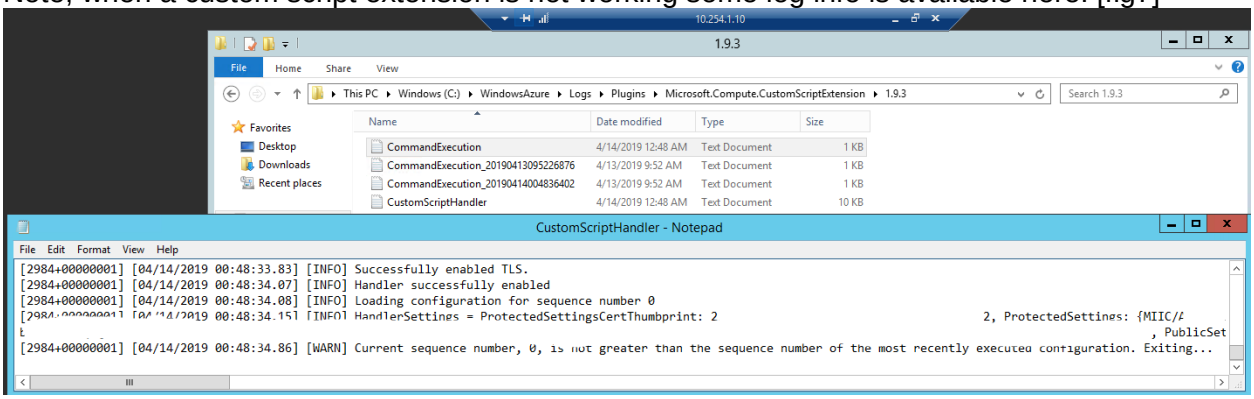
must match the location on the virtual machine to which the custom extension resource copies the fileUris used to provision the virtual machine, as used here in script ConfigureWinRmAndLog.ps1. [code6]

```
    ## Initialize empty disks
   Write-EventLog -LogName  $logNameToInstall -Source $logNameSrc -EventId 1000 `
    -Message "Verify, next message confirms extension, initialize-emptyDisk-
onNomVm.ps1, ran."
    . ./compute/vmExtensions/vmExtScripts/winRM/initialize-emptyDisk-onNomVm.ps1
```

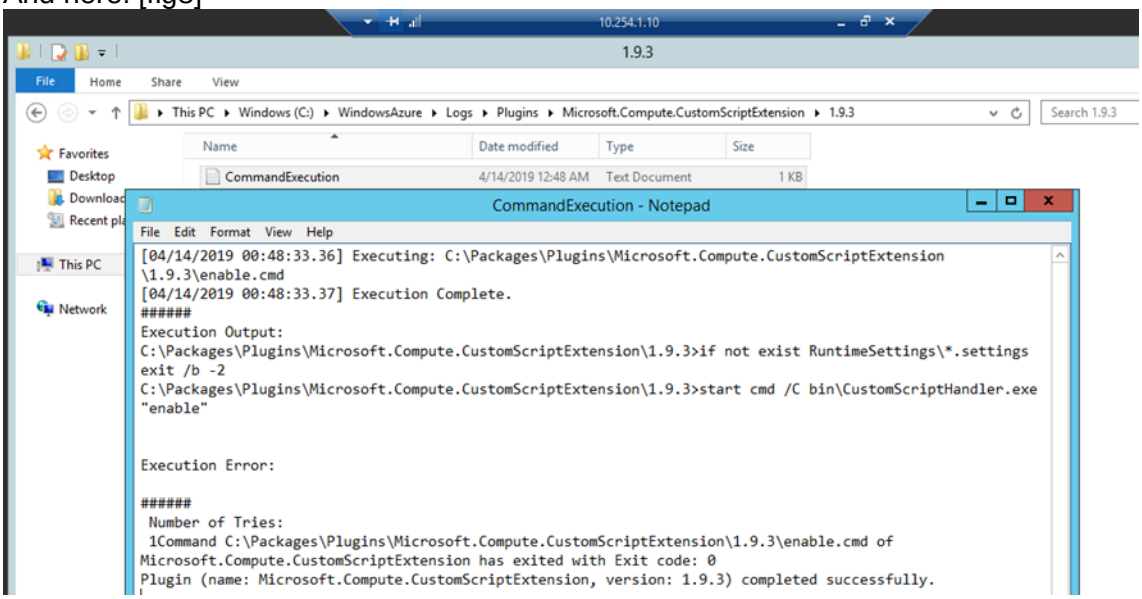
Which is copied to here, [fig6]



Note, when a custom script extension is not working some log info is available here: [fig7]



And here: [fig8]



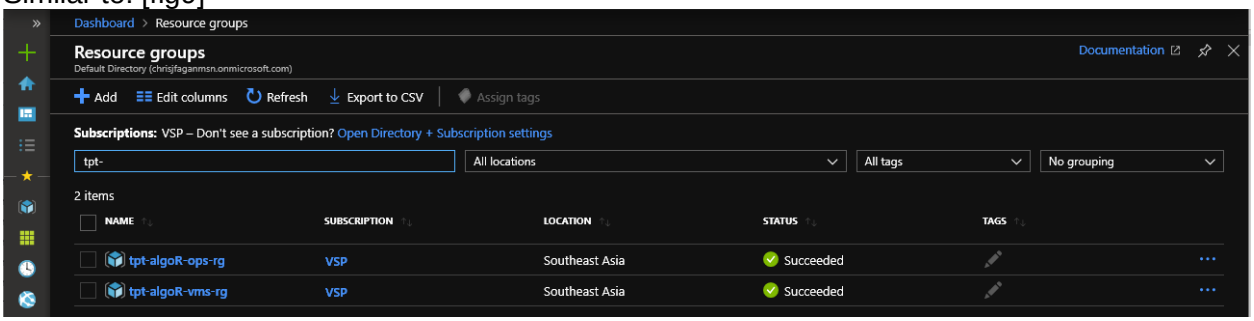
Implementation: storage and pre-installed resource objects

Some resource objects must be installed prior to those resource objects which are either a) assembled from them, or b) consume their services. For example, a virtual machine's admin user password, securely stored as a key vault secret in a key vault resource object, must be

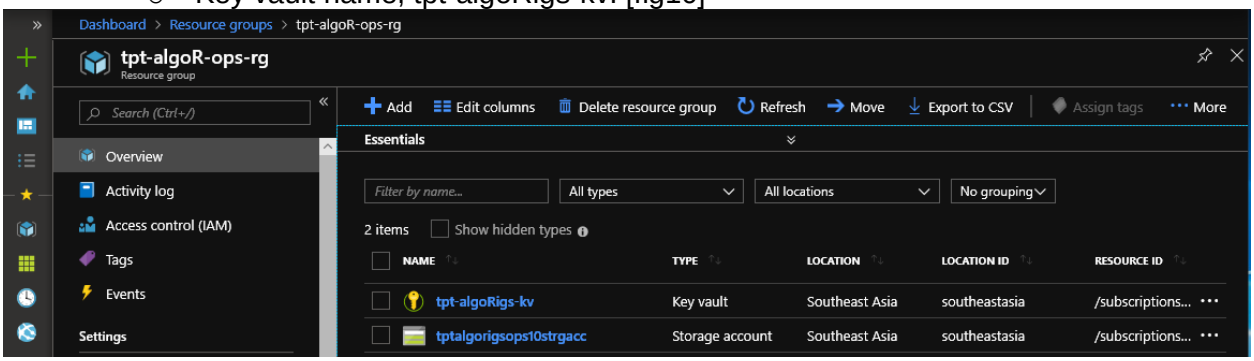
installed before virtual machine resource objects are installed. Similarly, resource group objects must be installed before virtual machine resource objects are installed.

Prior to arm rendering this design's resource specifications on the owner's arm canvas this automation design requires the two necessary resource group objects to be installed, tpt-algoR-ops-rg and tpt-algoR-vms-rg.

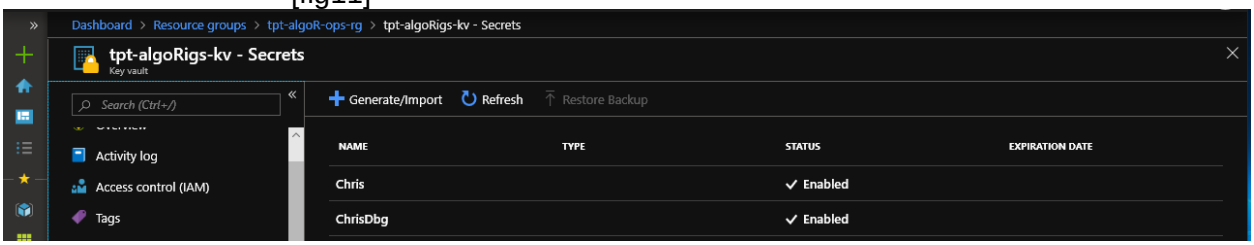
Similar to: [fig9]



- Automation operations resource group name, tpt-algoR-ops-rg.
    - Key vault name, tpt-algoRigs-kv. [fig10]



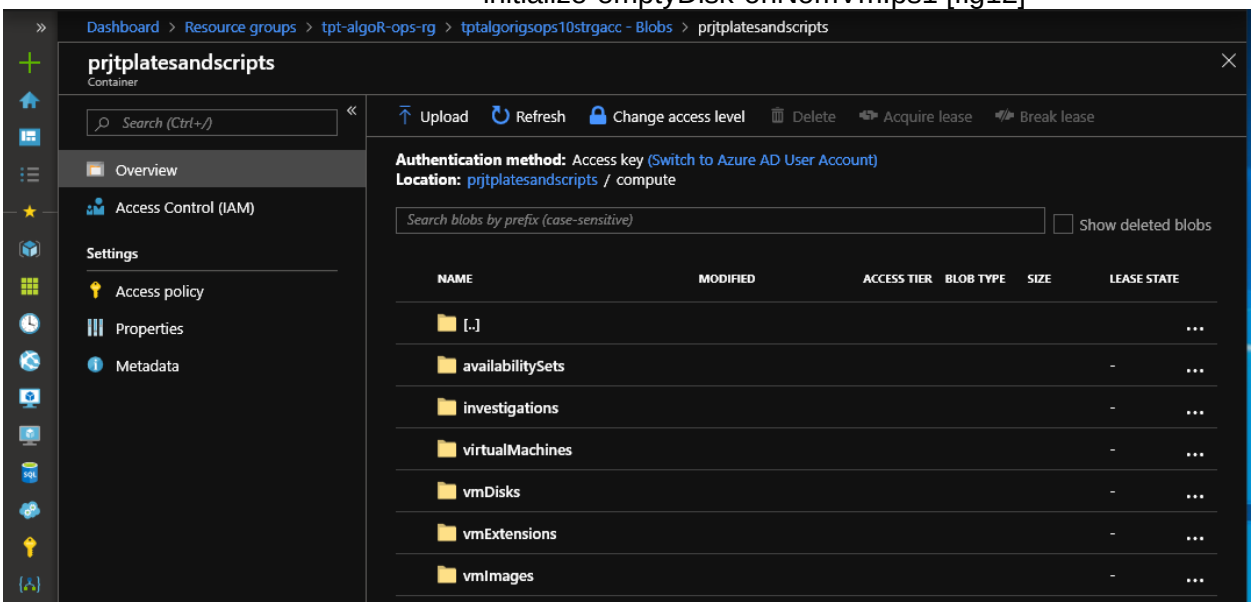
    - Virtual machine admin password secret name, kvSecretVmAdminPword. [fig11]



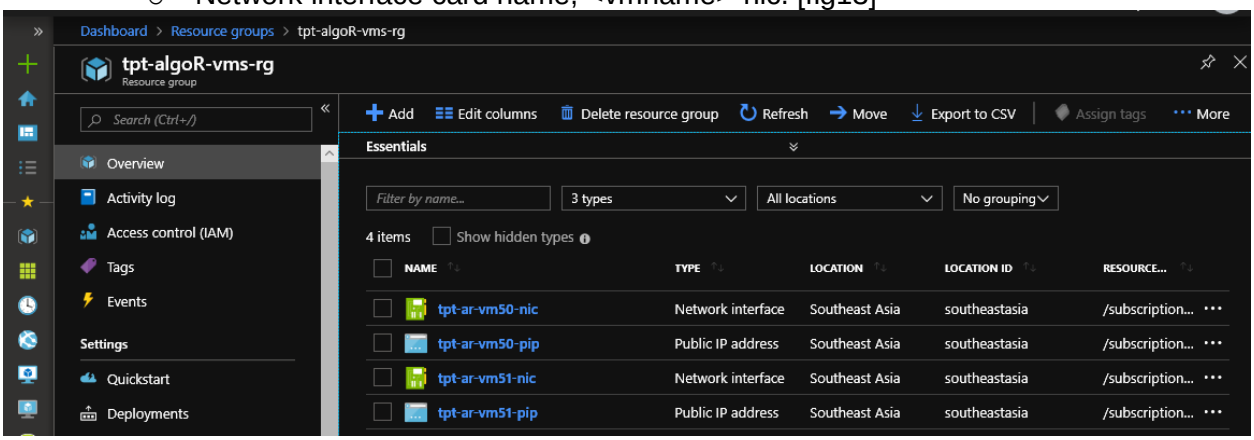
    - Storage account name, tptalgorigsops10strgacc.
        - Blob collection container name, prjtplatesandscripts. The following content needs to be stored at the nominated locations rooted in that container:
            - Nested azure task templates:
              prjtplatesandscripts/compute/virtualMachines/
                  assemble-vma-new-osmd-0toN-dmd-kv.json
                  assemble-new-dmd.json
              prjtplatesandscripts/compute/vmExtensions

install-0toN-pVmExts.json
provision-customScript-vmExts.json
provision-pVmExt-antiMalware.json
provision-pVmExt-bgInfo.json

- Virtual machine extension custom scripts:
  prjtplatesandscripts/compute/vmExtensions/vmExtScripts/winRM
  ConfigureWinRmAndLog.ps1
  initialize-emptyDisk-onNomVm.ps1 [fig12]

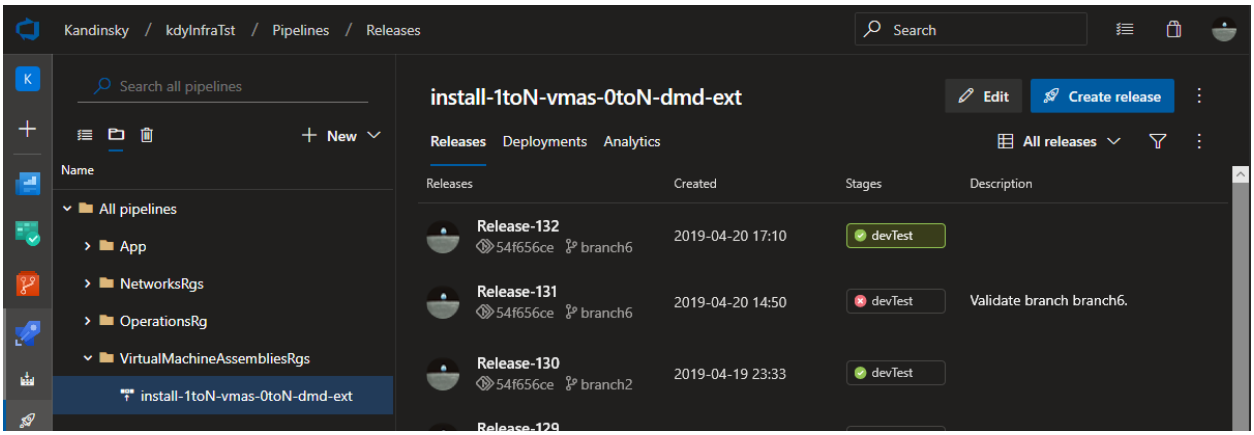


- Virtual machine assembly resource group name, tpt-algoR-vms-rg.
  - Public ip address name, <vmname>-pip.
  - Network interface card name, <vmname>-nic. [fig13]

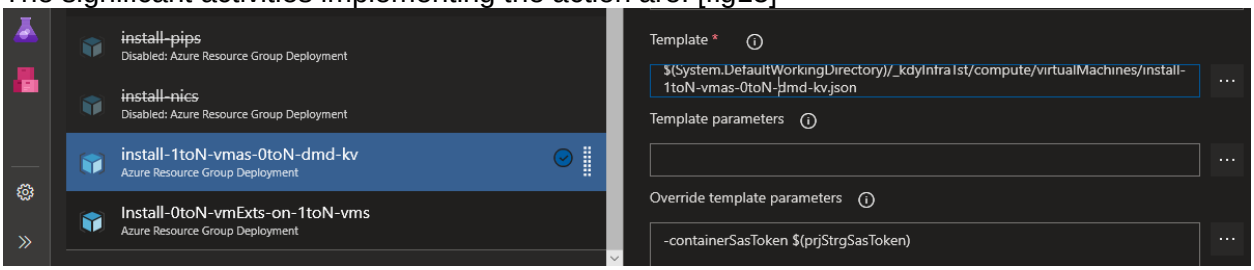


Implementation: devOps release pipeline and activities

The azure devOps release pipeline action is install-1toN-vmas-0toN-dmd-ext. [fig14]



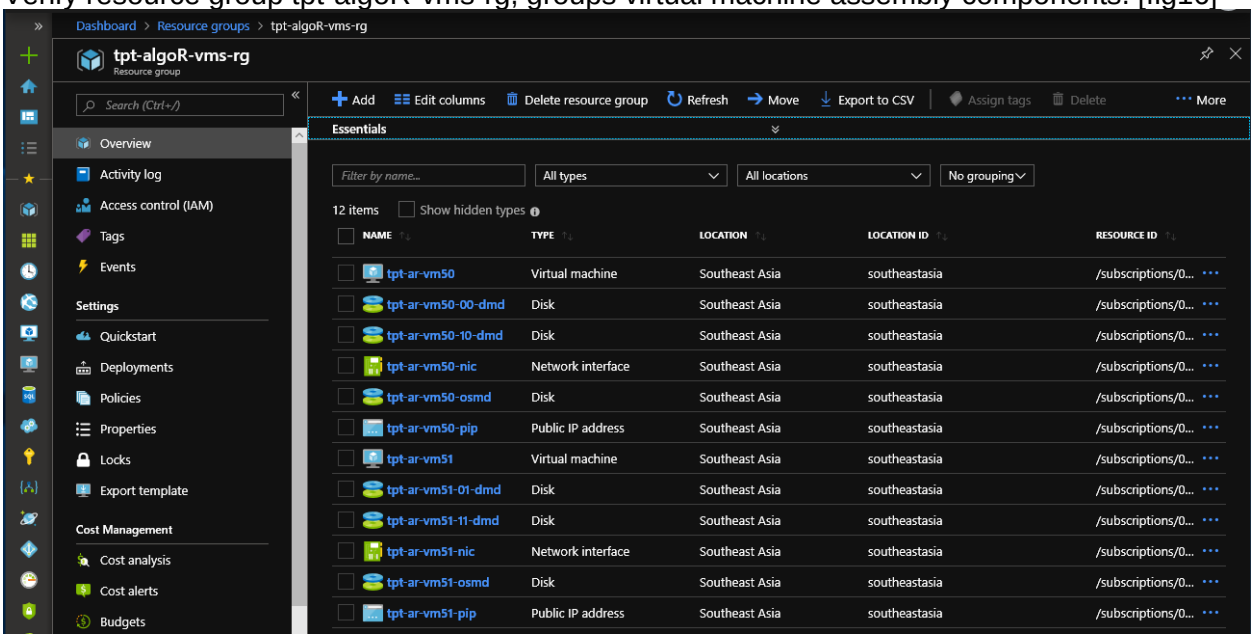
The significant activities implementing the action are: [fig15]



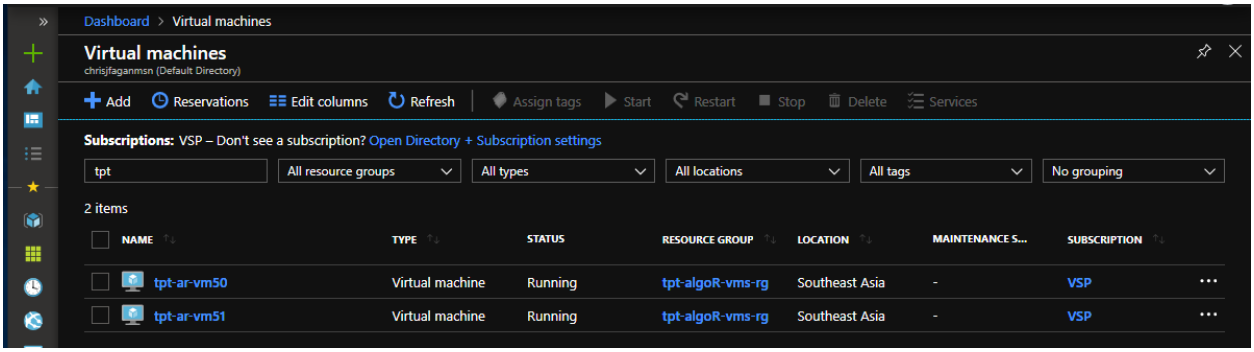
Notice the (disabled) install-pips and install-nics activities precede, and have installed their resource objects before the virtual machines and their data disks are installed. The activity install-0toN-vmExts-on-1toN-vms installs the custom script vm extensions after the virtual machines are installed.

Verify: installed virtual machine assemblies and provisioning

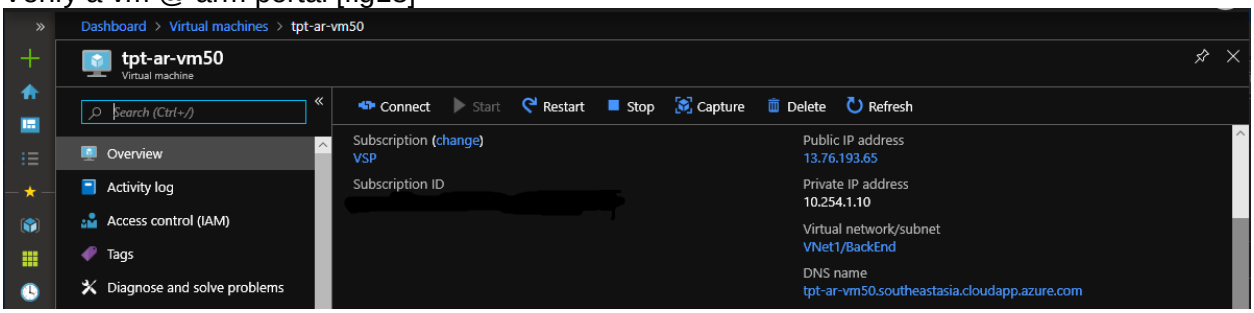
Verify resource group tpt-algoR-vms-rg, groups virtual machine assembly components. [fig16]



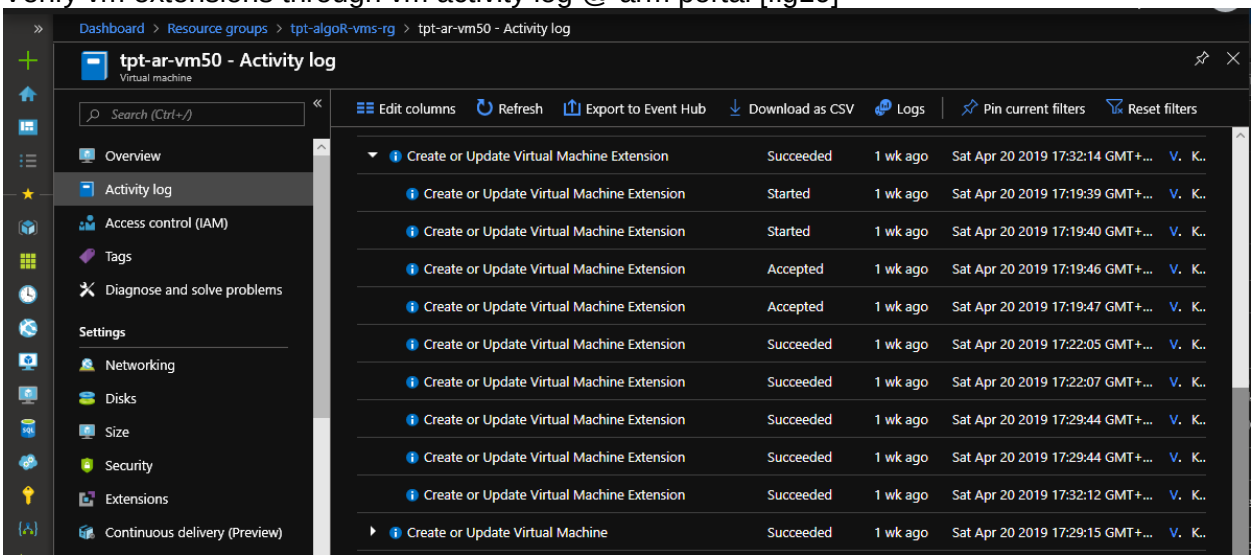
Verify vms @ arm portal [fig17]



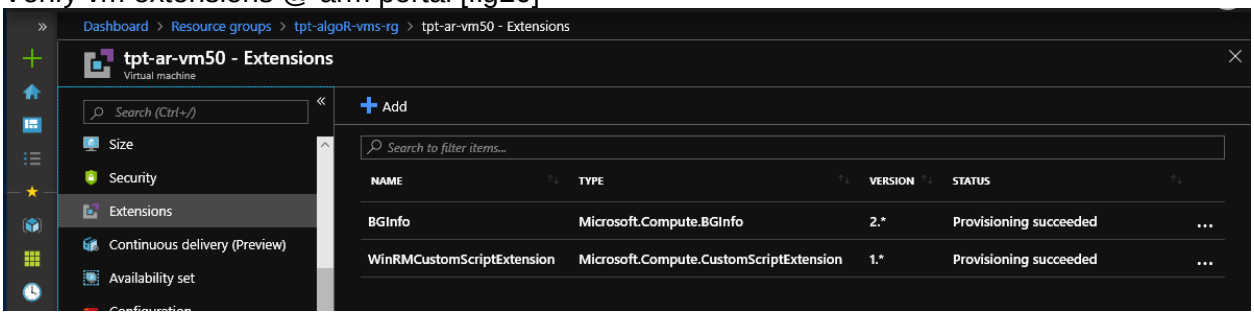
Verify a vm @ arm portal [fig18]



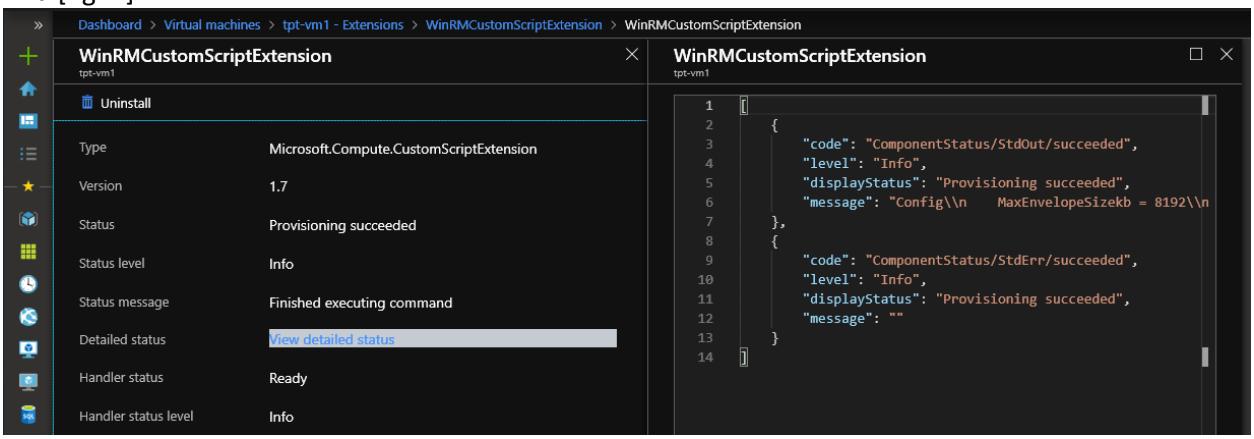
Verify vm extensions through vm activity log @ arm portal [fig19]



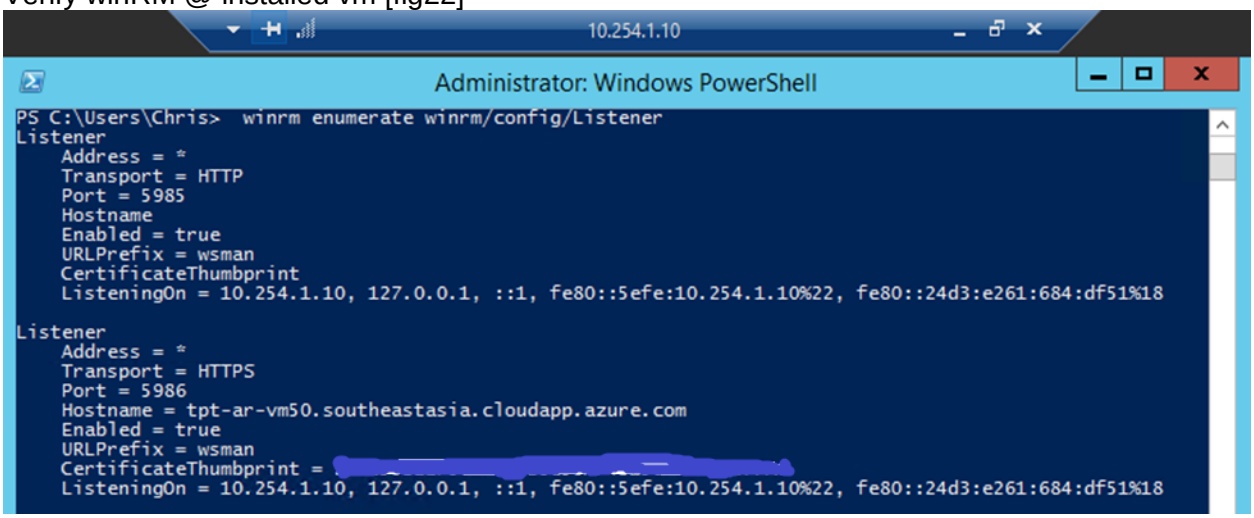
Verify vm extensions @ arm portal [fig20]



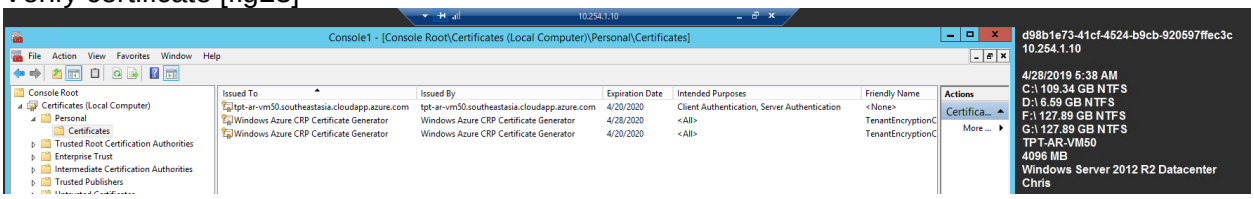
And [fig21]



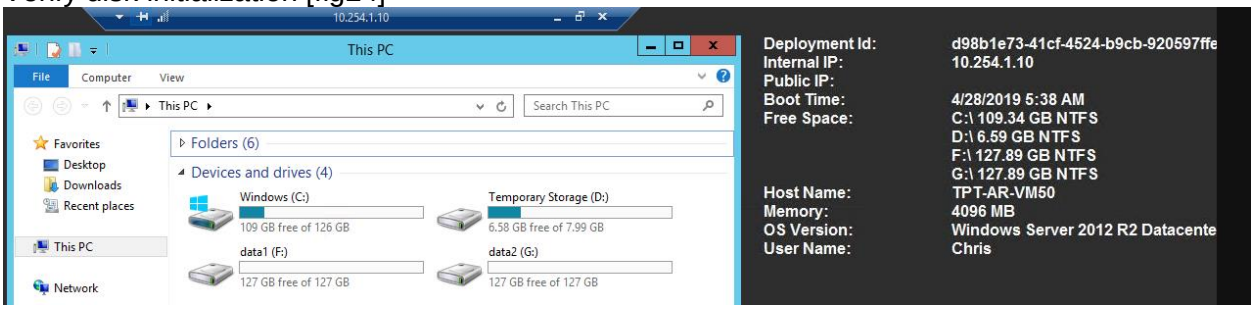
Verify winRM @ installed vm [fig22]



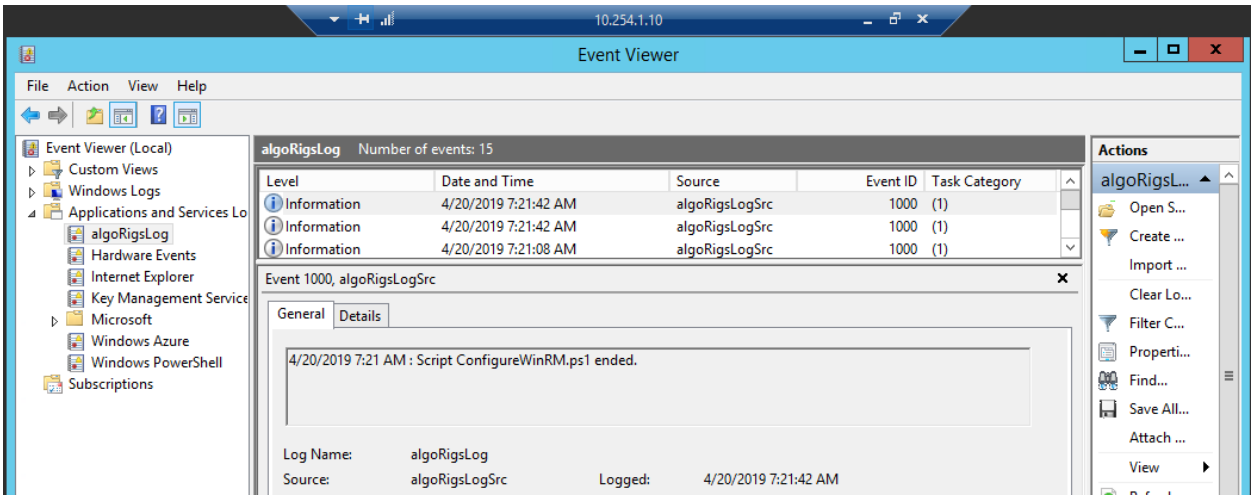
Verify certificate [fig23]



Verify disk initialization [fig24]



Verify application event log [fig25]



Resources

All information in this article is intended to provide general information only.
It does not purport to be a comprehensive advice.

An automation's resource objects can be installed on a subscription's arm canvas using these azure tools:

- Arm portal
- Command line interface, aka CLI
- Powershell
- DevOps release pipeline – the method used above.

The azure devOps task type, deploy resource group, creates a resource group if it doesn't exist.

Further guidance is here:
https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-template-deploy-portal
https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-manager-quickstart-create-templates-use-the-portal
Example templates for use with the above tools are here:
https://github.com/Azure/azure-quickstart-templates

This design's templates and scrips which are intended for elucidation not for production, are available here, under an MIT license:
https://github.com/TecProTools/azure-arm-vmas-NestedLoops