

Template Solutions in the Marketplace:

Best Practices

This document focuses on the guidance, best practices, and requirements for VM-based solutions in the Azure Marketplace. It does not touch upon solutions that include other non-compute resources, like App Services, Logic Apps, etc.

DRAFT. This document is still in draft form and is not to be distributed broadly.

Contents

Validation Criteria	3
Getting Started	3
Topology Definition & Platform Evaluation	3
Azure Compute Building Blocks	4
Developing building blocks	5
Developing ARM Templates	5
Additional Requirement and Recommendations	6
1) Standardized Minimum Set of User Input	6
2) Solution Templates should deploy multiple instances into Availability Sets and Load-balancers....	8
3) Solution Template should support deploying the VMs into a new or existing virtual network.	8
4) Template should support new or existing storage account to store the disks of a VM	11
5) Templates should take into account storage distribution	11
6) Premium Storage for Production Deployments	11
7) Local Storage Replication and Backups	12
8) Templates deploying Linux-based VMs should support using a password or SSH key.	12
9) VM Images used in a solution template must be from the Azure Marketplace	12
10) Templates should use “latest” version of a SKU	13
11) Template solution contains up-to-date and consistent API Versions across resource types	14
12) Use default values for parameters, except in a small a subset cases.	14
13) When applicable, a template should specify the allowedValues element for a parameter.	15
14) Metadata element on parameters should be specified.	15
15) Template outputs should be specified to return relevant deployment configuration	16
16) Deployment of an entire topology should generally take 90 minutes or less.	16
Appendix A: Validation Criteria Checklist	17

Appendix B: Links to Useful Articles	19
VM Images	19
VM Extensions	19
ARM	19
Storage Account Throttles	19
Compute on ARM	19

DRAFT

Validation Criteria



Mandatory—this is a mandatory best practice and is required to deploy into the Azure Marketplace for VM-based solutions.



Recommended—this is a recommended best-practice, but it may be deviated from as appropriate.



Optional—this is optional but important, and it is called out for reference.

Getting Started

As you begin to consider how to package and sell your applications in the Azure Marketplace, there is a simple framework we have created to help guide you in determining what solutions you are building for your new and existing Azure customers.

Topology Definition & Platform Evaluation

At this point, you may already be familiar with Azure compute artifacts and be thinking what exactly should I be building? VM Images, extensions, ARM templates?

Before trying to map your application to Azure artifacts, it will help to understand the different topologies that make sense for your solution that you would likely also recommend to a customer who considers deploying your application on-premises. Most enterprises generally go through some variation of the following four stages: Evaluation, Proof of Concept, Pre-Production, and Production. Usually, you are working with them through these different stages evaluating their requirements and recommending the exact layout and scale needed for their scenarios.

While every customer isn't the same, generally we expect that there are common topologies and sizes that you recommend given the size of the customer deployment. So even before beginning to develop any solutions, think through the different layouts that make sense for your application and what could meet a fair share of your customer's needs without major customizations. Again, we realize that there will always be customers who need something very exact, but we are looking for the common case.

For each type of deployment – Evaluation, Proof of Concept, Pre-Production, and Production – think through the exact layout and topology needed for your application. What are the different tiers? What resources are needed for each tier and of what size? What are the building blocks needed to create these resources? A production deployment, or even a dev/test environment, can vary significantly in size depending on the customer and their needs. Decide upon what the various sizes would look like for your workload and think through availability, scalability, and performance requirements needed to be successful.

Once you have thought through the different topology layouts and corresponding sizes (you may have most of this already complete with your experiences with on-premises deployments with customers), it's time to start bringing it back to what Azure has to offer.

Evaluate your topology requirements against what Azure Resource Manager (ARM) and Compute on Azure Resource Manager offers. While most customers will start with smaller deployments, ***we strongly recommend that you evaluate the performance, scale, and capability requirements of your large topology on the Compute on Azure Resource manager even if you may not initially be onboarding a large topology.*** This is mostly to ensure that as customers begin to adopt your smaller solutions they have a substantial amount of room to grow successfully to larger scale deployments. Any scale, performance, or capability limitations should be raised to your Microsoft counterparts to ensure that we continue to improve in the areas needed by you, our Marketplace partner, and in turn by your customers.

Further introductory information on Azure Resource Manager (ARM) can be found [HERE](#).



Recommended—Evaluate and build a template to match the largest expected deployment size of your application in order to offer growth opportunities to your end customers.

You can find the Compute on Azure Resource Manager compatibility matrix [HERE](#).

Azure Compute Building Blocks

Let's start off by identifying what compute artifacts are available to you and how to think about your application in terms of these artifacts.

VM Image

VM Images are the complete storage profile of a virtual machine, containing one operating system VHD and zero or more data disks. You can create a VM Image to package your software application with the operating system bits, in addition to any additional data needed. For example, today in the Azure Marketplace, there are a handful of VM Images for SQL Server, which package the Windows operating system with the SQL Server application. While this document will not go into the details of creating and onboarding VM Images into the Azure Marketplace, it is important to understand what a VM image is, since it will likely be the building block to your application solution.

VM Extension

In the Azure Marketplace today, there are a handful of operating system companies who publish Windows, Linux, and Unix operating systems on a regular basis. You can find images for Windows Server, Ubuntu, CentOS-based, CoreOS, Oracle Linux, etc. Instead of publishing your own VM Image, packaging both the operating system and application, you can consider writing a VM Extension Handler or leveraging an existing one. A VM extension is a software mechanism to simplify VM configuration and management, which leverages the guest agent (a secured, light-weight process) in an Azure VM. For example, the Custom Script Extension, takes in a custom script as the name suggests, and executes it within the VM. This allows any sort of configuration that a generic script would allow. For more information regarding Extension Handlers, please contact your Microsoft onboarding representative.

In a traditional image scenario, you would have installed all the bits on the image from the start, done any necessary configurations before or even possibly after VM start up (with a startup task). This image

could get customized and would not necessarily be easily re-usable. In some cases, this makes the most sense. In others, especially with light-weight applications or more complicated configurations which require being done after the VM is provisioned, the extension model can help. Sometimes, customers deploy a combination of extensions and VM images to have both pre-installed solutions but also enable complex post-start-up configuration.

Images and extensions are quite useful to configure a single VM. However, one VM is not sufficient to offer customers high availability in the cloud and scale to meet the needs of growing traffic. Therefore, for any production workload, we highly recommend planning for a multi-VM high availability solution using availability sets and load-balancing.



Recommended—All production workloads in the marketplace should be offered using multiple Virtual Machines serving traffic into an Availability Set and with a load-balancer.

To support these numerous deployment topologies your application can support and recommend, the Azure Marketplace offers the Azure Resource Manager and ARM templates. Azure Resource Manager deploys and manages the lifecycle of a collection of resources through declarative, model-based template language. This template is simply a parameterized JSON file which expresses the set of resources and their to be used for deployments. Each resource is placed in a resource group, which is simply a container for resources. The Resource Manager also offers a set of very valuable additional capabilities to end-customers, including centralized operation auditing, resource tagging through to the customer bill, and Role-Based Access Control for granular security.



Mandatory —Given the benefits for end-customers, all new marketplace content must be deployed using Azure Resource Manager.

With ARM and ARM templates, you can express more complex configuration of Azure resources, such as Virtual Machines, Storage Accounts, and Virtual Networks, which build upon Marketplace content such as VM Images and VM Extensions.

Developing building blocks

Once you have determined your recommended application topologies and which ones you will develop and onboard to the Marketplace in your first stage, it is time to identify and develop the VM building blocks. As discussed above, you will have two options. First, you can choose to package the contents of the VM as your own published VM Image. Alternatively, you can choose to leverage an already existing VM Image in the Azure Marketplace; configuring it with a published extension handler or with your own handler. There isn't a single correct solution; it is highly dependent how much control you want around the underlying operating system and requirements on how you install, configure, and manage your application. You can find additional information on images and extensions in the list of potentially useful articles in Appendix B.

Note, that for any image or extension handler used, it must first be published through the Marketplace. To find more information about building and publishing VM Images, you can go [here \[provide link to where our current onboarding document is hosted\]](#).

Developing ARM Templates

If you are new to Azure Resource Manager and ARM Templates, it may be worthwhile to review some of the documentation around Azure Resource Manager and Compute on Azure Resource Manager. The

guidance and examples provided below assume a basic level of understanding of the ARM template language and how to use Compute with ARM. This section is mostly devoted to the best practices and requirements to help you develop good templates specifically for the Azure Marketplace. These take into account the requirements that allow for a unified, easy-to-use, and relevant experience for customers from both the Portal and programmatically. If interested, a list of good resources about ARM and Compute on ARM in general can be found in Appendix B.

Your solution will likely be expressed with a set of templates. These templates are logical pieces of your topology. There are many reasons why to break-up the topology expression into multiple files, from ease of development and manageability, to reusability of certain subsets, to modular testing and expansion of template solutions. After working with many partners, the recommended breakdown of templates is to divide the topology into a template for common resources (such as VNets, storage accounts, etc.), called the Shared Resource template, zero or more templates for optional resources, called the Optional Resource templates, and one or more templates for the topology specific resources, called Known Configuration Resource templates. You can find more information about this type of template linking in [Marc's document](#) in Appendix A.

A complete template solution in the Marketplace will consist of the following items:

1. One Main Template (mainTemplate.json) – This is the entry point to the overall deployment expressed in the ARM JSON templating file. It is the first file that is evaluated for the template solution and the one used to stitch together (link) all the rest of the templates. This file must also contain all desired input (parameters) from the end user.
2. Zero or More Linked Templates – The rest of the templates that are linked from the main template to express the complete solution topology.
3. Zero or More Script Files – Any relevant script files which are expected as input to extensions to be used to configure a virtual machine.
4. One Create UI Definition File (createUiDefinition.json) – This is the file which maps the parameters in the ARM template to elements in the user experience. This file allows partners to have control on the UI rendered to their end customers in the Portal. All the inputs required from the end user should provide will be expressed in this file. Each entry allows you to control all the UI aspect associated with each input e.g. UI elements, defaults, validation rules, error messages, etc.

As you begin to develop each of these files to express your complete topology, consider the following criteria to ensure the best template for your Marketplace offering.

Additional Requirement and Recommendations




1) Standardized Minimum Set of User Input

Parameters allow you to express values that should be provided by the user at time of deployment in the template. User input can vary from application to application, and even from topology tier to tier. The parameter list needs to provide a balance between asking fewer required questions and providing the user the right knobs to control their deployment. While there are no requirements on the full list of

user input, we want to ensure that all Marketplace template solutions have a set of common input that many customers want to control.

Ensure that you have defined parameters for the following values:

- a. **Location** – the Azure region where resources are deployed
- b. **Virtual Network Name** – For deployments that create a new Virtual Network, the name to use for creating that resource. For deployments that use an existing Virtual Network, the name of the VNet to deploy into.
- c. **Storage Account** – For deployments that create a new storage account, the storage account name to create. For deployments that use an existing storage account, the name of the storage account to use.
- d. **Username** – User name for the virtual machine(s) and potentially the application(s). More than one user name can be requested from the end user, but at least one must be prompted.
- e. **Password** – Password for the virtual machine(s) and potentially the application(s). More than one password can be requested from the end user for different VMs or applications, but at least one must be prompted.
- f. **SSH Public Key** (only for Linux VMs) – SSH Key for the virtual machine(s). More than one key can be requested from the end user for different VMs, but at least one must be prompted.

-  **Mandatory** —All application templates must have Location, Virtual Network Name, and Storage Account.
-  **Mandatory** —Applications cannot use a pre-defined password embedded in the image, due to security concerns of unauthorized access to the VM.
-  **Optional**—End customers prefer to have as few parameters as possible to make customer deployments as easy and fast to launch as possible.

A snippet of what the parameters section in the main template could like for these parameters is below:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "location": {
      "type": "string",
      "allowedValues": [
        "West US",
        "East US",
        "West Europe",
        "East Asia",
        "Southeast Asia"
      ],
      "metadata": {
        "description": "The region in which to deploy the resources."
      }
    },
    "newVirtualNetworkName": {
      "type": "string",
      "metadata": {
        "description": "The name of the new Virtual Network to create."
      },
      "defaultValue": "siosVNET"
    }
  },
}
```

```

    "newStorageAccountName": {
      "type": "string",
      "metadata": {
        "description": "The name of the new storage account to create to store virtual machine
disks."
      }
    },
    "username": {
      "type": "string",
      "metadata": {
        "description": "The name of the administrator of the new virtual machines."
      },
      "defaultValue": "VMAdmin"
    },
    "password": {
      "type": "securestring",
      "metadata": {
        "description": "The password for the administrator account of the new virtual machine."
      }
    },
    "sshPublicKey" :{
      "type": "securestring",
      "metadata": {
        "description": "The SSH Public Key for the administrator account of the new virtual
machine."
      }
    }
  }
}

```

2) Solution Templates should deploy multiple instances into Availability Sets and Load-balancers

For any non-evaluation topologies, customers will expect the application to be highly available, once deployed. To enable this, deploying instances serving a similar purpose (three front-ends or multiple data back-ends) into the same availability set. Furthermore, with updates to the Azure Resource Manager, applications can now deploy with 3 Fault Domains, increasing availability even more. This will spread those like-function instances across multiple physical failure points, guaranteeing that if one instance loses connectivity due to infrastructure failure, the other will remain active to receive traffic. In addition to using availability sets, using a load balancer, whether internal or external, will offer traffic failover as well.



Recommended—All production workloads in the marketplace should be deployed using multiple Virtual Machines serving traffic into an Availability Set and with a load-balancer.

[We need a sample from Mahesh or Drew?]

3) Solution Template should support deploying the VMs into a new or existing virtual network.

For non-evaluation topologies, many customers will want to deploy a topology into one of their existing virtual networks. Depending on the customer, virtual network creation may even be locked down to only a handful of administrators with the appropriate RBAC privileges. As such, it is important for Marketplace templates to support both workflows – creation of a new virtual network or use of an existing virtual network as part of the solution template.



Mandatory —All Virtual Machines deployed using the Azure Resource Manager must be deployed into a Virtual Network, either pre-existing or new.



Recommended— Your application should be able to deploy into a pre-existing Virtual Network or deploy into a new Virtual Network (we have samples for both).

While the ARM templating language does not explicitly support an if conditional to support declarative behavior, it is possible with the use of parameters and linked templates to support this scenario.

An example of how to create a new or existing virtual network as part of a topology is illustrated below:

Main Template:

Depending on the value of the 'newOrExisting' parameter, the template calls two different linked templates. One template newvnet.json, creates a new virtual network, the other template existingvnet.json, simply outputs the existing virtual network information.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "newOrExisting": {
      "type": "string",
      "allowedValues": [
        "new",
        "existing"
      ],
      "metadata": {
        "description": "Whether a new or existing virtual network is needed."
      }
    },
    "virtualNetworkName": {
      "type": "string",
      "metadata": {
        "description": "The name of the virtual network to create or the name of an existing virtual network to use."
      }
    },
    "location": {
      "type": "string",
      "allowedValues": ["East US", "West US", "West Europe", "East Asia", "South East Asia"],
      "metadata": {
        "description": "Location in which to create the Virtual Network"
      }
    }
  },
  "variables": {
    "storageAccountName": "templatestorageaccount",
    "templatelink": "[concat('https://', variables('storageAccountName'), '.blob.core.windows.net/templates/', parameters('newOrExisting'), 'vnet.json')]"
  },
  "resources": [
    {
      "apiVersion": "2015-01-01",
      "name": "nestedTemplate",
      "type": "Microsoft.Resources/deployments",
      "properties": {
        "mode": "incremental",
        "templateLink": {
          "uri": "[variables('templatelink')]",
          "contentVersion": "1.0.0.0"
        },
        "parameters": {
          "location": {
            "value": "[parameters('location')]"
          },
          "virtualNetworkName": {
            "value": "[parameters('virtualNetworkName')]"
          }
        }
      }
    }
  ]
}
```

```
]
}
```

Linked Template 1: (the one referenced above that is stored at:

'https://templatestorageaccount.blob.core.windows.net/templates/newvnet.json')

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "location": {
      "type": "string",
      "allowedValues": ["East US", "West US", "West Europe", "East Asia", "South East Asia"],
      "metadata": {
        "description": "Deployment location"
      }
    },
    "virtualNetworkName": {
      "type": "string",
      "metadata": {
        "description": "The name of the virtual network to create or the name of an existing virtual network to use."
      }
    }
  },
  "variables": {
    "addressPrefix": "10.0.0.0/16",
    "subnetPrefix": "10.0.0.0/24",
    "subnetName": "subnet1"
  },
  "resources": [
    {
      "apiVersion": "2015-05-01-preview",
      "type": "Microsoft.Network/virtualNetworks",
      "name": "[parameters('virtualNetwork')]",
      "location": "[parameters('location')]",
      "properties": {
        "addressSpace": {
          "addressPrefixes": [
            "[variables('addressPrefix')]"
          ]
        },
        "subnets": [
          {
            "name": "[variables('subnetName')]",
            "properties": {
              "addressPrefix": "[variables('subnetPrefix')]"
            }
          }
        ]
      }
    }
  ]
}
```

Linked Template 2: (the one referenced above that is stored at:

'https://templatestorageaccount.blob.core.windows.net/templates/existingvnet.json')

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "location": {
      "type": "string",
      "allowedValues": ["East US", "West US", "West Europe", "East Asia", "South East Asia"],
      "metadata": {
        "description": "Deployment location"
      }
    },
    "virtualNetworkName": {
      "type": "string",
      "metadata": {
        "description": "The name of the virtual network to create or the name of an existing virtual network to use."
      }
    }
  },
  "resources": [
    {
      "apiVersion": "2015-05-01-preview",
      "type": "Microsoft.Network/virtualNetworks",
      "name": "[parameters('virtualNetwork')]",
      "location": "[parameters('location')]",
      "properties": {
        "addressSpace": {
          "addressPrefixes": [
            "[variables('addressPrefix')]"
          ]
        },
        "subnets": [
          {
            "name": "[variables('subnetName')]",
            "properties": {
              "addressPrefix": "[variables('subnetPrefix')]"
            }
          }
        ]
      }
    }
  ]
}
```


```

    },
    "variables": {},
    "resources": [],
    "outputs": {
      "virtualNetworkInformation": {
        "value": "[reference(parameters(virtualNetworkName),providers('Microsoft.Network',
'virtualNetwork').apiVersions[0]))]",
        "type": "object"
      }
    }
  }
}

```

4) Small topology templates should support new or existing storage account to store the disks of a VM


Similar to Item #2), with a newOrExisting parameter a new or existing storage account can be created or used, respectively.

 **Recommended**— Your application should be able to deploy into a pre-existing storage account or create a new storage account.

If your topology requires multi-storage accounts to distribute disks for scale and performance, the new and existing pattern may not be required. This can be reviewed on a case-by-case basis.

5) Templates should take into account storage distribution


When deploying large scale topologies with many virtual machines, it is important to consider the best practices around the storage accounts that back these virtual machines. Not only is it important to think about the single disk IOPS, but also about hard storage account limits for overall bandwidth. When designing your deployment architecture for your solution, it is important to keep these limits in mind. Your template should distribute high IO disks across multiple storage accounts to avoid platform throttling.

 **Recommended**— Your application should deploy across multiple storage accounts, where required, limiting deployment to just 40 disk per account (Standard storage) and 32 disks per account (premium storage).

For standard storage, the limit is 20,000 IO units, roughly equating to 40 disks per storage account. For premium storage, it is 50 gigabits per second for inbound and outbound, roughly equating to 32 disks per storage account. More information about this topic can be found in Appendix A.

6) Premium Storage for Production Deployments

Besides paying attention to storage distribution, being selective in the type of storage you use to back the virtual machines in your topology is important. We recommend being selective in the VM series you support and to ultimately match the VM size selection to the type of deployment in your solution.

 **Recommended**— It is highly recommended that applications use premium storage for both OS and data disks for the application, both dev/test and production-based deployments. This will improve both the performance of the application and the availability of the instances.


Specifically for production scale workloads, we highly recommend Premium Storage Virtual Machines (the DS Series), backed by Premium storage, which are explicitly designed to support I/O intensive

workloads with high-performance, low-latency disks. Premium Storage also increases VM availability when using it for persistent disks.


More information about VM sizes and their specifications can be found [here](#).

7) Local Storage Replication and Backups

If your application uses local storage (non-persistent SSDs) and the data is intended to be persisted, it comes with a risk that the data may be lost due to failed VMs. Thus, it is required that your application replicate across at least 3 instances in 3 fault domains, to reduce this risk.


 **Mandatory** — Applications using local disk storage for data intended to be persisted must replicate at least 3 times across 3 fault domain instances.

Furthermore, it is highly recommended that your application take regular back-ups to persisted storage in order to reduce the impact of lost data, if the unexpected does happen.

 **Recommended** — It is highly recommended that if an application uses local storage, replicated across multiple instances, for persistent data, that the application takes regular back-ups to persisted storage (like Azure Storage blobs) to reduce the impact of data loss.

8) Templates deploying Linux-based VMs should support using a password or SSH key.


Similar to Item #2), with a password/SSH key parameter a Linux VM can be provisioned using a password or SSH key, respectively.

 **Mandatory** — Linux images are recommended to have both SSH Public Key and username/password as parameters to offer customers choice.

 **Recommended** — Windows images should have username/password.

9) VM Images used in a solution template must be from the Azure Marketplace

Template solutions that are in the Azure Marketplace will be globally available and deployable. As such, we want to ensure that any building block which is used as part of the solution has the same global availability. The VM Images in the Azure Marketplace are backed by a global platform image repository. When a VM image is published into the Azure Marketplace, it is globally replicated in a way to ensure fast deployment of virtual machines leveraging this content.

 **Mandatory** — In order to give customers the best experience, all images referenced in templates must use VM Images from the Azure Marketplace.

When creating a virtual machine via ARM, the image reference object in the properties section must refer to an Azure Marketplace image. An example of how to reference an operating system image from the Azure Marketplace, like the Windows Server image, is shown below.

```
{
  "apiVersion": "2015-05-01-preview",
  "type": "Microsoft.Compute/virtualMachines",
  "name": "[parameters('vmName')]",
  "location": "[parameters('location')]",
  "properties": {
    "hardwareProfile": {
      "vmSize": "[parameters('vmSize')]"
    },
    "availabilitySet": {
```

```

        "id": "[resourceId('Microsoft.Compute/availabilitySets',
parameters('adAvailabilitySetName'))]"
    },
    "osProfile": {
        "computername": "[parameters('vmName')]",
        "adminUsername": "[parameters('adminUsername')]",
        "adminPassword": "[parameters('adminPassword')]"
    },
    "storageProfile": {
        "imageReference": {
            "publisher": "MicrosoftWindowsServer",
            "offer": "WindowsServer",
            "sku": "2012-R2-Datacenter",
            "version": "latest"
        },
        ...
    }
    ...
}

```

When using a VM Image containing a third party application, you must express your intent to purchase the image by also providing 'Plan' information. The Plan expresses that a transaction needs to take place and what exactly is being transacted. If the plan is omitted, Compute plane on Azure Resource Manager will stop the VM deployment. An example of how to reference a third party application image from the Azure Marketplace, such as image, is shown below

```

{
    "apiVersion": "2015-05-01-preview",
    "type": "Microsoft.Compute/virtualMachines",
    "name": "[parameters('vmName')]",
    "plan": {
        "name": "csmav10",
        "publisher": "commvault",
        "product": "commvault"
    },
    "location": "[parameters('location')]",
    "properties": {
        "hardwareProfile": {
            "vmSize": "[parameters('vmSize')]"
        },
        "availabilitySet": {
            "id": "[resourceId('Microsoft.Compute/availabilitySets',
parameters('adAvailabilitySetName'))]"
        },
        "osProfile": {
            "computername": "[parameters('vmName')]",
            "adminUsername": "[parameters('adminUsername')]",
            "adminPassword": "[parameters('adminPassword')]"
        },
        "storageProfile": {
            "imageReference": {
                "publisher": "commvault",
                "offer": "commvault",
                "sku": "csmav10",
                "version": "latest"
            },
            ...
        }
        ...
    }
}

```

10) Templates should use "latest" version of a SKU

When creating a virtual machine from an image, you need to specify a single image. This can be done by explicitly specifying the <Publisher, Offer, SKU, Version> 4-tuple. However, whenever a new version is published or an older version taken down, you have to keep changing your image reference and thus template.



Recommended—When specifying a VM image (Windows or Linux), application templates should reference “latest” as the version of the image to keep the template current with the latest patched platform images.

Compute on Azure Resource Manager supports the concept of “latest” for image references. “Latest” allows you to reference the “latest” version (i.e. image) in the SKU, rather than taking a hard dependency on a particular version.

Especially when you are looking to reference a SKU that you did not publish, it is important to not take these hard dependencies, since SKUs are regularly updated and versions added or removed. Explicitly referencing a version of a SKU will undoubtedly lead to a broken template at some point in time.

The examples in Item #0 already illustrate how to reference the “latest” image in the SKU.

11) Template solution contains up-to-date and consistent API Versions across resource types

New functionality and bug fixes are added to newer versions of the API regular for all resource providers – Compute, Storage, and Networking. For Marketplace content, it is important to leverage this up-to-date platform functionality, since customers expect the partner provided content to be the recommended approach.

For each resource type, the Latest-2 versions of the API can be used. The same version for a given resource type should be used consistently within all templates (main and linked). In addition, no preview APIs should be used.



Recommended—In the template being deployed, the latest two versions of the API should be used and the same API version should be used consistently across the template.

12) Use default values for parameters, except in a small a subset cases.

For most parameters, a default value can make sense. It helps the end user with a suggested value, if they are uncertain with what to enter, and it makes for an even simpler deployment experience, especially when there are many configurations that you want to surface back to the end user. Some end users will want to configure everything, while others, will want to be able to have as close to a “single-click” deployment experience as possible.



Optional—Using defaults for template parameters make it significantly easier for end-customers to deploy quickly and with the correct options for optimal experience. It is recommended that applications use as many defaults as possible for parameters.

Even though the Multi-VM UI will calculate appropriate defaults based on the subscription, resource group, and location and some similar elements will also be defined in the createUiDefinition.json file, many customers may actually deploy these solution templates programmatically, either referencing the location of the mainTemplate.json file or downloading the template file. For these end users, having a well formulated template will be very helpful.

There are a few cases where it does not make sense to provide a default value, so good judgement should always be used to ensure no defaults are provided for “secrets.” Specifically, some common parameters where no default values should be provided include:

- a. Location – Some end users may never change defaults, leading to directional load to one region
- b. Storage Account – Storage account names need to be unique.
- c. **DNS Name** – Domain name of the virtual machine needs to be unique.
- d. Password – An obvious one, since this is a user secret. We do not want users who never change defaults to be left with insecure passwords across deployments.
- e. SSH Key - An obvious one, since this is a user secret.



Mandatory Applications must not supply any defaults for location, storage account name, password, or SSH Key.

13) When applicable, a template should specify the `allowedValues` element for a parameter.

Parameters are important and useful and provide flexibility when used correctly to customize deployments to meet an end user's needs. While this flexibility is powerful, it is important to also ensure a successful end customer experience. By using `allowedValues` for a parameter, you can reduce the surface area of incorrect input values an end customer can input, and thus mistakes an end customer can make, when deploying a topology.



Recommended—All parameters should include `allowedValues`, to guide customers to select the right options before deploying the template. Once the template is launched, it may take a large amount of time to return failures back to customers.

A few cases that we suggest that allowed values should always be specified for parameters are:

- f. VM Sizes – if you have a parameter or parameters which represent an Azure virtual machine size, it is important that the applicable sizes be outlined as part of the allowed values for that parameter. If there are no parameters defined for VM sizes, this will not apply.
- g. Location – Compute stack on Azure Resource Manager is available in most, but not all Azure regions. It is important to restrict deployment locations in the template to be those regions where the Compute stack on Azure resource manager is available.

14) Metadata element on parameters should be specified.

While all of the portal facing metadata will be specified in the `createUiDefinition.json` file, it is important to still have good descriptions for the parameters in the template itself, since end users may decide to use the template programmatically, rather than going through the Azure Preview Portal. An end user should be able to decipher what values to pass to the template, from the parameter names, parameter metadata, and allowed values. Therefore, any information that is relevant for the parameters in the UI, you should also consider to be relevant for the metadata on the parameters themselves.



Optional—Metadata on the parameters is helpful hints for customers when they deploy your application programmatically.

15) Template outputs should be specified to return relevant deployment configuration

Every topology will have a certain set of configuration information, some provided by the end user, some derived from user input, and some simply decided upon by the design of the topology. Once an end user deploys the topology, he/she may need information to use the topology. This information could include endpoint or URLs to which to connect, such as a SQL connection string, default usernames used for the application, etc. It tends to be very specific to your topology and application.

To ensure the best experience for a customer, so that they do not need to guess these needed values or read through extensive documentation describing the topology or to remember this information long after deployment, it is important to populate the Output section of the template with relevant information. The Output section allows you to specify any number of values and the type corresponding to that value.



Recommended—Supplying information in the Output section will help customers debug issues as part of their deployment of your application.

A snippet of what the Outputs section in the main template could like for a very simple example is below:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "outputs": {
    "appUserName" : {
      "value" : "appAdmin",
      "type" : "string"
    },
    "appURL" : {
      "value" : "[concat('http://', parameters('dnsName'), ':8080')]",
      "type" : "string"
    }
  }
}
```

16) Deployment of an entire topology should generally take 90 minutes or less.

An ARM Template is executed in a series of steps, with a maximum allotted time of 2 hours per step. These steps are loosely correlated to the distinct segments of execution of your template that are articulated through the dependencies described with the dependsOn element.



Recommended— It is strongly recommended that the application template finishes in less than 2 hours. Wanting to ensure consistency in deployment times and not running up against the 2 hour limit, we recommend template deployments not taking longer than 90 minutes per step.



Recommended— To ensure a reasonable customer experience, we also recommend deployments take less than 2 hours in the majority of cases. For exceptional large deployments, we can work with you to understand what the right value should be.

Appendix A: Validation Criteria Checklist

The following is a compiled list of all Mandatory, Recommended, and Optional Criteria contained within this document.



Mandatory—this is a mandatory best practice and is required to deploy into the Azure Marketplace for VM-based solutions.

- ☐ Given the benefits for end-customers, all marketplace content must be deployed using Azure Resource Manager templates.
- ☐ All application templates must have Location, Virtual Network Name, Storage Account, and DNS Name.
- ☐ Applications cannot use a pre-defined password embedded in the image, due to security concerns of unauthorized access to the VM.
- ☐ All Virtual Machines deployed using the Azure Resource Manager must be deployed into a Virtual Network, either pre-existing or new.
- ☐ In order to give customers the best experience, all images referenced in templates must use VM Images from the Azure Marketplace.
- ☐ Applications using local disk storage for data intended to be persisted must replicate at least 3 times across 3 fault domain instances.
- ☐ Applications must not supply any defaults for location, storage account name, password, or SSH Key.



Recommended—this is a recommended best-practice, but it may be deviated from as appropriate.

- ☐ Evaluate and build a template to match the largest expected deployment size of your application in order to offer growth opportunities to your end customers.
- ☐ All production workloads in the marketplace should be deployed using multiple Virtual Machines serving traffic into an Availability Set (with three fault domains) and with a load balancer.
- ☐ Your application should be able to deploy into a pre-existing Virtual Network or deploy into a new Virtual Network (we have samples for both).
- ☐ Your application should be able to deploy into a pre-existing storage account or create a new storage account.
- ☐ Your application should deploy across multiple storage accounts, where required, limiting deployment to just 40 disk per account (Standard storage) and 32 disks per account (premium storage).
- ☐ It is highly recommended that applications use premium storage for both OS and data disks for the application, both dev/test and production-based deployments. This will improve both the performance of the application and the availability of the instances.
- ☐ It is highly recommended that if an application uses local storage, replicated across multiple instances, for persistent data, that the application takes regular back-ups to persisted storage (like Azure Storage blobs) to reduce the impact of data loss.
- ☐ Linux images are recommended to have both SSH Public Key and username/password as parameters to offer customers choice. Windows images should have username/password.

- ☐ When specifying a VM image (Windows or Linux), application templates should reference “latest” as the version of the image to keep the template current with the latest patched platform images.
- ☐ In the template being deployed, the latest two versions of the API should be used and the same API version should be used consistently across the template.
- ☐ All parameters should include allowedValues, to guide customers to select the right options before deploying the template. Once the template is launched, it may take a large amount of time to return failures back to customers.
- ☐ Supplying information in the Output section will help customers debug issues as part of their deployment of your application.
- ☐ It is strongly recommended that the application template finishes in less than 2 hours. Wanting to ensure consistency in deployment times and not running up against the 2 hour limit, we recommend template deployments not taking longer than 90 minutes per step.
- ☐ To ensure a reasonable customer experience, we also recommend deployments take less than 2 hours in the majority of cases. For exceptional large deployments, we can work with you to understand what the right value should be.

 **Optional**—this is optional but important, and it is called out for reference.

- ☐ End customers prefer to have as few parameters as possible to make customer deployments as easy and fast to launch as possible.
- ☐ Using defaults for template parameters make it significantly easier for end-customers to deploy quickly and with the correct options for optimal experience. It is recommended that applications use as many defaults as possible for parameters.
- ☐ Metadata on the parameters is helpful hints for customers when they deploy your application programmatically.

Appendix B: Links to Useful Articles

VM Images

- [About Virtual Machine Images in Azure](#)
- [A Set of Blog Posts by Christine Avanesians](#)

VM Extensions

- [VM Agent and VM Extensions Overview](#)
- [Azure VM Extensions and Features](#)
- [A Set of Blog Posts by Kundana Palagiri](#)

ARM

- [Authoring Azure ARM Templates](#)
- [Simple ARM Template Examples](#)

Storage Account Throttles

- [How to Monitor for Storage Account Throttling](#)
- [Premium Storage](#)

Compute on ARM

- [Azure Compute, Network & Storage Providers under the Azure Resource Manager](#)

Appendix C: Future Updates

This is a 'Best Practices' document. As such, we are continually identifying new guidance and recommendations. The following is a running-list of future information and guidance. This content is for internal use only and not to be included as part of the public version.

- Performance and Stress requirement and guidelines
- Guidance around collaboration and validation
- (source control and template validating techniques [CLI\PS\UI])
- Common challenges
- Common well known errors (troubleshooting)