Contents

Azure managed applications overview

1/2/2020 • 4 minutes to read • Edit Online

Azure managed applications enable you to offer cloud solutions that are easy for consumers to deploy and operate. You implement the infrastructure and provide ongoing support. To make a managed application available to all customers, publish it in the Azure marketplace. To make it available to only users in your organization, publish it to an internal catalog.

A managed application is similar to a solution template in the Marketplace, with one key difference. In a managed application, the resources are deployed to a resource group that's managed by the publisher of the app. The resource group is present in the consumer's subscription, but an identity in the publisher's tenant has access to the resource group. As the publisher, you specify the cost for ongoing support of the solution.

> **NOTE**
>
> Formerly, the documentation for Azure Customer Providers was included with the documentation for Managed Applications. That documentation has been moved. Now, see Azure Custom Providers.

Advantages of managed applications

Managed applications reduce barriers to consumers using your solutions. They don't need expertise in cloud infrastructure to use your solution. Consumers have limited access to the critical resources, don't need to worry about making a mistake when managing it.

Managed applications enable you to establish an ongoing relationship with your consumers. You define terms for managing the application, and all charges are handled through Azure billing.

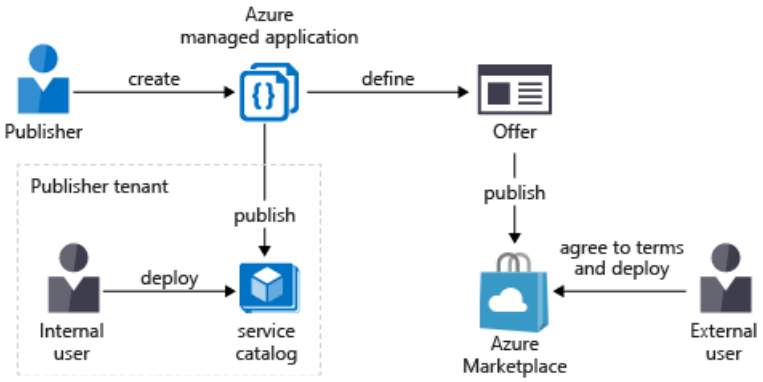
Although customers deploy these managed applications in their subscriptions, they don't have to maintain, update, or service them. You can make sure that all customers are using approved versions. Customers don't have to develop application-specific domain knowledge to manage these applications. Customers automatically acquire application updates without the need to worry about troubleshooting and diagnosing issues with the applications.

For IT teams, managed applications enable you to offer pre-approved solutions to users in the organization. You know these solutions are compliant with organizational standards.

Managed Applications support managed identities for Azure resources.

Types of managed applications

You can publish your managed application either externally or internally.



Service catalog

The service catalog is an internal catalog of approved solutions for users in an organization. You use the catalog to meet organizational standards while they offering solutions for the organizations. Employees use the catalog to easily find applications that are recommended and approved by their IT departments. They see the managed applications that other people in their organization share with them.

For information about publishing a Service Catalog managed application, see Create service catalog application.

Marketplace

Vendors wishing to bill for their services can make a managed application available through the Azure marketplace. After the vendor publishes an application, it's available to users outside the organization. With this approach, managed service providers (MSPs), independent software vendors (ISVs), and system integrators (SIs) can offer their solutions to all Azure customers.

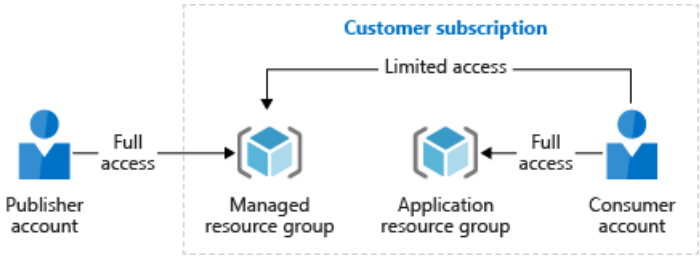
For information about publishing a managed application to the Marketplace, see Create marketplace application.

Resource groups for managed applications

Typically, the resources for a managed application are in two resource groups. The consumer manages one resource group, and the publisher manages the other resource group. When defining the managed application, the publisher specifies the levels of access. The publisher can request either a permanent role assignment, or just-in-time access for an assignment that is constrained to a time period.

Restricting access for data operations is currently not supported for all data providers in Azure.

The following image shows a scenario where the publisher requests the owner role for the managed resource group. The publisher placed a read-only lock on this resource group for the consumer. The publisher's identities that are granted access to the managed resource group are exempt from the lock.



Application resource group

This resource group holds the managed application instance. This resource group may only contain one resource. The resource type of the managed application is **Microsoft.Solutions/applications**.

The consumer has full access to the resource group and uses it to manage the lifecycle of the managed application.

Managed resource group

This resource group holds all the resources that are required by the managed application. For example, this resource group contains the virtual machines, storage accounts, and virtual networks for the solution. The consumer has limited access to this resource group because the consumer doesn't manage the individual resources for the managed application. The publisher's access to this resource group corresponds to the role specified in the managed application definition. For example, the publisher might request the Owner or Contributor role for this resource group. The access is either permanent or limited to a specific time.

When publishing the managed application to the marketplace, the publisher can grant consumers the ability to perform specific actions on resources in the managed resource group. For example, the publisher can specify that consumers can restart virtual machines. All other actions beyond read actions are still denied.

When the consumer deletes the managed application, the managed resource group is also deleted.

Azure Policy

You can apply an Azure Policy to your managed application. You apply policies to make sure deployed instances of your managed application fulfill data and security requirements. If your application interacts with sensitive data, make sure you've evaluated how that should be protected. For example, if your application interacts with data from Office 365, apply a policy to make sure data encryption is enabled.

Next steps

In this article, you learned about benefits of using managed applications. Go to the next article to create a managed application definition.

Quickstart: Publish an Azure managed application definition

Publish an Azure managed application definition

1/2/2020 • 3 minutes to read • Edit Online

This quickstart provides an introduction to working with managed applications. You add a managed application definition to an internal catalog for users in your organization. To simplify the introduction, we have already built the files for your managed application. Those files are available through GitHub. You learn how to build those files in the Create service catalog application tutorial.

When you're finished, you have a resource group named **appDefinitionGroup** that has the managed application definition.

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select **Try It** in the upper-right corner of a code block. Selecting **Try It** doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com, or select the **Launch Cloud Shell** button to open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu bar at the upper right in the Azure portal.	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl**+**Shift**+**V** on Windows and Linux or by selecting **Cmd**+**Shift**+**V** on macOS.
4. Select **Enter** to run the code.

Create a resource group for definition

Your managed application definition exists in a resource group. The resource group is a logical collection into which Azure resources are deployed and managed.

To create a resource group, use the following command:

```
az group create --name appDefinitionGroup --location westcentralus
```

Create the managed application definition

When defining the managed application, you select a user, group, or application that manages the resources for the consumer. This identity has permissions on the managed resource group according to the role that is assigned. Typically, you create an Azure Active Directory group to manage the resources. However, for this article, use your own identity.

To get the object ID of your identity, provide your user principal name in the following command:

```
userid=$(az ad user show --id example@contoso.org --query objectId --output tsv)
```

Next, you need the role definition ID of the RBAC built-in role you want to grant access to the user. The following command shows how to get the role definition ID for the Owner role:

```
roleid=$(az role definition list --name Owner --query [].name --output tsv)
```

Now, create the managed application definition resource. The managed application contains only a storage account.

```
az managedapp definition create \
  --name "ManagedStorage" \
  --location "westcentralus" \
  --resource-group appDefinitionGroup \
  --lock-level ReadOnly \
  --display-name "Managed Storage Account" \
  --description "Managed Azure Storage Account" \
  --authorizations "$userid:$roleid" \
  --package-file-uri "https://github.com/Azure/azure-managedapp-
samples/raw/master/Managed%20Application%20Sample%20Packages/201-managed-storage-account/managedstorage.zip"
```

When the command completes, you have a managed application definition in your resource group.

Some of the parameters used in the preceding example are:

- **resource-group**: The name of the resource group where the managed application definition is created.
- **lock-level**: The type of lock placed on the managed resource group. It prevents the customer from performing undesirable operations on this resource group. Currently, ReadOnly is the only supported lock level. When ReadOnly is specified, the customer can only read the resources present in the managed resource group. The publisher identities that are granted access to the managed resource group are exempt from the lock.
- **authorizations**: Describes the principal ID and the role definition ID that are used to grant permission to the managed resource group. It's specified in the format of `<principalId>:<roleDefinitionId>`. If more than one value is needed, specify them in the form `<principalId1>:<roleDefinitionId1> <principalId2>:<roleDefinitionId2>`. The values are separated by a space.
- **package-file-uri**: The location of a .zip package that contains the required files. The package must have the **mainTemplate.json** and **createUiDefinition.json** files. **mainTemplate.json** defines the Azure resources that are created as part of the managed application. The template is no different than a regular Resource Manager template. **createUiDefinition.json** generates the user interface for users who create the managed application through the portal.

Next steps

You've published the managed application definition. Now, learn how to deploy an instance of that definition.

Quickstart: Deploy service catalog app

Deploy service catalog app through Azure portal

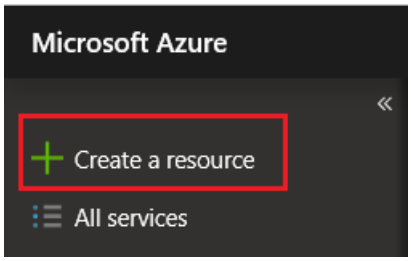
1/2/2020 • 2 minutes to read • Edit Online

In the preceding quickstart, you published a managed application definition. In this quickstart, you create a service catalog app from that definition.

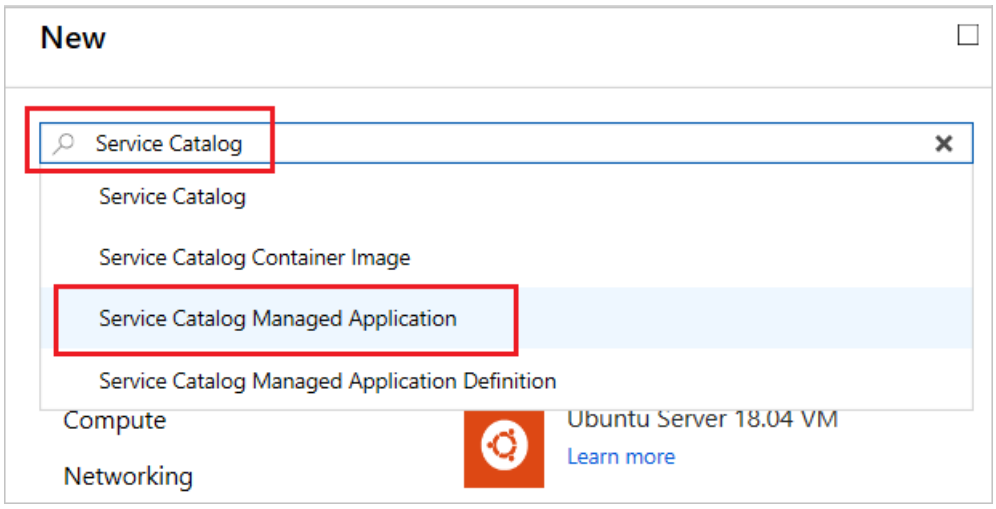
Create service catalog app

In the Azure portal, use the following steps:

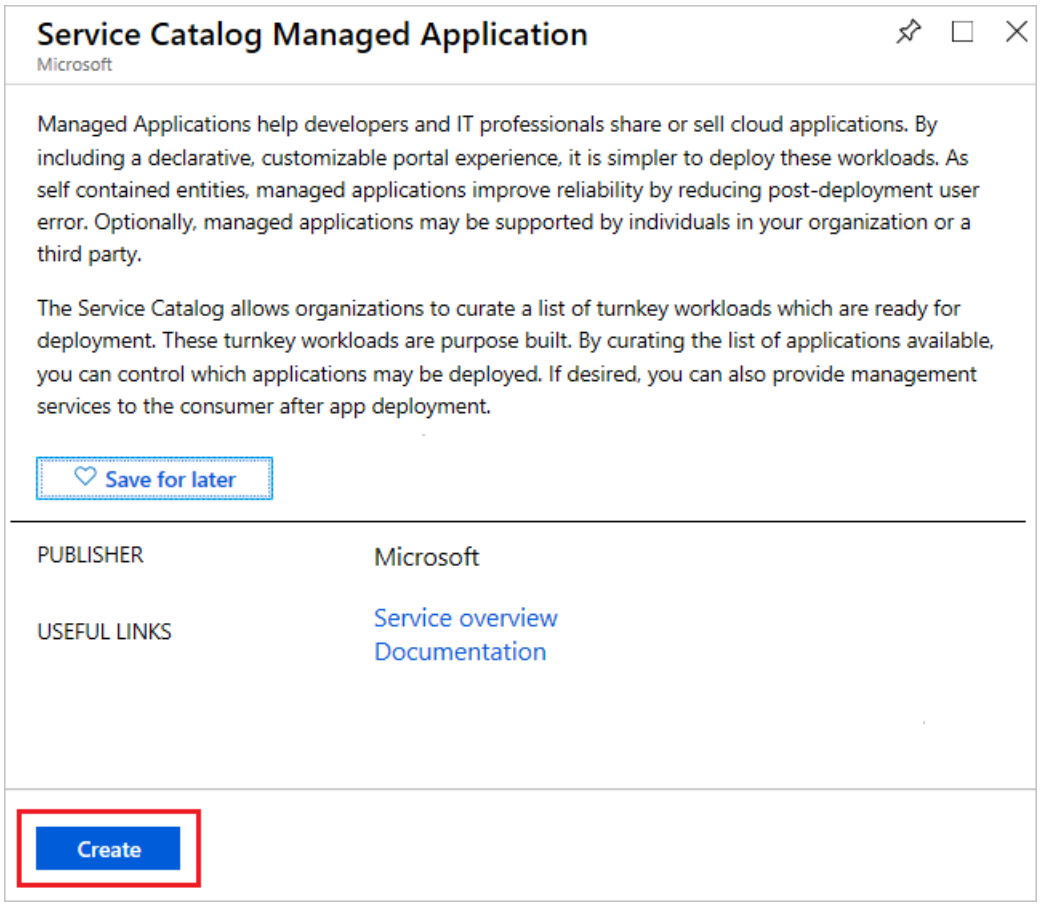
1. Select **Create a resource**.



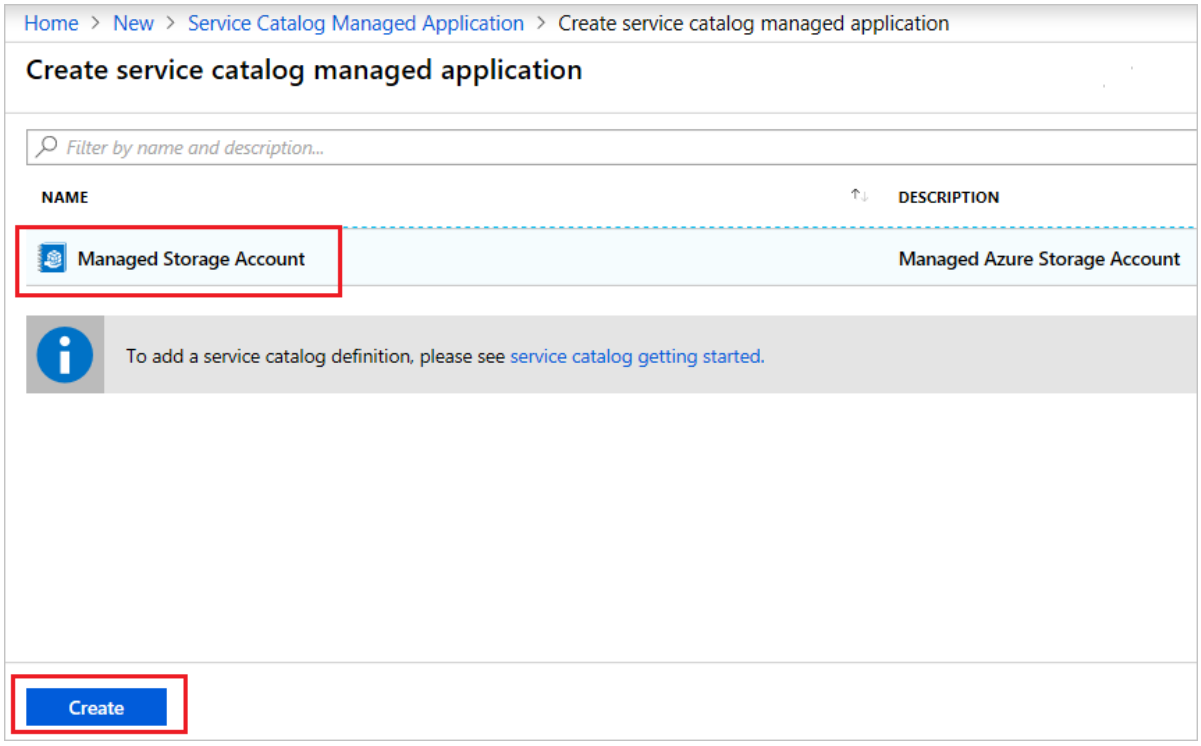
2. Search for **Service Catalog Managed Application** and select it from the available options.



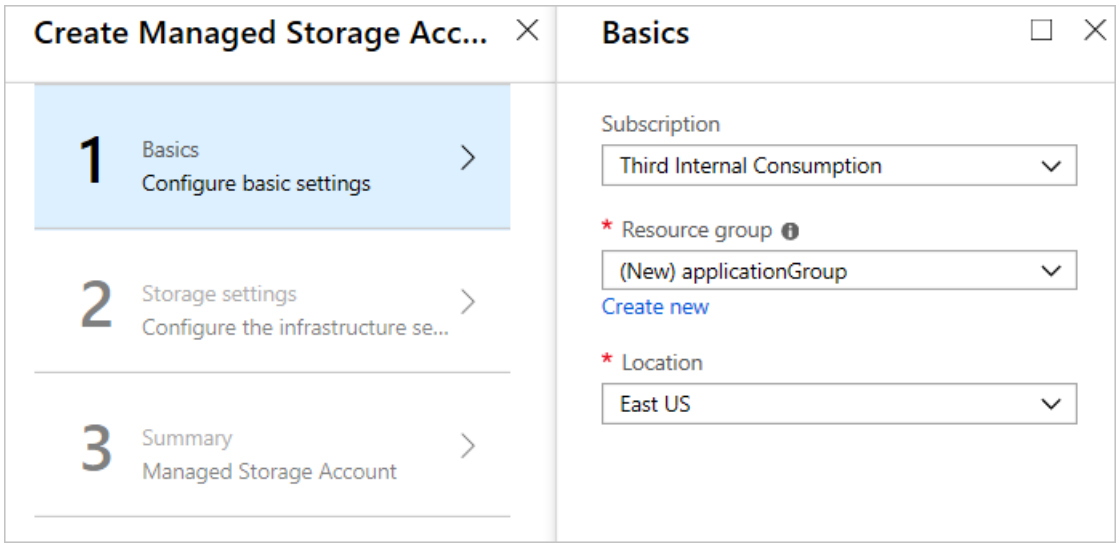
3. You see a description of the Managed Application service. Select **Create**.



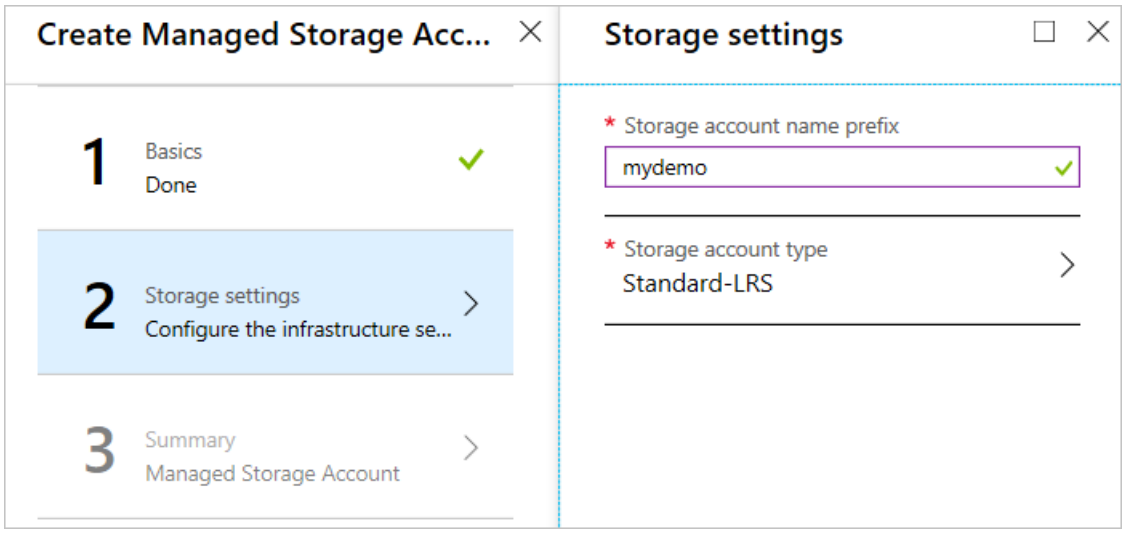
4. The portal shows the managed application definitions that you have access to. From the available definitions, select the one you wish to deploy. In this quickstart, use the **Managed Storage Account** definition that you created in the preceding quickstart. Select **Create**.



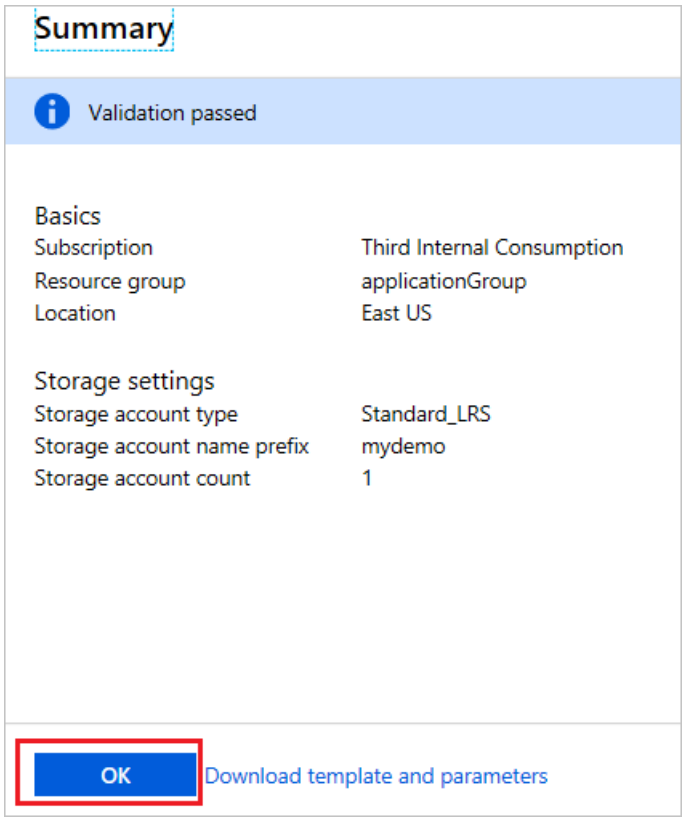
5. Provide values for the **Basics** tab. Select the Azure subscription to deploy your service catalog app to. Create a new resource group named **applicationGroup**. Select a location for your app. When finished, select **OK**.



6. Provide a prefix for the storage account name. Select the type of storage account to create. When finished, select **OK**.



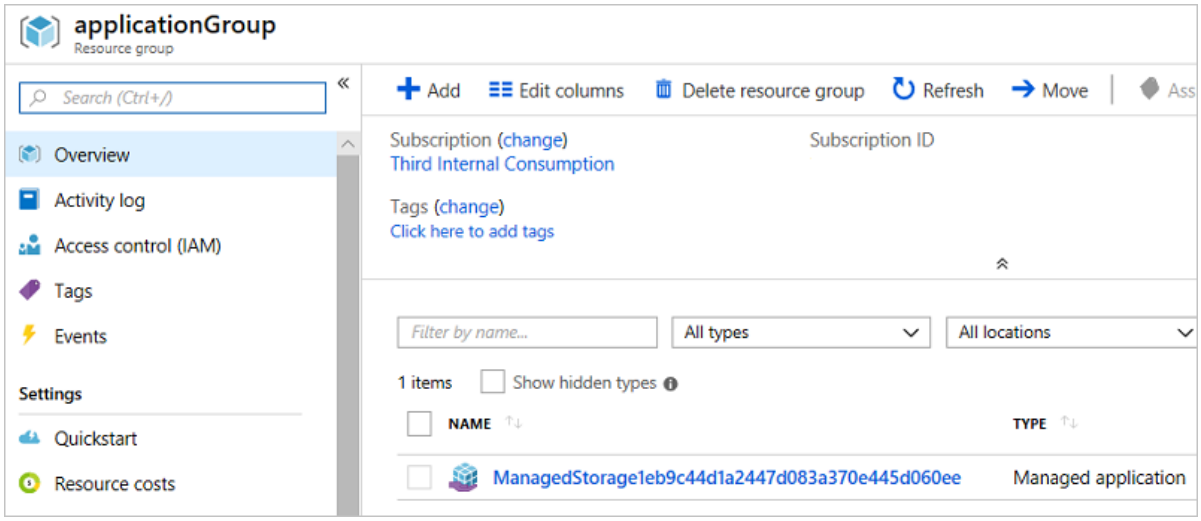
7. Review the summary. After validation succeeds, select **OK** to begin deployment.



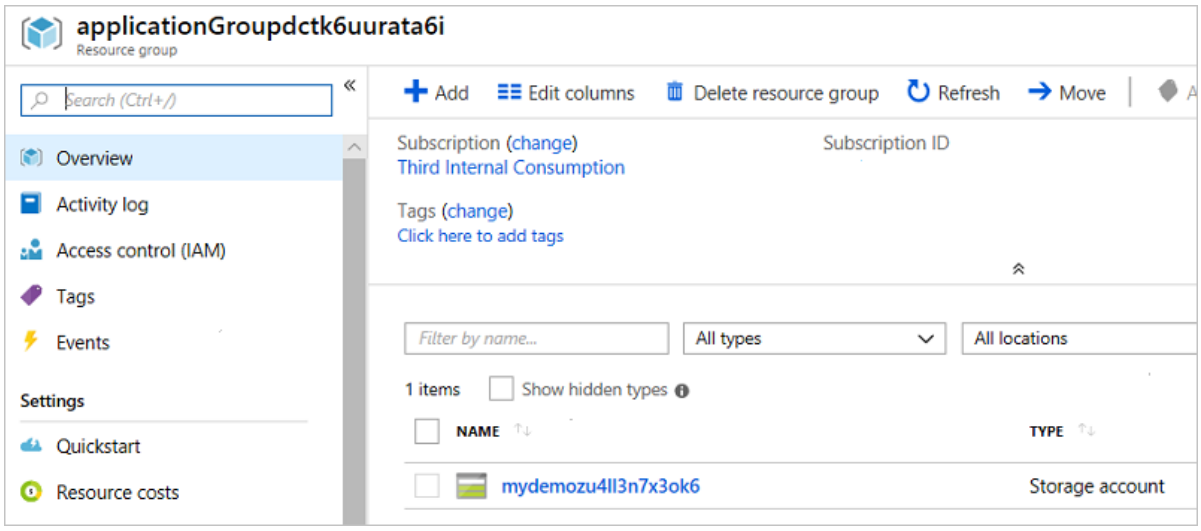
View results

After the service catalog app has been deployed, you have two new resource groups. One resource group holds the service catalog app. The other resource group holds the resources for the service catalog app.

1. View the resource group named **applicationGroup** to see the service catalog app.



2. View the resource group named **applicationGroup{hash-characters}** to see the resources for the service catalog app.



Next steps

- To learn how to create the definition files for a managed application, see Create and publish a managed application definition.
- For Azure CLI, see Deploy service catalog app with Azure CLI.
- For PowerShell, see Deploy service catalog app with PowerShell.

Create and publish a managed application definition

1/11/2020 • 7 minutes to read • Edit Online

> **NOTE**
>
> This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see Introducing the new Azure PowerShell Az module. For Az module installation instructions, see Install Azure PowerShell.

You can create and publish Azure managed applications that are intended for members of your organization. For example, an IT department can publish managed applications that fulfill organizational standards. These managed applications are available through the service catalog, not the Azure marketplace.

To publish a managed application to your Azure Service Catalog, you must:

- Create a template that defines the resources to deploy with the managed application.
- Define the user interface elements for the portal when deploying the managed application.
- Create a .zip package that contains the required template files.
- Decide which user, group, or application needs access to the resource group in the user's subscription.
- Create the managed application definition that points to the .zip package and requests access for the identity.

For this article, your managed application has only a storage account. It's intended to illustrate the steps of publishing a managed application. For complete examples, see Sample projects for Azure managed applications.

The PowerShell examples in this article require Azure PowerShell 6.2 or later. If needed, update your version.

Create the resource template

Every managed application definition includes a file named **mainTemplate.json**. In it, you define the Azure resources to deploy. The template is no different than a regular Resource Manager template.

Create a file named **mainTemplate.json**. The name is case-sensitive.

Add the following JSON to your file. It defines the parameters for creating a storage account, and specifies the properties for the storage account.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "storageAccountNamePrefix": {
            "type": "string"
        },
        "storageAccountType": {
            "type": "string"
        },
        "location": {
            "type": "string",
            "defaultValue": "[resourceGroup().location]"
        }
    },
    "variables": {
        "storageAccountName": "[concat(parameters('storageAccountNamePrefix'),
uniqueString(resourceGroup().id))]"
    },
    "resources": [
        {
            "type": "Microsoft.Storage/storageAccounts",
            "name": "[variables('storageAccountName')]",
            "apiVersion": "2016-01-01",
            "location": "[parameters('location')]",
            "sku": {
                "name": "[parameters('storageAccountType')]"
            },
            "kind": "Storage",
            "properties": {}
        }
    ],
    "outputs": {
        "storageEndpoint": {
            "type": "string",
            "value": "[reference(resourceId('Microsoft.Storage/storageAccounts/',
variables('storageAccountName')), '2016-01-01').primaryEndpoints.blob]"
        }
    }
}
```

Save the mainTemplate.json file.

Defining your create experience using CreateUiDefinition.json

As a publisher, you define your create experience using the **createUiDefinition.json** file which generates the interface for users creating managed applications. You define how users provide input for each parameter using control elements including drop-downs, text boxes, and password boxes.

Create a file named **createUiDefinition.json** (This name is case-sensitive)

Add the following starter JSON to the file and save it.

```
{
   "$schema": "https://schema.management.azure.com/schemas/0.1.2-preview/CreateUIDefinition.MultiVm.json#",
   "handler": "Microsoft.Azure.CreateUIDef",
   "version": "0.1.2-preview",
   "parameters": {
        "basics": [
            {}
        ],
        "steps": [
            {
                "name": "storageConfig",
                "label": "Storage settings",
                "subLabel": {
                    "preValidation": "Configure the infrastructure settings",
                    "postValidation": "Done"
                },
                "bladeTitle": "Storage settings",
                "elements": [
                    {
                        "name": "storageAccounts",
                        "type": "Microsoft.Storage.MultiStorageAccountCombo",
                        "label": {
                            "prefix": "Storage account name prefix",
                            "type": "Storage account type"
                        },
                        "defaultValue": {
                            "type": "Standard_LRS"
                        },
                        "constraints": {
                            "allowedTypes": [
                                "Premium_LRS",
                                "Standard_LRS",
                                "Standard_GRS"
                            ]
                        }
                    }
                ]
            }
        ],
        "outputs": {
            "storageAccountNamePrefix": "[steps('storageConfig').storageAccounts.prefix]",
            "storageAccountType": "[steps('storageConfig').storageAccounts.type]",
            "location": "[location()]"
        }
    }
}
```

To learn more, see Get started with CreateUiDefinition.

Package the files

Add the two files to a .zip file named app.zip. The two files must be at the root level of the .zip file. If you put them in a folder, you receive an error when creating the managed application definition that states the required files aren't present.

Upload the package to an accessible location from where it can be consumed.

```
New-AzResourceGroup -Name storageGroup -Location eastus
$storageAccount = New-AzStorageAccount -ResourceGroupName storageGroup `
  -Name "mystorageaccount" `
  -Location eastus `
  -SkuName Standard_LRS `
  -Kind Storage

$ctx = $storageAccount.Context

New-AzStorageContainer -Name appcontainer -Context $ctx -Permission blob

Set-AzStorageBlobContent -File "D:\myapplications\app.zip" `
  -Container appcontainer `
  -Blob "app.zip" `
  -Context $ctx
```

Create the managed application definition

Create an Azure Active Directory user group or application

The next step is to select a user group or application for managing the resources on behalf of the customer. This user group or application has permissions on the managed resource group according to the role that is assigned. The role can be any built-in Role-Based Access Control (RBAC) role like Owner or Contributor. You also can give an individual user permission to manage the resources, but typically you assign this permission to a user group. To create a new Active Directory user group, see Create a group and add members in Azure Active Directory.

You need the object ID of the user group to use for managing the resources.

```
$groupID=(Get-AzADGroup -DisplayName mygroup).Id
```

Get the role definition ID

Next, you need the role definition ID of the RBAC built-in role you want to grant access to the user, user group, or application. Typically, you use the Owner or Contributor or Reader role. The following command shows how to get the role definition ID for the Owner role:

```
$ownerID=(Get-AzRoleDefinition -Name Owner).Id
```

Create the managed application definition

If you don't already have a resource group for storing your managed application definition, create one now:

```
New-AzResourceGroup -Name appDefinitionGroup -Location westcentralus
```

Now, create the managed application definition resource.

```
$blob = Get-AzStorageBlob -Container appcontainer -Blob app.zip -Context $ctx

New-AzManagedApplicationDefinition `
  -Name "ManagedStorage" `
  -Location "westcentralus" `
  -ResourceGroupName appDefinitionGroup `
  -LockLevel ReadOnly `
  -DisplayName "Managed Storage Account" `
  -Description "Managed Azure Storage Account" `
  -Authorization "${groupID}:$ownerID" `
  -PackageFileUri $blob.ICloudBlob.StorageUri.PrimaryUri.AbsoluteUri
```

Bring your own storage for the managed application definition

You can choose to store your managed application definition within a storage account provided by you during creation so that it's location and access can be fully managed by you for your regulatory needs.

> **NOTE**
>
> Bring your own storage is only supported with ARM Template or REST API deployments of the managed application definition.

Select your storage account

You must create a storage account to contain your managed application definition for use with Service Catalog.

Copy the storage account's resource ID. It will be used later when deploying the definition.

Set the role assignment for "Appliance Resource Provider" in your storage account

Before your managed application definition can be deployed to your storage account, you must give contributor permissions to the **Appliance Resource Provider** role so that it can write the definition files to your storage account's container.

1. In the Azure portal, navigate to your storage account.
2. Select **Access control (IAM)** to display the access control settings for the storage account. Select the **Role assignments** tab to see the list of role assignments.
3. In the **Add role assignment** window, select the **Contributor** role.
4. From the **Assign access to** field, select **Azure AD user, group, or service principal**.
5. Under **Select** search for **Appliance Resource Provider** role and select it.
6. Save the role assignment.

Deploy the managed application definition with an ARM Template

Use the following ARM Template to deploy your packaged managed application as a new managed application definition in Service Catalog whose definition files are stored and maintained in your own storage account:

```
{
"$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
"contentVersion": "1.0.0.0",
"parameters": {
    "location": {
        "type": "string",
        "defaultValue": "[resourceGroup().location]"
    },
    "applicationName": {
        "type": "string",
        "metadata": {
            "description": "Managed Application name"
        }
    },
    "storageAccountType": {
  "type": "string",
  "defaultValue": "Standard_LRS",
  "allowedValues": [
    "Standard_LRS",
    "Standard_GRS",
    "Standard_ZRS",
    "Premium_LRS"
  ],
  "metadata": {
    "description": "Storage Account type"
  }
},
```

```
        "definitionStorageResourceID": {
            "type": "string",
            "metadata": {
                "description": "Storage account resource ID for where you're storing your definition"
            }
        },
        "_artifactsLocation": {
            "type": "string",
            "metadata": {
                "description": "The base URI where artifacts required by this template are located."
            }
        }
    },
    "variables": {
        "lockLevel": "None",
        "description": "Sample Managed application definition",
        "displayName": "Sample Managed application definition",
        "managedApplicationDefinitionName": "[parameters('applicationName')]",
        "packageFileUri": "[parameters('_artifactsLocation')]",
        "defLocation": "[parameters('definitionStorageResourceID')]",
        "managedResourceGroupId": "[concat(subscription().id,'/resourceGroups/',
concat(parameters('applicationName'),'_managed'))]",
        "applicationDefinitionResourceId": "
[resourceId('Microsoft.Solutions/applicationDefinitions',variables('managedApplicationDefinitionName'))]"
    },
    "resources": [
        {
            "type": "Microsoft.Solutions/applicationDefinitions",
            "apiVersion": "2019-07-01",
            "name": "[variables('managedApplicationDefinitionName')]",
            "location": "[parameters('location')]",
            "properties": {
                "lockLevel": "[variables('lockLevel')]",
                "description": "[variables('description')]",
                "displayName": "[variables('displayName')]",
                "packageFileUri": "[variables('packageFileUri')]",
                "storageAccountId": "[variables('defLocation')]"
            }
        }
    ],
    "outputs": {}
}
```

We have added a new property named **storageAccountId** to your applicationDefintion's properties and provide storage account id you wish to store your definition in as its value:

You can verify that the application definition files are saved in your provided storage account in a container titled **applicationdefinitions**.

> **NOTE**
>
> For added security, you can create a managed applications definition store it in an Azure storage account blob where encryption is enabled. The definition contents are encrypted through the storage account's encryption options. Only users with permissions to the file can see the definition in Service Catalog.

Make sure users can see your definition

You have access to the managed application definition, but you want to make sure other users in your organization can access it. Grant them at least the Reader role on the definition. They may have inherited this level of access from the subscription or resource group. To check who has access to the definition and add users or groups, see Use Role-Based Access Control to manage access to your Azure subscription resources.

Next steps

- To publish your managed application to the Azure Marketplace, see Azure managed applications in the Marketplace.
- To deploy a managed application instance, see Deploy service catalog app through Azure portal.

Azure managed applications in the Marketplace

1/2/2020 • 8 minutes to read • Edit Online

Vendors can use Azure managed applications to offer their solutions to all Azure Marketplace customers. Those vendors can include managed service providers (MSPs), independent software vendors (ISVs), and system integrators (SIs). Managed applications reduce the maintenance and servicing overhead for customers. Vendors sell infrastructure and software through the marketplace. They can attach services and operational support to managed applications. For more information, see Managed application overview.

This article explains how you can publish an application to the marketplace and make it broadly available to customers.

Prerequisites for publishing a managed application

To complete this article, you must already have the .zip file for your managed application definition. For more information, see Create service catalog application.

There are several business prerequisites. They are:

- Your company or its subsidiary must be located in a country/region where sales are supported by the marketplace.
- Your product must be licensed in a way that is compatible with billing models supported by the marketplace.
- Make technical support available to customers in a commercially reasonable manner. The support can be free, paid, or through community support.
- License your software and any third-party software dependencies.
- Provide content that meets criteria for your offering to be listed in the Marketplace and in the Azure portal.
- Agree to the terms of the Azure Marketplace Participation Policies and Publisher Agreement.
- Agree to comply with the Terms of Use, Microsoft Privacy Statement, and Microsoft Azure Certified Program Agreement.

You must also have a Marketplace account. To create an account, see How to create a Commercial Marketplace account in Partner Center.

Create a new Azure application offer

After creating your partner portal account, you're ready to create your managed application offer.

Set up an offer

The offer for a managed application corresponds to a class of product offering from a publisher. If you have a new type of application that you want to make available in the marketplace, you can set it up as a new offer. An offer is a collection of SKUs. Every offer appears as its own entity in the marketplace.

1. Sign in to the Cloud Partner portal.

2. In the navigation pane on the left, select **+ New offer** > **Azure Applications**.

3. In the **Editor** view, you see the required forms. Each form is described later in this article.

Offer Settings form

The fields for the **Offer Settings** form are:

- **Offer ID**: This unique identifier identifies the offer within a publisher profile. This ID is visible in product URLs, Resource Manager templates, and billing reports. It can only be composed of lowercase alphanumeric characters or dashes (-). The ID can't end in a dash. It's limited to a maximum of 50 characters. After an offer goes live, this field is locked.
- **Publisher ID**: Use this drop-down list to choose the publisher profile you want to publish this offer under. After an offer goes live, this field is locked.
- **Name**: This display name for your offer appears in the Marketplace and in the portal. It can have a maximum of 50 characters. Include a recognizable brand name for your product. Don't include your company name here unless that's how it's marketed. If you're marketing this offer on your own website, ensure that the name is exactly how it appears on your website.

When done, select **Save** to save your progress.

SKUs form

The next step is to add SKUs for your offer.

A SKU is the smallest purchasable unit of an offer. You can use a SKU within the same product class (offer) to differentiate between:

- Different features that are supported
- Whether the offer is managed or unmanaged
- Billing models that are supported

A SKU appears under the parent offer in the marketplace. It appears as its own purchasable entity in the Azure portal.

1. Select **SKUs** > **New SKU**.

2. Enter a **SKU ID**. A SKU ID is a unique identifier for the SKU within an offer. This ID is visible in product URLs, Resource Manager templates, and billing reports. It can only be composed of lowercase alphanumeric characters or dashes (-). The ID can't end in a dash, and it's limited to a maximum of 50 characters. After an offer goes live, this field is locked. You can have multiple SKUs within an offer. You need a SKU for each image you plan to publish.

3. Fill out the **SKU Details** section on the following form:

   Fill out the following fields:

   - **Title**: Enter a title for this SKU. This title appears in the gallery for this item.
   - **Summary**: Enter a short summary for this SKU. This text appears underneath the title.
   - **Description**: Enter a detailed description about the SKU.
   - **SKU Type**: The allowed values are *Managed Application* and *Solution Templates*. For this case, select *Managed Application*.
   - **Country/Region availability**: Select the countries/regions where the managed application is available.
   - **Pricing**: Provide a price for management of the application. Select the available countries/regions before setting the price.
4. Add a new package. Fill out the **Package Details** section on the following form:

   Fill out the following fields:

   - **Version**: Enter a version for the package you upload. It should be in the format `{number}.{number}.{number}{number}`.

   - **Package file (.zip)**: This package contains two required files compressed into a .zip package. One file is a Resource Manager template that defines the resources to deploy for the managed

application. The other file defines the user interface for consumers deploying the managed application through the portal. In the user interface, you specify elements that enable consumers to provide parameter values.

- **Tenant ID**: The tenant ID for the account to get access.
- **Enable JIT Access**: Select **Yes** to enable just-in-time access control for the account. When enabled, you request access to the consumer's account for a specified time period. To require that consumers of your managed application grant your account permanent access, select **No**.
- **Customize allowed customer actions?**: Select **Yes** to specify which actions consumers can perform on the managed resources.
- **Allowed customer actions**: If you select **Yes** for the previous setting, you can specify which actions are permitted to consumers by using deny assignments for Azure resources.

  For available actions, see Azure Resource Manager resource provider operations. For example, to permit consumers to restart virtual machines, add `Microsoft.Compute/virtualMachines/restart/action` to the allowed actions. The `*/read` action is automatically allowed so you don't need to include that setting.
- **PrincipalId**: This property is the Azure Active Directory (Azure AD) identifier of a user, user group, or application that's granted access to the resources in the customer's subscription. The Role Definition describes the permissions.
- **Role Definition**: This property is a list of all the built-in Role-Based Access Control (RBAC) roles provided by Azure AD. You can select the role that's most appropriate to use to manage the resources on behalf of the customer.
- **Policy Settings**: Apply an Azure Policy to your managed application to specify compliance requirements for the deployed solutions. From the available options, select the policies to apply. For **Policy Parameters**, provide a JSON string with the parameter values. For policy definitions and the format of the parameter values, see Azure Policy Samples.

You can add several authorizations. We recommend that you create an AD user group and specify its ID in **PrincipalId**. This way, you can add more users to the user group without the need to update the SKU.

For more information about RBAC, see Get started with RBAC in the Azure portal.

Marketplace form

The Marketplace form asks for fields that show up on the Azure Marketplace and on the Azure portal.

Preview subscription IDs

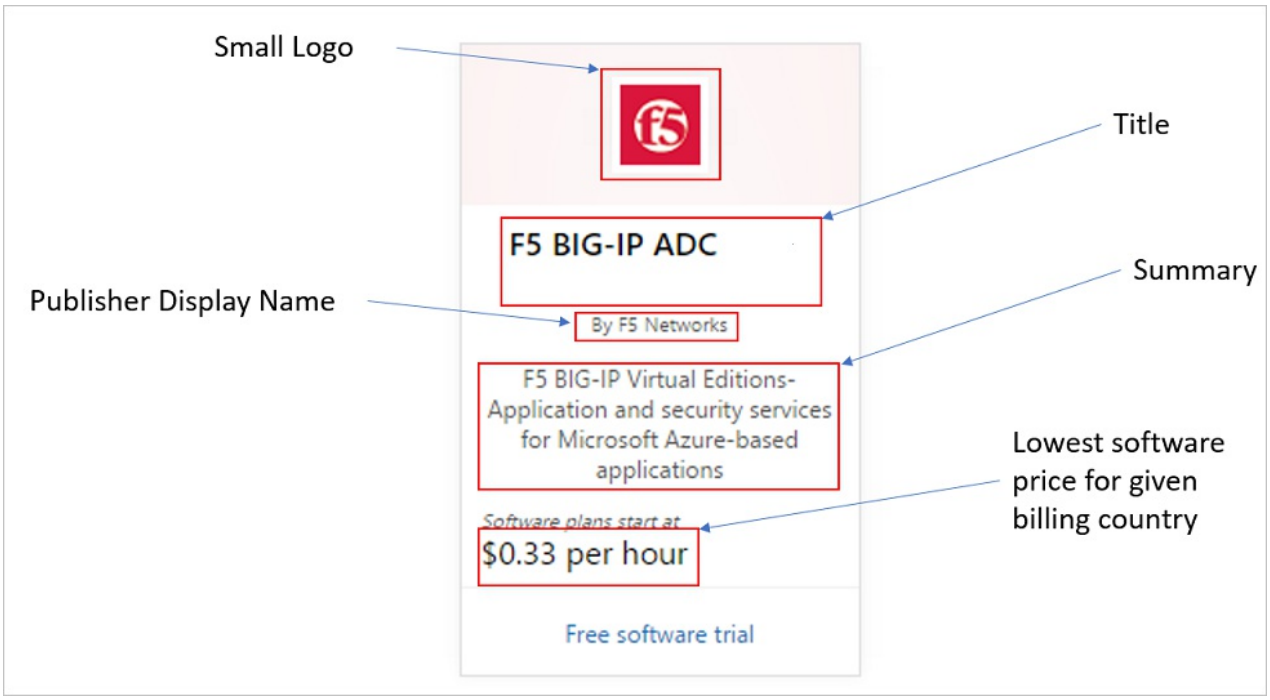
Enter a list of Azure subscription IDs that can access the offer after it's published. You can use these white-listed subscriptions to test the previewed offer before you make it live. You can compile an allow list of up to 100 subscriptions in the partner portal.

Suggested categories

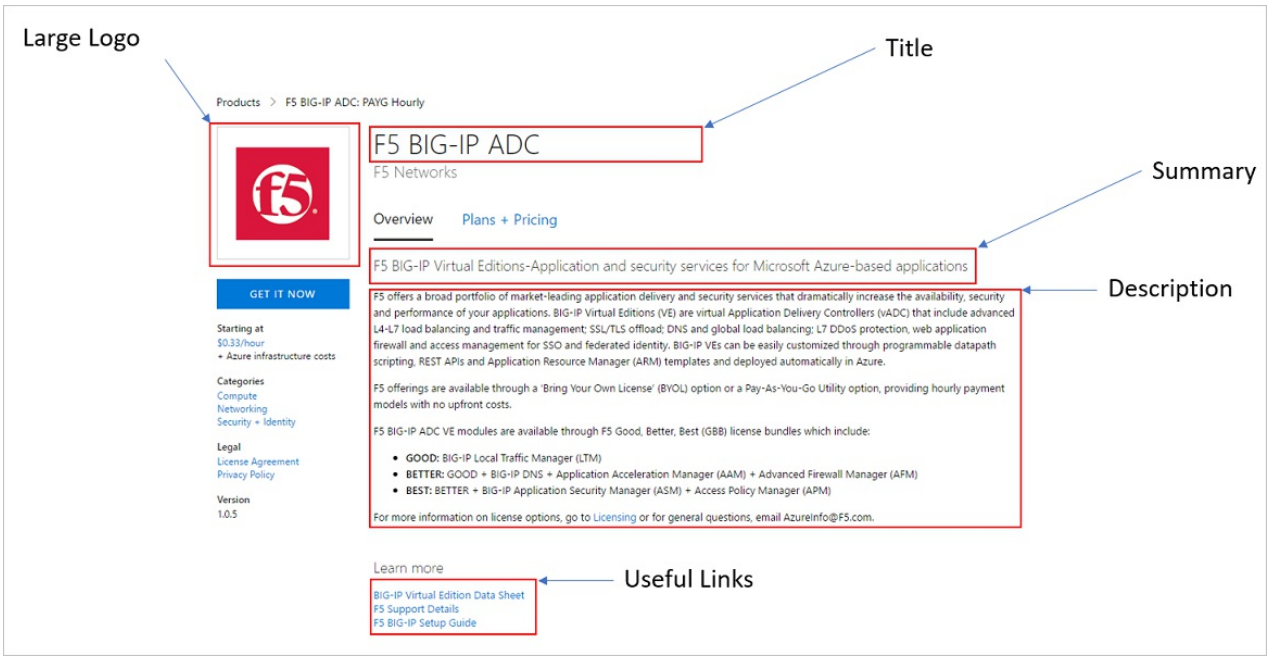
Select up to five categories from the list that your offer can be best associated with. These categories are used to map your offer to the product categories that are available in the Azure Marketplace and the Azure portal.

Azure Marketplace

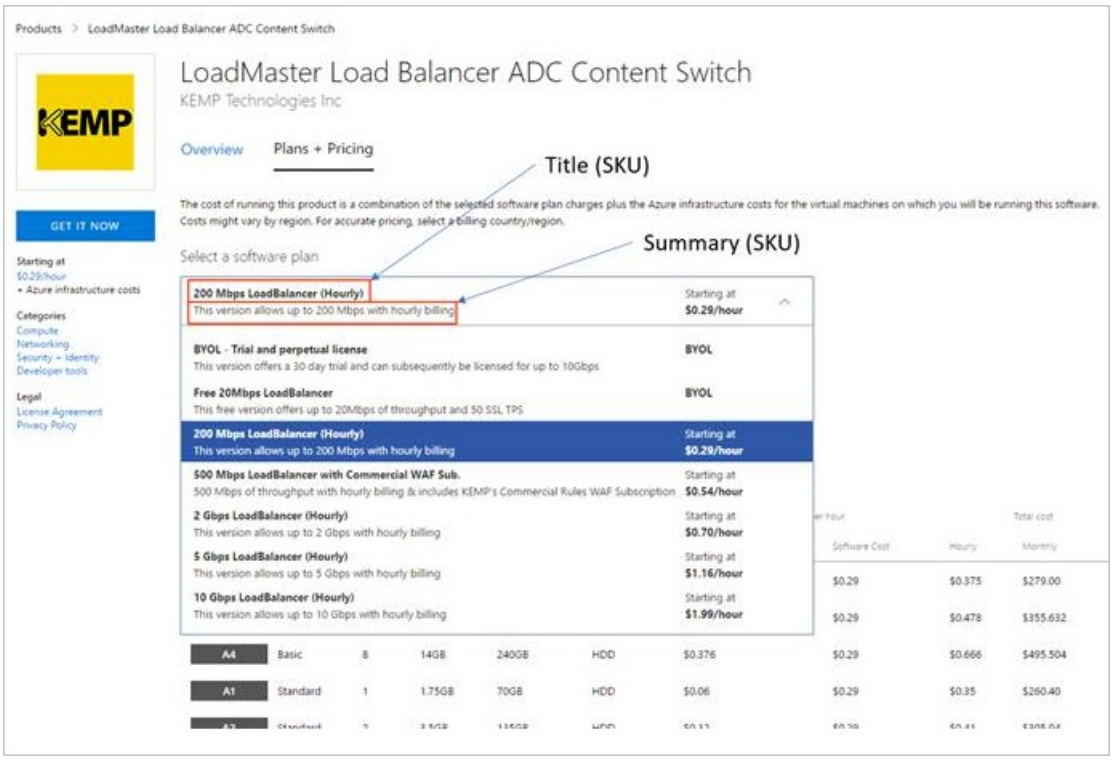
The summary of your managed application displays the following fields:



The **Overview** tab for your managed application displays the following fields:

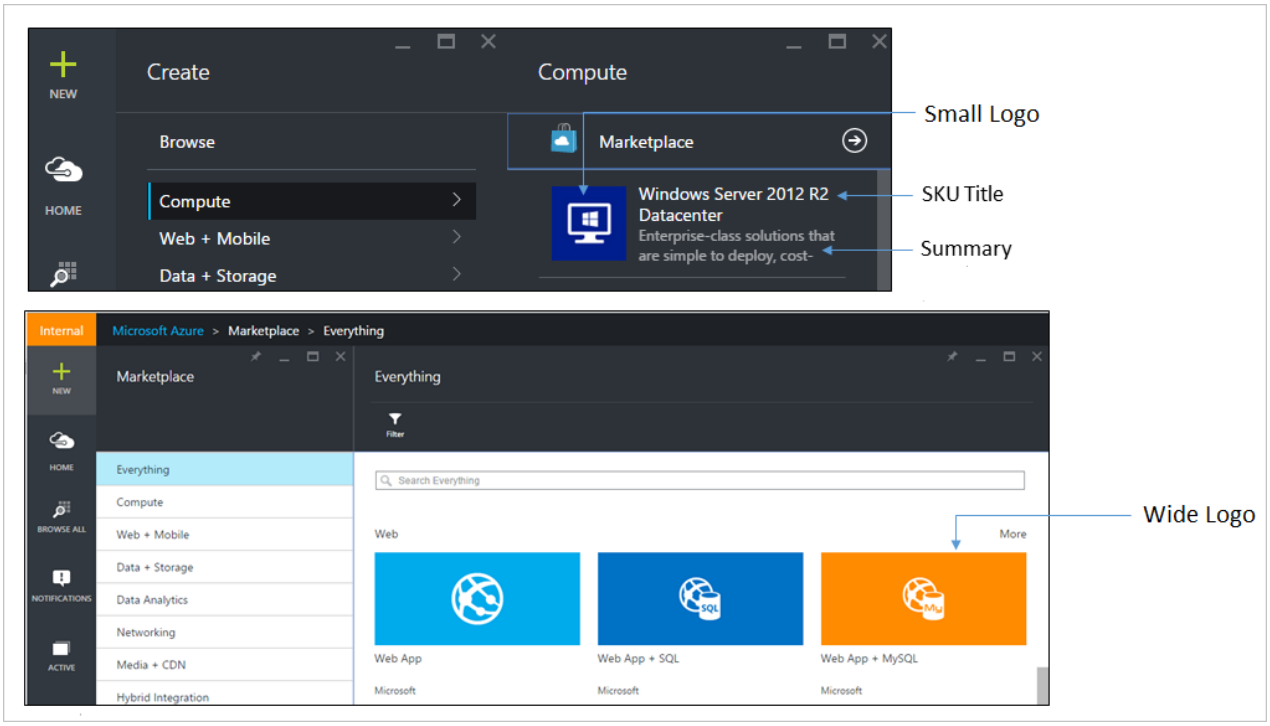


The **Plans + Pricing** tab for your managed application displays the following fields:

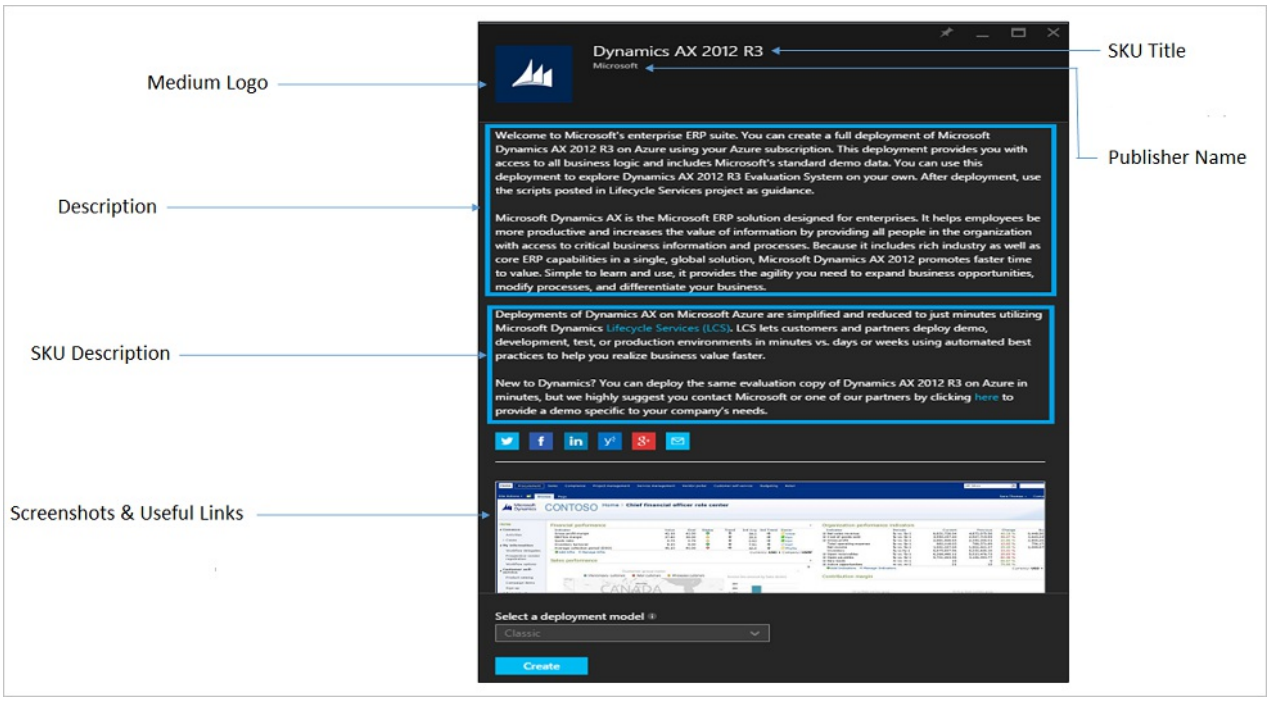


Azure portal

The summary of your managed application displays the following fields:



The overview for your managed application displays the following fields:



Logo guidelines

Follow these guidelines for any logo that you upload in the Cloud Partner portal:

- The Azure design has a simple color palette. Limit the number of primary and secondary colors on your logo.
- The theme colors of the portal are white and black. Don't use these colors as the background color for your logo. Use a color that makes your logo prominent in the portal. We recommend simple primary colors. *If you use a transparent background, make sure that the logo and text aren't white, black, or blue.*
- Don't use a gradient background on the logo.
- Don't place text on the logo, not even your company or brand name. The look and feel of your logo should be flat and avoid gradients.
- Make sure the logo isn't stretched.

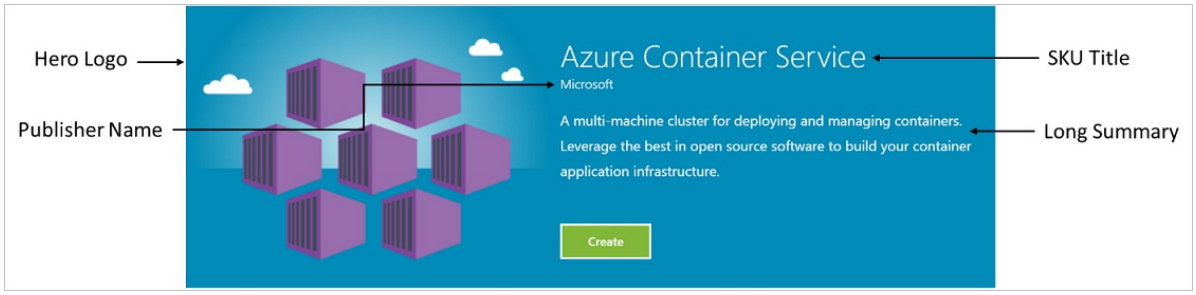
Hero logo

The hero logo is optional. The publisher can choose not to upload a hero logo. After the hero icon is uploaded, it can't be deleted. At that time, the partner must follow the Marketplace guidelines for hero icons.

Follow these guidelines for the hero logo icon:

- The publisher display name, the plan title, and the offer long summary are displayed in white. Therefore, don't use a light color for the background of the hero icon. A black, white, or transparent background isn't allowed for hero icons.

- After the offer is listed, elements are embedded programmatically inside the hero logo. The embedded elements include the publisher display name, the plan title, the offer long summary, and the **Create** button. Consequently, don't enter any text while you design the hero logo. Leave empty space on the right because the text is included programmatically in that space. The empty space for the text should be 415 x 100 pixels on the right. It's offset by 370 pixels from the left.



Support form

Fill out the **Support** form with support contacts from your company. This information might be engineering contacts and customer support contacts.

Publish an offer

After you fill out all the sections, select **Publish** to start the process that makes your offer available to customers.

Next steps

- For information about what happens after you click **Publish**, see Publish Azure application offer
- For an introduction to managed applications, see Managed application overview.
- For information about publishing a Service Catalog managed application, see Create and publish a Service Catalog managed application.

Tutorial: Create managed application with custom actions and resources

1/2/2020 • 8 minutes to read • Edit Online

In this tutorial, you create your own managed application with custom actions and resources. The managed application will contain a custom action on the `Overview` page, a custom resource type displayed as a separate menu item in `Table of Content` and a custom context action on the custom resource page.

This tutorial includes the following steps:

- Author user interface definition file for creating a managed application instance
- Author deployment template with Azure Custom Provider, Azure Storage Account and Azure Function
- Author view definition artifact with custom actions and resources
- Deploy a managed application definition
- Deploy an instance of managed application
- Perform custom actions and create custom resources

Prerequisites

To complete this tutorial, you need to know:

- How to Create and publish a managed application definition.
- How to Deploy Service Catalog app through Azure portal.
- How to Create Azure portal user interface for your managed application.
- View definition artifact capabilities.
- Azure Custom Provider capabilities.

User interface definition

In this tutorial, you create a managed application and its managed resource group will contain custom provider instance, storage account, and function. The Azure Function used in this example implements an API that handles custom provider operations for actions and resources. Azure Storage Account is used as basic storage for your custom provider resources.

The user interface definition for creating a managed application instance includes `funcname` and `storagename` input elements. Storage account name and function name must be globally unique. By default function files will be deployed from sample function package, but you can change it by adding an input element for a package link in *createUIDefinition.json*:

```
{
  "name": "funcname",
  "type": "Microsoft.Common.TextBox",
  "label": "Name of the function to be created",
  "toolTip": "Name of the function to be created",
  "visible": true,
  "constraints": {
    "required": true
  }
},
{
  "name": "storagename",
  "type": "Microsoft.Common.TextBox",
  "label": "Name of the storage to be created",
  "toolTip": "Name of the storage to be created",
  "visible": true,
  "constraints": {
    "required": true
  }
},
{
  "name": "zipFileBlobUri",
  "type": "Microsoft.Common.TextBox",
  "defaultValue": "https://github.com/Azure/azure-quickstart-templates/tree/master/101-custom-rp-with-
function/artifacts/functionzip/functionpackage.zip",
  "label": "The Uri to the uploaded function zip file",
  "toolTip": "The Uri to the uploaded function zip file",
  "visible": true
}
```

and output in *createUIDefinition.json*:

```
  "funcname": "[steps('applicationSettings').funcname]",
  "storageName": "[steps('applicationSettings').storagename]",
  "zipFileBlobUri": "[steps('applicationSettings').zipFileBlobUri]"
```

The complete *createUIDefinition.json* sample can be found at Reference: User interface elements artifacts.

Template with custom provider

To create a managed application instance with custom provider, you need to define custom provider resource with name **public** and type **Microsoft.CustomProviders/resourceProviders** in your **mainTemplate.json**. In that resource, you define the resource types and actions for your service. To deploy Azure Function and Azure Storage Account instances define resources of type `Microsoft.Web/sites` and `Microsoft.Storage/storageAccounts` respectively.

In this tutorial, you will create one `users` resource type, `ping` custom action, and `users/contextAction` custom action that will be performed in a context of a `users` custom resource. For each resource type and action provide an endpoint pointing to the function with name provided in createUIDefinition.json. Specify the **routingType** as `Proxy,Cache` for resource types and `Proxy` for actions:

```
{
  "apiVersion": "[variables('customrpApiversion')]",
  "type": "Microsoft.CustomProviders/resourceProviders",
  "name": "[variables('customProviderName')]",
  "location": "[parameters('location')]",
  "properties": {
    "actions": [
      {
        "name": "ping",
        "routingType": "Proxy",
        "endpoint": "[listSecrets(resourceId('Microsoft.Web/sites/functions', parameters('funcname'),
'HttpTrigger1'), '2018-02-01').trigger_url]"
      },
      {
        "name": "users/contextAction",
        "routingType": "Proxy",
        "endpoint": "[listSecrets(resourceId('Microsoft.Web/sites/functions', parameters('funcname'),
'HttpTrigger1'), '2018-02-01').trigger_url]"
      }
    ],
    "resourceTypes": [
      {
        "name": "users",
        "routingType": "Proxy,Cache",
        "endpoint": "[listSecrets(resourceId('Microsoft.Web/sites/functions', parameters('funcname'),
'HttpTrigger1'), '2018-02-01').trigger_url]"
      }
    ]
  },
  "dependsOn": [
    "[concat('Microsoft.Web/sites/',parameters('funcname'))]"
  ]
}
```

The complete *mainTemplate.json* sample can be found at Reference: Deployment template artifact.

View definition artifact

To define user interface that includes custom actions and custom resources in your managed application, you need to author **viewDefinition.json** artifact. For more information about view definition artifact, see View definition artifact in Azure Managed Applications.

In this tutorial, you define:

- An *Overview* page with toolbar button that represents a custom action `TestAction` with basic text input.
- A *Users* page that represents a custom resource type `users`.
- A custom resource action `users/contextAction` in *Users* page that will be performed in a context of custom resource of type `users`.

The following example shows view configuration for an "Overview" page:

```
{
    "kind": "Overview",
    "properties": {
      "header": "Welcome to your Demo Azure Managed Application",
      "description": "This Managed application with Custom Provider is for demo purposes only.",
      "commands": [{
          "displayName": "Ping Action",
          "path": "/customping",
          "icon": "LaunchCurrent"
      }]
    }
  }
```

The example below includes "Users" resources page configuration with custom resource action:

```
{
    "kind": "CustomResources",
    "properties": {
      "displayName": "Users",
      "version": "1.0.0.0",
      "resourceType": "users",
      "createUIDefinition": {
      },
      "commands": [{
        "displayName": "Custom Context Action",
        "path": "users/contextAction",
        "icon": "Start"
      }],
      "columns": [
        { "key": "properties.FullName", "displayName": "Full Name" },
        { "key": "properties.Location", "displayName": "Location", "optional": true }
      ]
    }
  }
```

The complete *viewDefinition.json* sample can be found at Reference: View definition artifact.

Managed application definition

Package the following managed application artifacts to zip archive and upload it to storage:

- createUiDefinition.json
- mainTemplate.json
- viewDefinition.json

All files must be at root level. The package with artifacts can be stored in any storage, for example GitHub blob or Azure Storage Account blob. Here is a script to upload the application package to storage account:

```
$resourceGroup="appResourcesGroup"
$storageName="mystorageaccount$RANDOM"

# Sign in to your Azure subscription
Connect-AzAccount
# Create resource group for managed application definition and application package
New-AzResourceGroup -Name $resourceGroup -Location eastus

# Create storage account for a package with application artifacts
$storageAccount=New-AzStorageAccount `
  -ResourceGroupName $resourceGroup `
  -Name $storageName `
  -SkuName Standard_LRS `
  -Location eastus `
$ctx=$storageAccount.Context

# Create storage container and upload zip to blob
New-AzStorageContainer -Name appcontainer -Context $ctx -Permission blob
Set-AzStorageBlobContent `
  -File "path_to_your_zip_package" `
  -Container appcontainer `
  -Blob app.zip `
  -Context $ctx

# Get blob absolute uri
$blobUri=(Get-AzureStorageBlob -Container appcontainer -Blob app.zip -Context $ctx).ICloudBlob.uri.AbsoluteUri
```

Run the Azure CLI script below or follow the steps in Azure portal to deploy a Service Catalog managed application definition:

To run this sample, install the latest version of the Azure CLI. To start, run `az login` to create a connection with Azure.

Samples for the Azure CLI are written for the `bash` shell. To run this sample in Windows PowerShell or Command Prompt, you may need to change elements of the script.

- Azure CLI
- Portal

```
resourceGroup="appResourcesGroup"
# Select subscription and create resource group (if you have not created yet)
az account set --subscription <subscriptionID>
az group create --name $resourceGroup --location eastus

# Get object ID of your identity
userid=$(az ad user show --upn-or-object-id example@contoso.org --query objectId --output tsv)
# Get role definition ID for the Owner role
roleid=$(az role definition list --name Owner --query [].name --output tsv)

# Create managed application definition resource
az managedapp definition create \
  --name "ManagedUsersAppDefinition" \
  --location "eastus" \
  --resource-group $resourceGroup \
  --lock-level ReadOnly \
  --display-name "Managed users app definition" \
  --description "Managed application with Azure Custom Provider" \
  --authorizations "$userid:$roleid" \
  --package-file-uri "path to your app.zip package"
```

Managed application instance

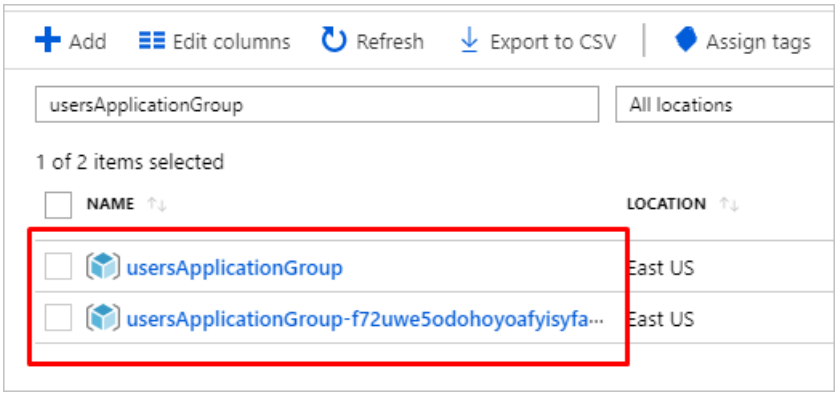
When managed application definition is deployed, run the script below or follow the steps in Azure portal to deploy your managed application instance with custom provider:

- Azure CLI
- Portal

```
appResourcesGroup="appResourcesGroup"
applicationGroup="usersApplicationGroup"

# Create resource group for managed application instance
az group create --name $applicationGroup --location eastus

# Get ID of managed application definition
appid=$(az managedapp definition show --name ManagedUsersAppDefinition --resource-group $appResourcesGroup --
query id --output tsv)

# Create the managed application
az managedapp create \
  --name ManagedUsersApp \
  --location "eastus" \
  --kind "Servicecatalog" \
  --resource-group $applicationGroup \
  --managedapp-definition-id $appid \
  --managed-rg-id "managedResourcesGroup" \
  --parameters "{\"funcname\": {\"value\": \"managedusersappfunction\"}, \"storageName\": {\"value\":
\"managedusersappstorage\"}}"
```

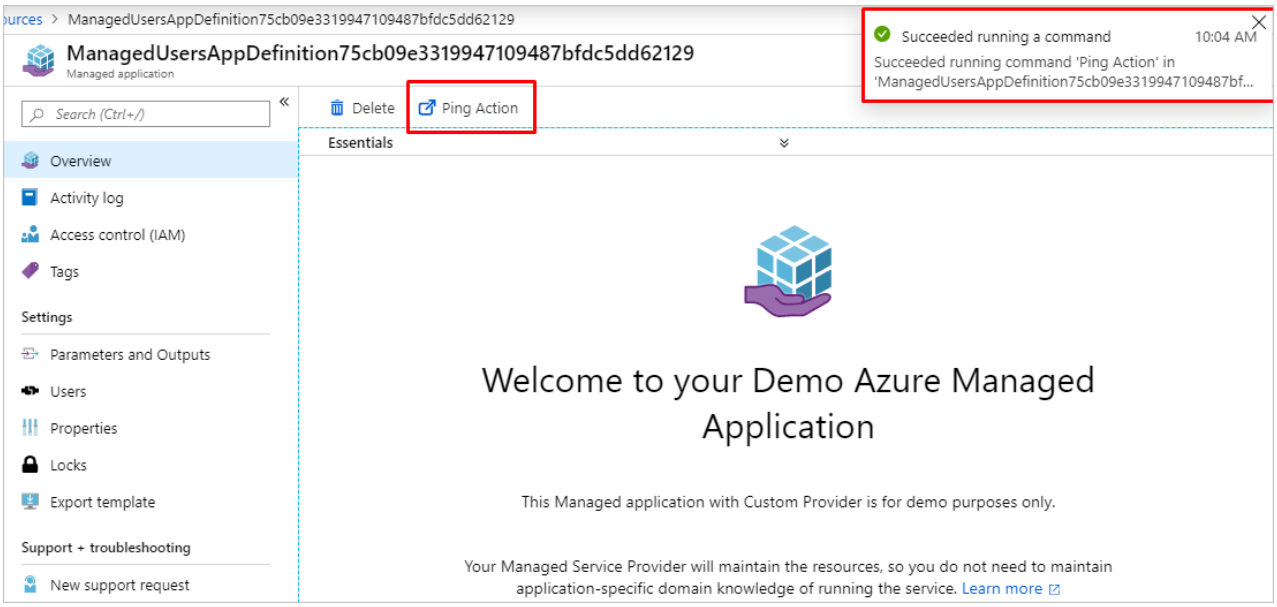
Custom actions and resources

After the service catalog application instance has been deployed, you have two new resource groups. First resource group `applicationGroup` contains an instance of the managed application, second resource group `managedResourceGroup` holds the resources for the managed application, including **custom provider**.

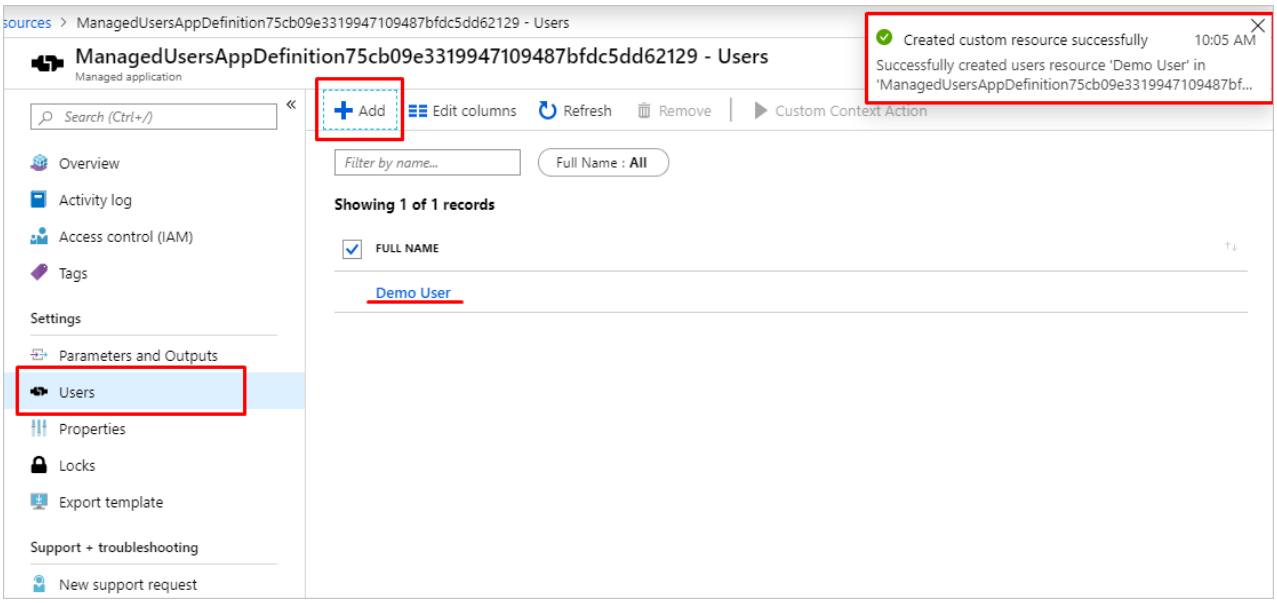


You can go to managed application instance and perform **custom action** in "Overview" page, create **users** custom resource in "Users" page and run **custom context action** on custom resource.

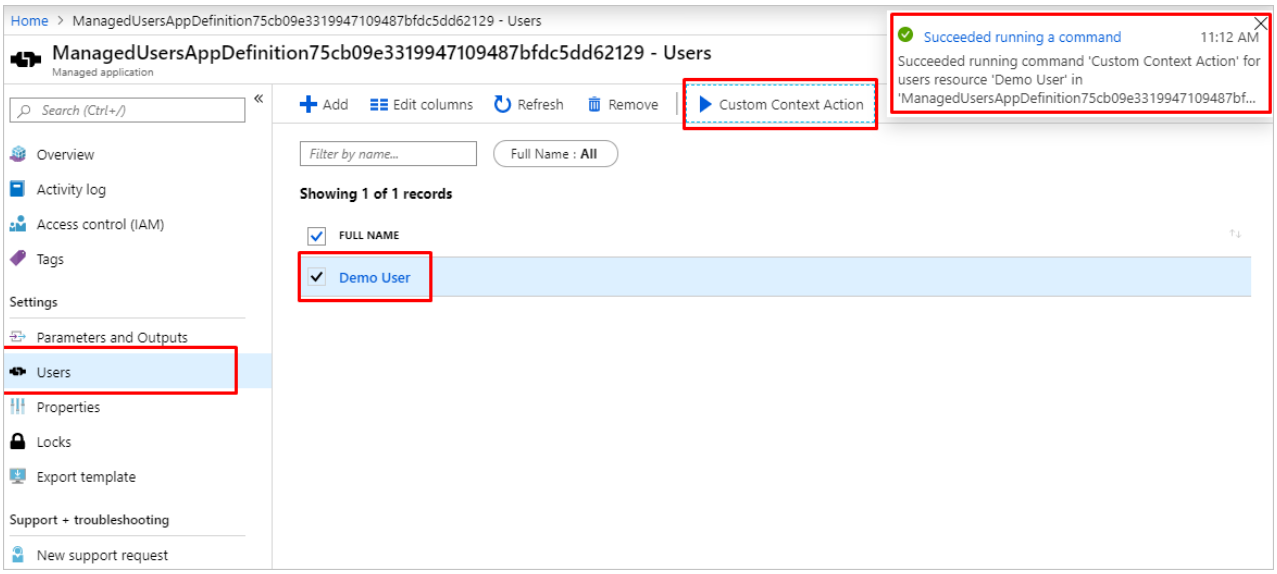
- Go to "Overview" page and click "Ping Action" button:



- Go to "Users" page and click "Add" button. Provide inputs for creating a resource and submit the form:



- Go to "Users" page, select a "users" resource and click "Custom Context Action":



Clean up resources

In the preceding steps, you created Azure resources in a resource group. If you don't expect to need these resources in the future, you can delete them by deleting the resource group.

From the Azure portal menu or **Home** page, select **Resource groups**, and on the **Resource groups** page, select **myResourceGroup**.

On the **myResourceGroup** page, make sure that the listed resources are the ones you want to delete.

Select **Delete**, type **myResourceGroup** in the text box, and then select **Delete**.

Looking for help

If you have questions about Azure Managed Applications, try asking on Stack Overflow. A similar question may have already been asked and answered, so check first before posting. Add the tag `azure-managedapps` to get a fast response!

Next steps

To publish your managed application to the Azure Marketplace, see Azure managed applications in the Marketplace.

Learn more about Azure Custom Providers.

Azure CLI Samples

1/2/2020 • 2 minutes to read • Edit Online

The following table includes links to bash scripts for Azure Managed Applications that use the Azure CLI.

Create managed application	
Create managed application definition	Creates a managed application definition in the service catalog.
Deploy managed application	Deploys a managed application from the service catalog.
Update managed resource group	
Get resources in managed resource group and resize VMs	Gets resources from the managed resource group, and resizes the VMs.

Azure PowerShell samples

1/2/2020 • 2 minutes to read • Edit Online

The following table includes links to scripts for Azure Managed Applications that use the Azure PowerShell.

Create managed application	
Create managed application definition	Creates a managed application definition in the service catalog.
Deploy managed application	Deploys a managed application from the service catalog.
Update managed resource group	
Get resources in managed resource group and resize VMs	Gets resources from the managed resource group, and resizes the VMs.

Sample projects for Azure managed applications

1/2/2020 • 2 minutes to read • Edit Online

The following table links to sample Azure managed applications in GitHub.

Examples	
Managed Application (Trial or Production) into a new or existing virtual network	Demonstrates how you can create flexible deployment options for customers. This managed application can be deployed to a new virtual network or an existing virtual network. Customers can specify either trial or production version of the managed applications.
Managed Azure Storage Account	Deploys a single storage account. Use this sample project as an introduction to creating managed applications.
Managed Service Fabric with Azure management services	Deploys a service fabric cluster and virtual machine scale sets. Includes storage accounts for logging and diagnostics.
Managed Web Application (IaaS) with Azure management services	Deploys a virtual machine that hosts a web application.
Managed SQL 2017 IaaS with automated patching and backup	Deploys a virtual machine that hosts SQL 2017.

View definition artifact in Azure Managed Applications

1/2/2020 • 6 minutes to read • Edit Online

View definition is an optional artifact in Azure Managed Applications. It allows to customize overview page and add more views such as Metrics and Custom resources.

This article provides an overview of view definition artifact and its capabilities.

View definition artifact

The view definition artifact must be named **viewDefinition.json** and placed at the same level as **createUiDefinition.json** and **mainTemplate.json** in the .zip package that creates a managed application definition. To learn how to create the .zip package and publish a managed application definition, see Publish an Azure Managed Application definition

View definition schema

The **viewDefinition.json** file has only one top level `views` property, which is an array of views. Each view is shown in the managed application user interface as a separate menu item in the table of contents. Each view has a `kind` property that sets the type of the view. It must be set to one of the following values: Overview, Metrics, CustomResources, Associations. For more information, see current JSON schema for viewDefinition.json.

Sample JSON for view definition:

```
{
    "$schema": "https://schema.management.azure.com/schemas/viewdefinition/0.0.1-
preview/ViewDefinition.json#",
    "contentVersion": "0.0.0.1",
    "views": [
        {
            "kind": "Overview",
            "properties": {
                "header": "Welcome to your Azure Managed Application",
                "description": "This managed application is for demo purposes only.",
                "commands": [
                    {
                        "displayName": "Test Action",
                        "path": "testAction"
                    }
                ]
            }
        },
        {
            "kind": "Metrics",
            "properties": {
                "displayName": "This is my metrics view",
                "version": "1.0.0",
                "charts": [
                    {
                        "displayName": "Sample chart",
                        "chartType": "Bar",
                        "metrics": [
                            {
                                "name": "Availability",
                                "aggregationType": "avg",
```

```
                                "resourceTagFilter": [ "tag1" ],
                                "resourceType": "Microsoft.Storage/storageAccounts",
                                "namespace": "Microsoft.Storage/storageAccounts"
                            }
                        ]
                    }
                ]
            }
        },
        {
            "kind": "CustomResources",
            "properties": {
                "displayName": "Test custom resource type",
                "version": "1.0.0",
                "resourceType": "testCustomResource",
                "createUIDefinition": { },
                "commands": [
                    {
                        "displayName": "Custom Context Action",
                        "path": "testCustomResource/testContextAction",
                        "icon": "Stop",
                        "createUIDefinition": { }
                    }
                ],
                "columns": [
                    {"key": "name", "displayName": "Name"},
                    {"key": "properties.myProperty1", "displayName": "Property 1"},
                    {"key": "properties.myProperty2", "displayName": "Property 2", "optional": true}
                ]
            }
        },
        {
            "kind": "Associations",
            "properties": {
                "displayName": "Test association resource type",
                "version": "1.0.0",
                "targetResourceType": "Microsoft.Compute/virtualMachines",
                "createUIDefinition": { }
            }
        }
    ]
}
```

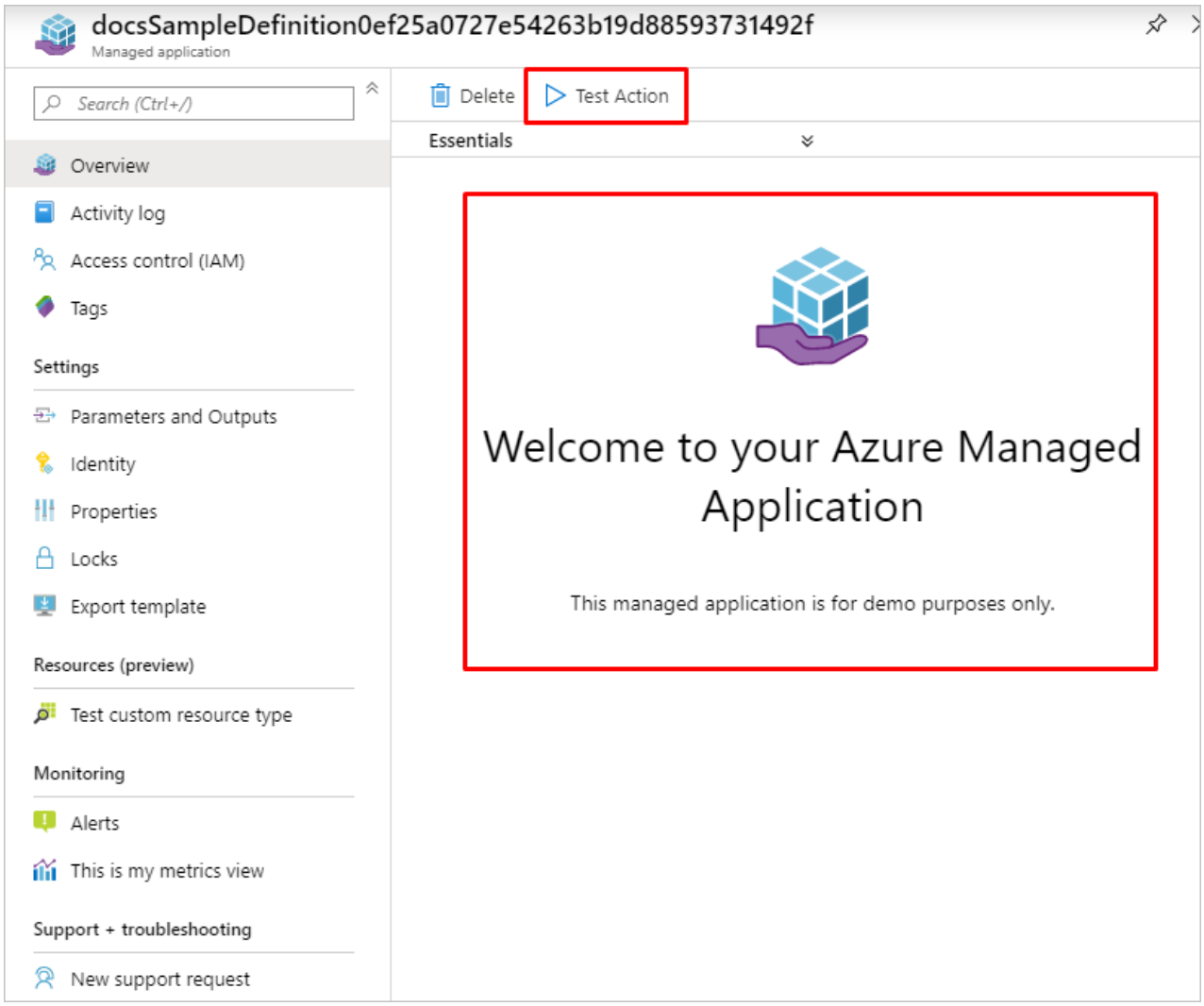
Overview

`"kind": "Overview"`

When you provide this view in **viewDefinition.json**, it overrides the default Overview page in your managed application.

```
{
    "kind": "Overview",
    "properties": {
        "header": "Welcome to your Azure Managed Application",
        "description": "This managed application is for demo purposes only.",
        "commands": [
            {
                "displayName": "Test Action",
                "path": "testAction"
            }
        ]
    }
}
```

PROPERTY	REQUIRED	DESCRIPTION
header	No	The header of the overview page.
description	No	The description of your managed application.
commands	No	The array of additional toolbar buttons of the overview page, see commands.



Metrics

`"kind": "Metrics"`

The metrics view enables you to collect and aggregate data from your managed application resources in Azure Monitor Metrics.

```
{
    "kind": "Metrics",
    "properties": {
        "displayName": "This is my metrics view",
        "version": "1.0.0",
        "charts": [
            {
                "displayName": "Sample chart",
                "chartType": "Bar",
                "metrics": [
                    {
                        "name": "Availability",
                        "aggregationType": "avg",
                        "resourceTagFilter": [ "tag1" ],
                        "resourceType": "Microsoft.Storage/storageAccounts",
                        "namespace": "Microsoft.Storage/storageAccounts"
                    }
                ]
            }
        ]
    }
}
```

PROPERTY	REQUIRED	DESCRIPTION
displayName	No	The displayed title of the view.
version	No	The version of the platform used to render the view.
charts	Yes	The array of charts of the metrics page.

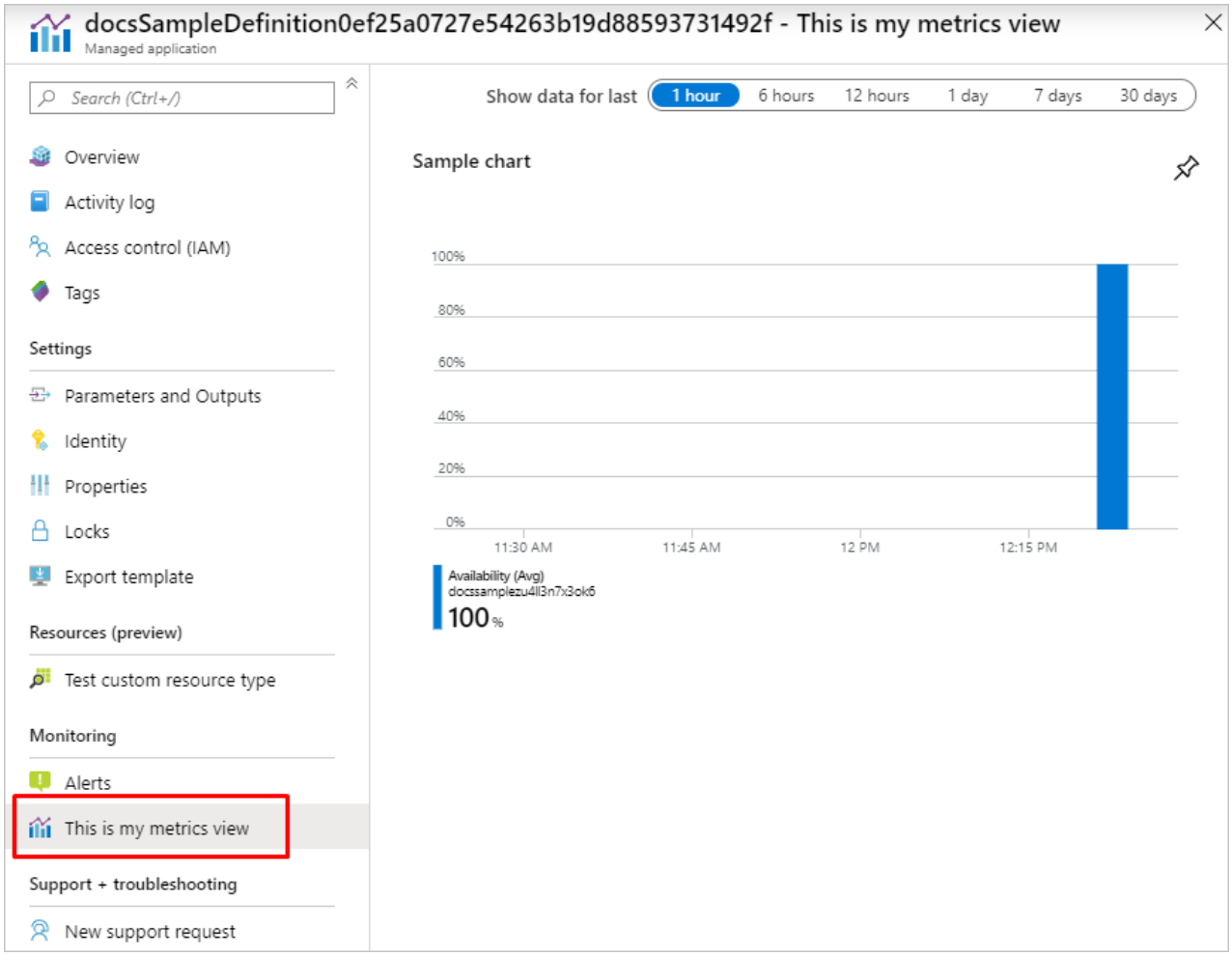
Chart

PROPERTY	REQUIRED	DESCRIPTION
displayName	Yes	The displayed title of the chart.
chartType	No	The visualization to use for this chart. By default, it uses a line chart. Supported chart types: `Bar, Line, Area, Scatter`.
metrics	Yes	The array of metrics to plot on this chart. To learn more about metrics supported in Azure portal, see Supported metrics with Azure Monitor

Metric

PROPERTY	REQUIRED	DESCRIPTION
name	Yes	The name of the metric.
aggregationType	Yes	The aggregation type to use for this metric. Supported aggregation types: `none, sum, min, max, avg, unique, percentile, count`

PROPERTY	REQUIRED	DESCRIPTION
namespace	No	Additional information to use when determining the correct metrics provider.
resourceTagFilter	No	The resource tags array (will be separated with `or` word) for which metrics would be displayed. Applies on top of resource type filter.
resourceType	Yes	The resource type for which metrics would be displayed.



Custom resources

`"kind": "CustomResources"`

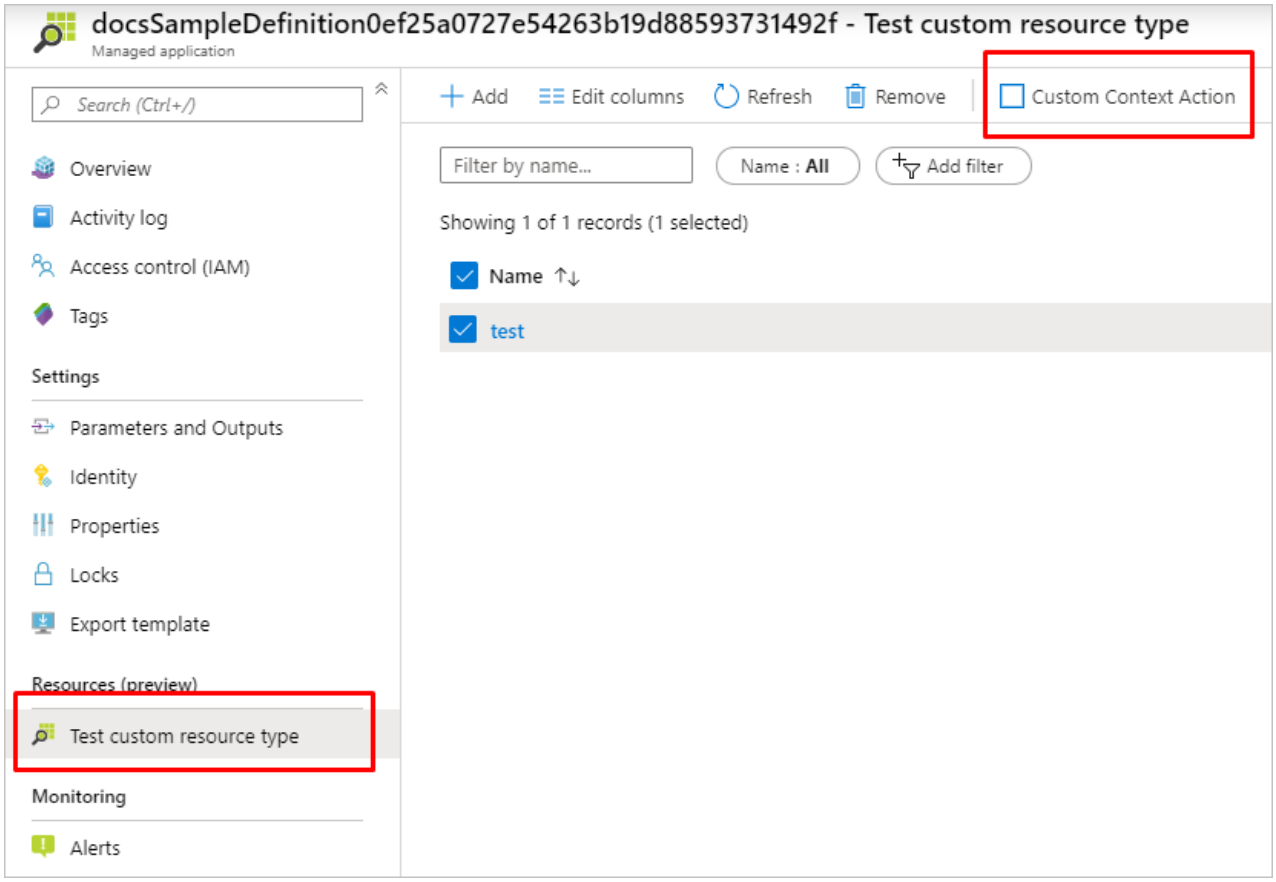
You can define multiple views of this type. Each view represents a **unique** custom resource type from the custom provider you defined in **mainTemplate.json**. For an introduction to custom providers, see Azure Custom Providers Preview overview.

In this view you can perform GET, PUT, DELETE and POST operations for your custom resource type. POST operations could be global custom actions or custom actions in a context of your custom resource type.

```
{
    "kind": "CustomResources",
    "properties": {
        "displayName": "Test custom resource type",
        "version": "1.0.0",
        "resourceType": "testCustomResource",
        "icon": "Polychromatic.ResourceList",
        "createUIDefinition": { },
        "commands": [
            {
                "displayName": "Custom Context Action",
                "path": "testCustomResource/testContextAction",
                "icon": "Stop",
                "createUIDefinition": { },
            }
        ],
        "columns": [
            {"key": "name", "displayName": "Name"},
            {"key": "properties.myProperty1", "displayName": "Property 1"},
            {"key": "properties.myProperty2", "displayName": "Property 2", "optional": true}
        ]
    }
}
```

PROPERTY	REQUIRED	DESCRIPTION
displayName	Yes	The displayed title of the view. The title should be **unique** for each CustomResources view in your **viewDefinition.json**.
version	No	The version of the platform used to render the view.
resourceType	Yes	The custom resource type. Must be a **unique** custom resource type of your custom provider.
icon	No	The icon of the view. List of example icons is defined in JSON Schema.
createUIDefinition	No	Create UI Definition schema for create custom resource command. For an introduction to creating UI definitions, see Getting started with CreateUiDefinition
commands	No	The array of additional toolbar buttons of the CustomResources view, see commands.

PROPERTY	REQUIRED	DESCRIPTION
columns	No	The array of columns of the custom resource. If not defined the `name` column will be shown by default. The column must have `"key"` and `"displayName"`. For key, provide the key of the property to display in a view. If nested, use dot as delimiter, for example, `"key": "name"` or `"key": "properties.property1"`. For display name, provide the display name of the property to display in a view. You can also provide an `"optional"` property. When set to true, the column is hidden in a view by default.



Commands

Commands is an array of additional toolbar buttons that are displayed on page. Each command represents a POST action from your Azure Custom Provider defined in **mainTemplate.json**. For an introduction to custom providers, see Azure Custom Providers overview.

```
{
    "commands": [
        {
            "displayName": "Start Test Action",
            "path": "testAction",
            "icon": "Start",
            "createUIDefinition": { }
        },
    ]
}
```

PROPERTY	REQUIRED	DESCRIPTION
displayName	Yes	The displayed name of the command button.
path	Yes	The custom provider action name. The action must be defined in **mainTemplate.json**.
icon	No	The icon of the command button. List of example icons is defined in JSON Schema.
createUIDefinition	No	Create UI Definition schema for command. For an introduction to creating UI definitions, see Getting started with CreateUiDefinition.

Associations

`"kind": "Associations"`

You can define multiple views of this type. This view allows you to link existing resources to the managed application through the custom provider you defined in **mainTemplate.json**. For an introduction to custom providers, see Azure Custom Providers Preview overview.

In this view you can extend existing Azure resources based on the `targetResourceType`. When a resource is selected, it will create an onboarding request to the **public** custom provider, which can apply a side effect to the resource.

```
{
    "kind": "Associations",
    "properties": {
        "displayName": "Test association resource type",
        "version": "1.0.0",
        "targetResourceType": "Microsoft.Compute/virtualMachines",
        "createUIDefinition": { }
    }
}
```

PROPERTY	REQUIRED	DESCRIPTION
displayName	Yes	The displayed title of the view. The title should be **unique** for each Associations view in your **viewDefinition.json**.

PROPERTY	REQUIRED	DESCRIPTION
version	No	The version of the platform used to render the view.
targetResourceType	Yes	The target resource type. This is the resource type that will be displayed for resource onboarding.
createUIDefinition	No	Create UI Definition schema for create association resource command. For an introduction to creating UI definitions, see Getting started with CreateUiDefinition

Looking for help

If you have questions about Azure Managed Applications, try asking on Stack Overflow. A similar question may have already been asked and answered, so check first before posting. Add the tag `azure-managedapps` to get a fast response!

Next steps

- For an introduction to managed applications, see Azure Managed Application overview.
- For an introduction to custom providers, see Azure Custom Providers overview.
- For creating an Azure Managed Application with Azure Custom Providers, see Tutorial: Create managed application with custom provider actions and resource types

Deploy associations for a managed application using Azure Policy

1/2/2020 • 2 minutes to read • Edit Online

Azure policies can be used to deploy associations to associate resources to a managed application. In this article, we describe a built-in policy that deploys associations and how you can use that policy.

Built-in policy to deploy associations

Deploy associations for a managed application is a built-in policy that can be used to deploy association to associate a resource to a managed application. The policy accepts three parameters:

- Managed application ID - This ID is the resource ID of the managed application to which the resources need to be associated.
- Resource types to associate - These resource types are the list of resource types to be associated to the managed application. You can associate multiple resource types to a managed application using the same policy.
- Association name prefix - This string is the prefix to be added to the name of the association resource being created. The default value is "DeployedByPolicy".

The policy uses DeployIfNotExists evaluation. It runs after a Resource Provider has handled a create or update resource request of the selected resource type(s) and the evaluation has returned a success status code. After that, the association resource gets deployed using a template deployment. For more information on associations, see Azure Custom Providers resource onboarding

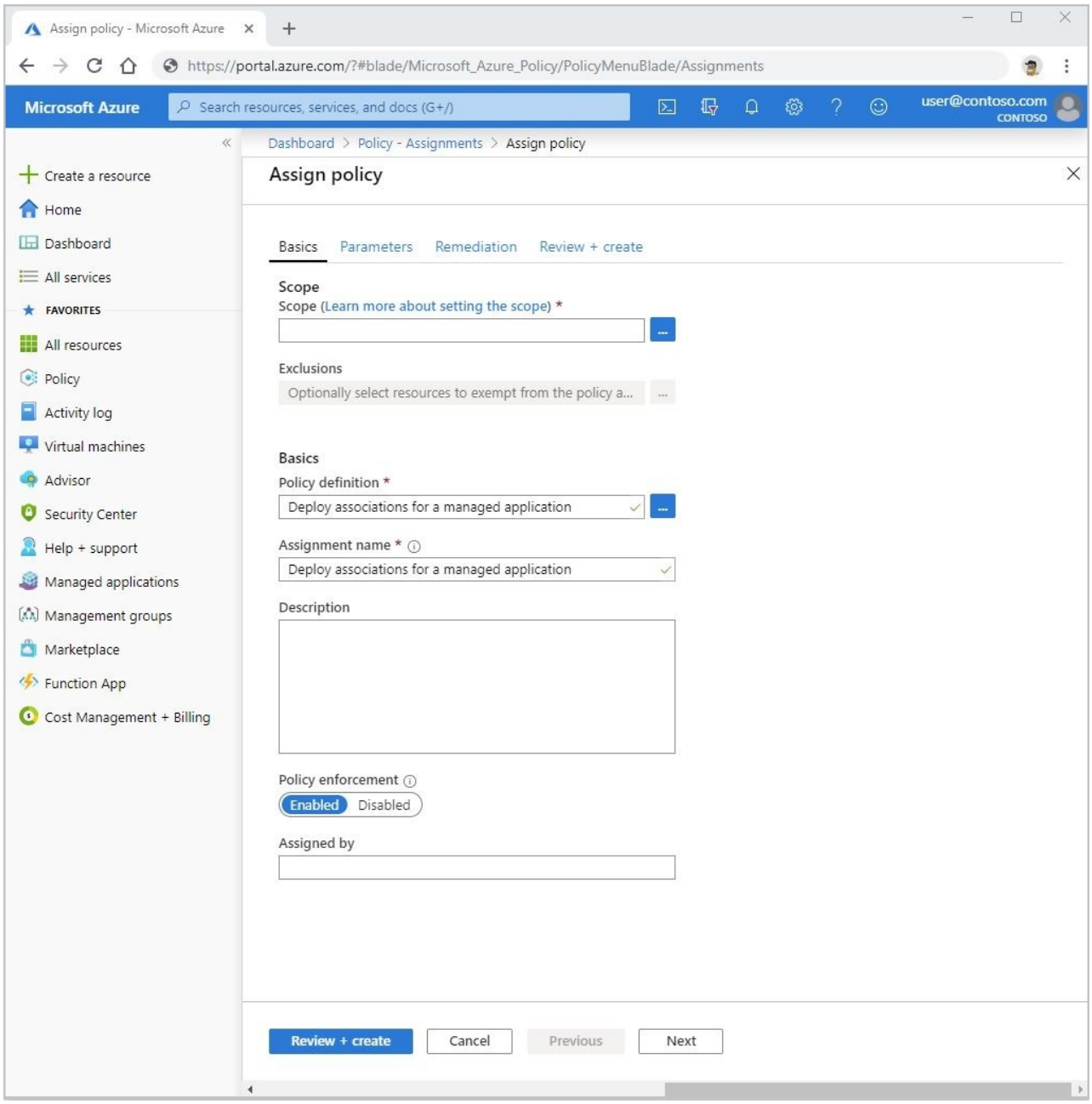
How to use the deploy associations built-in policy

Prerequisites

If the managed application needs permissions to the subscription to perform an action, the policy deployment of association resource wouldn't work without granting the permissions.

Policy assignment

To use the built-in policy, create a policy assignment and assign the Deploy associations for a managed application policy. Once the policy has been assigned successfully, the policy will identify non-compliant resources and deploy association for those resources.



Getting help

If you have questions about Azure Custom Resource Providers development, try asking them on Stack Overflow. A similar question might have already been answered, so check first before posting. Add the tag `azure-custom-providers` to get a fast response!

Next steps

In this article, you learnt about using built-in policy to deploy associations. See these articles to learn more:

- Concepts: Azure Custom Providers resource onboarding
- Tutorial: Resource onboarding with custom providers
- Tutorial: Create custom actions and resources in Azure
- Quickstart: Create a custom resource provider and deploy custom resources
- How to: Adding custom actions to an Azure REST API
- How to: Adding custom resources to an Azure REST API

Custom role definition artifact in Azure Managed Applications

1/2/2020 • 2 minutes to read • Edit Online

Custom role definition is an optional artifact in managed applications. It's used to determine what permissions the managed application needs to perform its functions.

This article provides an overview of the custom role definition artifact and its capabilities.

Custom role definition artifact

You need to name the custom role definition artifact customRoleDefinition.json. Place it at the same level as createUiDefinition.json and mainTemplate.json in the .zip package that creates a managed application definition. To learn how to create the .zip package and publish a managed application definition, see Publish a managed application definition.

Custom role definition schema

The customRoleDefinition.json file has a top-level `roles` property that's an array of roles. These roles are the permissions that the managed application needs to function. Currently, only built-in roles are allowed, but you can specify multiple roles. A role can be referenced by the ID of the role definition or by the role name.

Sample JSON for custom role definition:

```
{
    "contentVersion": "0.0.0.1",
    "roles": [
        {
            "properties": {
                "roleName": "Contributor"
            }
        },
        {
            "id": "acdd72a7-3385-48ef-bd42-f606fba81ae7"
        },
        {
            "id": "/providers/Microsoft.Authorization/roledefinitions/9980e02c-c2be-4d73-94e8-173b1dc7cf3c"
        }
    ]
}
```

Roles

A role is composed of either a `$.properties.roleName` or an `id`:

```
{
    "id": null,
    "properties": {
        "roleName": "Contributor"
    }
}
```

> **NOTE**
> You can use either the `id` or `roleName` field. Only one is required. These fields are used to look up the role definition that should be applied. If both are supplied, the `id` field will be used.

PROPERTY	REQUIRED?	DESCRIPTION
id	Yes	The ID of the built-in role. You can use the full ID or just the GUID.
roleName	Yes	The name of the built-in role.

CreateUiDefinition.json for Azure managed application's create experience

1/2/2020 • 2 minutes to read • Edit Online

This document introduces the core concepts of the **createUiDefinition.json** file which Azure portal uses to define the user interface when creating a managed application.

The template is as follows

```
{
   "$schema": "https://schema.management.azure.com/schemas/0.1.2-
preview/CreateUIDefinition.MultiVm.json#",
   "handler": "Microsoft.Azure.CreateUIDef",
   "version": "0.1.2-preview",
   "parameters": {
      "basics": [ ],
      "steps": [ ],
      "outputs": { },
      "resourceTypes": [ ]
   }
}
```

A CreateUiDefinition always contains three properties:

- handler
- version
- parameters

The handler should always be `Microsoft.Azure.CreateUIDef`, and the latest supported version is `0.1.2-preview`.

The schema of the parameters property depends on the combination of the specified handler and version. For managed applications, the supported properties are `basics`, `steps`, and `outputs`. The basics and steps properties contain the elements - like textboxes and dropdowns - to be displayed in the Azure portal. The outputs property is used to map the output values of the specified elements to the parameters of the Azure Resource Manager deployment template.

Including `$schema` is recommended, but optional. If specified, the value for `version` must match the version within the `$schema` URI.

You can use a JSON editor to create your createUiDefinition then test it in the createUiDefinition Sandbox to preview it. For more information about the sandbox, see Test your portal interface for Azure Managed Applications.

Basics

Basics is the first step generated when the Azure portal parses the file. In addition to displaying the elements specified in `basics`, the portal injects elements for users to choose the subscription, resource group, and location for the deployment. When possible, elements that query deployment-wide parameters, like the name of a cluster or administrator credentials, should go in this step.

Steps

The steps property can contain zero or more additional steps to display after basics, each of which contains one or more elements. Consider adding steps per role or tier of the application being deployed. For example, add a step for master node inputs, and a step for the worker nodes in a cluster.

Outputs

The Azure portal uses the `outputs` property to map elements from `basics` and `steps` to the parameters of the Azure Resource Manager deployment template. The keys of this dictionary are the names of the template parameters, and the values are properties of the output objects from the referenced elements.

To set the managed application resource name, you must include a value named `applicationResourceName` in the outputs property. If you don't set this value, the application assigns a GUID for the name. You can include a textbox in the user interface that requests a name from the user.

```
"outputs": {
    "vmName": "[steps('appSettings').vmName]",
    "trialOrProduction": "[steps('appSettings').trialOrProd]",
    "userName": "[steps('vmCredentials').adminUsername]",
    "pwd": "[steps('vmCredentials').vmPwd.password]",
    "applicationResourceName": "[steps('appSettings').vmName]"
}
```

Resource types

To filter the available locations to only those locations that support the resource types to deploy, provide an array of the resource types. If you provide more than one resource type, only those locations that support all of the resource types are returned. This property is optional.

```
{
    "$schema": "https://schema.management.azure.com/schemas/0.1.2-
preview/CreateUIDefinition.MultiVm.json#",
    "handler": "Microsoft.Azure.CreateUIDef",
    "version": "0.1.2-preview",
    "parameters": {
      "resourceTypes": ["Microsoft.Compute/disks"],
      "basics": [
        ...
```

Functions

CreateUiDefinition provides functions for working with elements' inputs and outputs, and features such as conditionals. These functions are similar in both syntax and functionality to Azure Resource Manager template functions.

Next steps

The createUiDefinition.json file itself has a simple schema. The real depth of it comes from all the supported elements and functions. Those items are described in greater detail at:

- Elements
- Functions

A current JSON schema for createUiDefinition is available here: https://schema.management.azure.com/schemas/0.1.2-preview/CreateUIDefinition.MultiVm.json.

For an example user interface file, see createUiDefinition.json.

Test your portal interface for Azure Managed Applications

1/2/2020 • 3 minutes to read • Edit Online

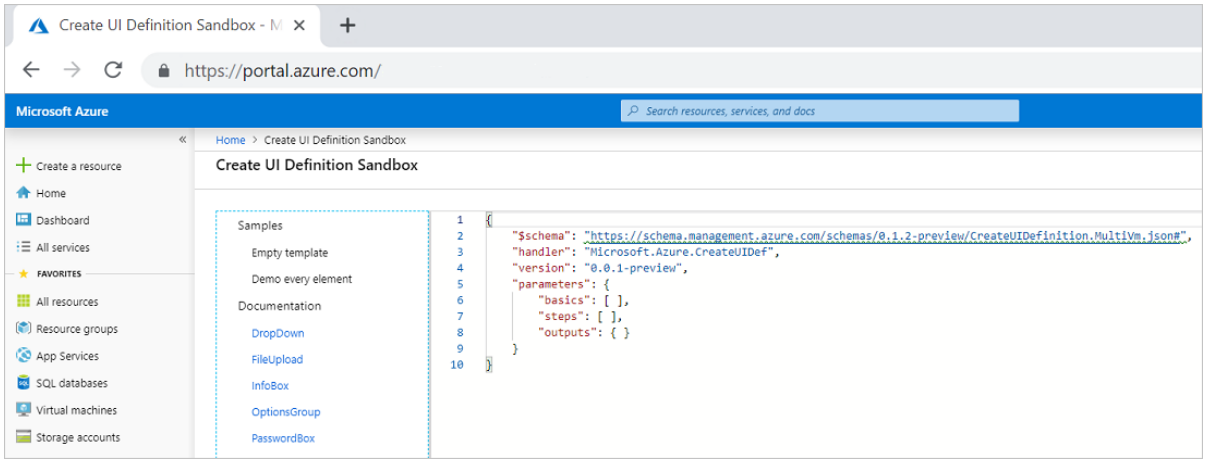
After creating the createUiDefinition.json file for your managed application, you need to test the user experience. To simplify testing, use a sandbox environment that loads your file in the portal. You don't need to actually deploy your managed application. The sandbox presents your user interface in the current, full-screen portal experience. Or, you can use a script for testing the interface. Both approaches are shown in this article. The sandbox is the recommended way to preview the interface.

Prerequisites

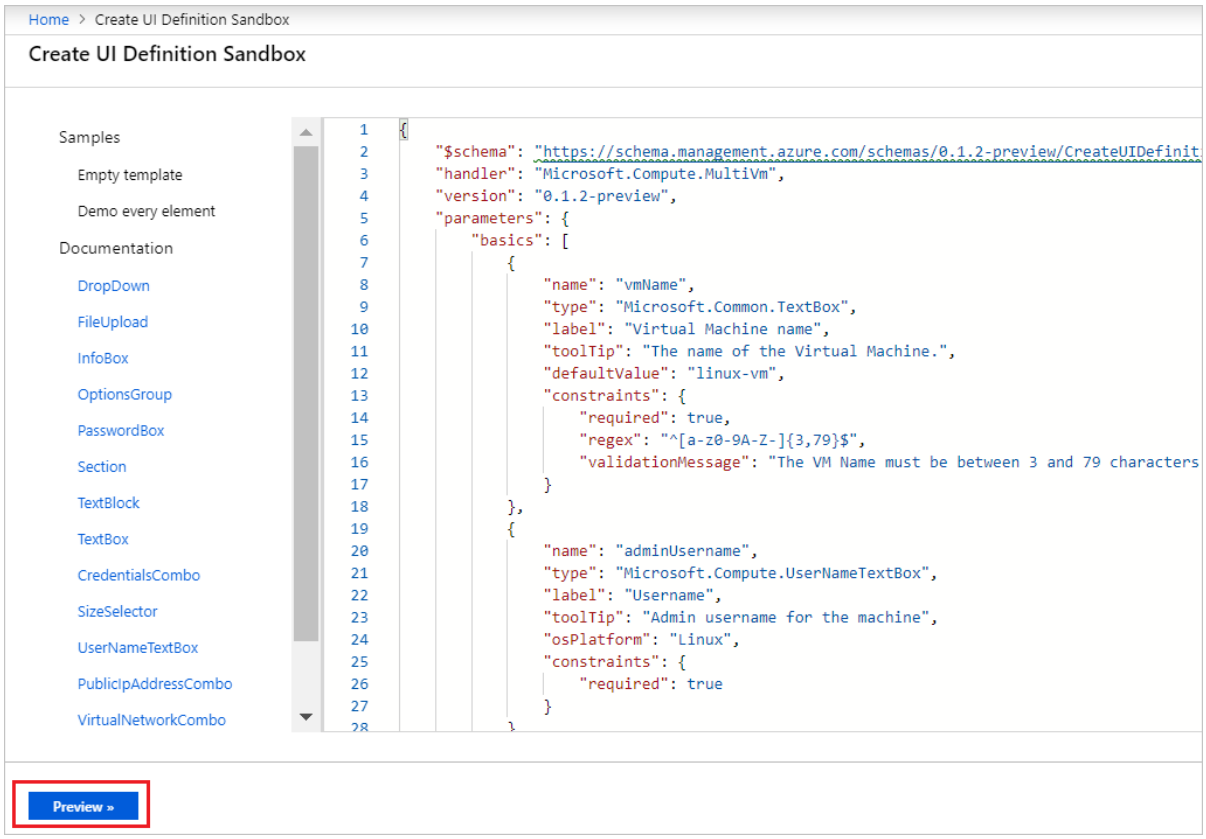
- A **createUiDefinition.json** file. If you don't have this file, copy the sample file.
- An Azure subscription. If you don't have an Azure subscription, create a free account before you begin.

Use sandbox

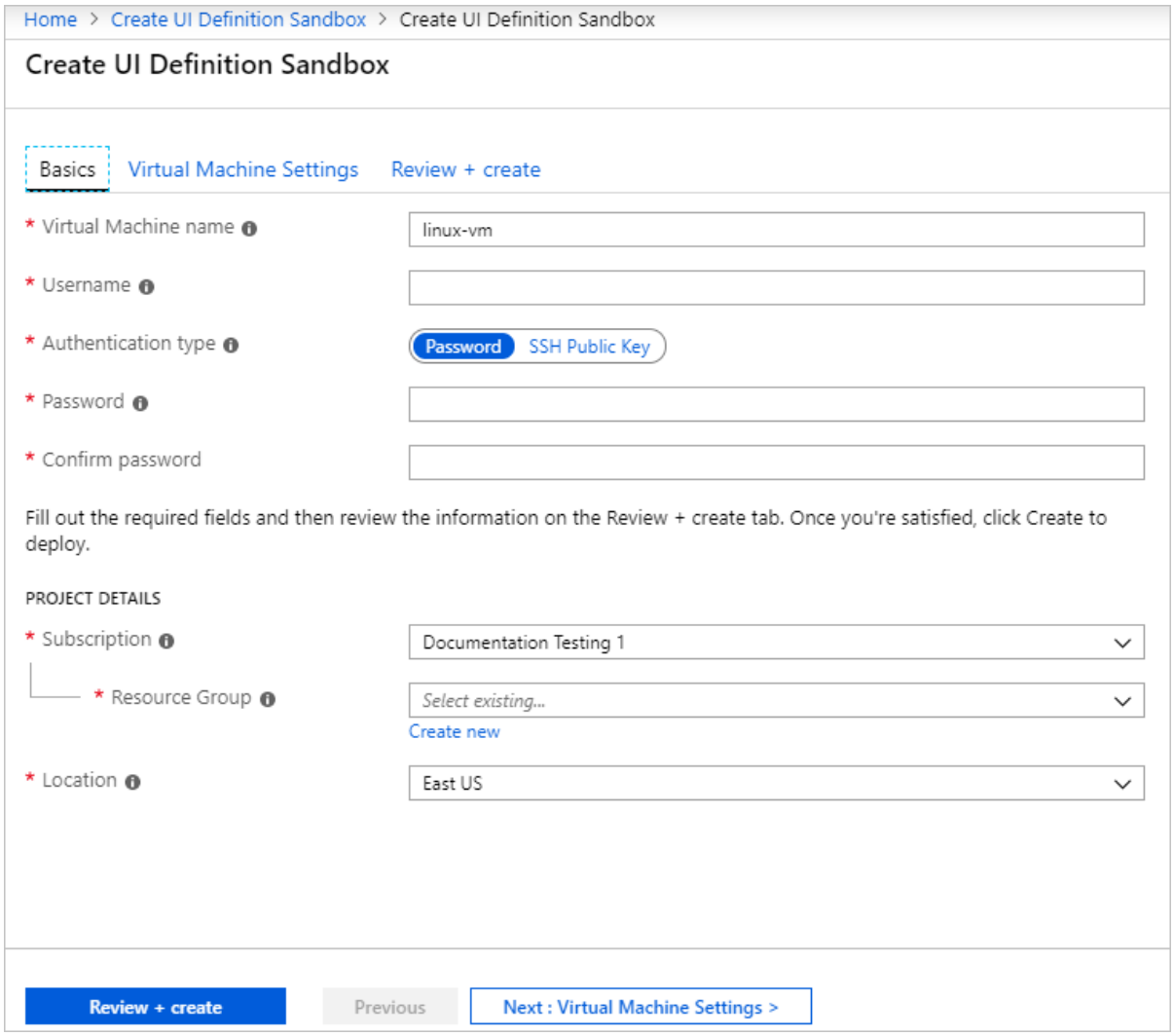
1. Open the Create UI Definition Sandbox.



2. Replace the empty definition with the contents of your createUiDefinition.json file. Select **Preview**.



3. The form you created is displayed. You can step through the user experience and fill in the values.



Troubleshooting

If your form doesn't display after selecting **Preview**, you may have a syntax error. Look for the red indicator on the

right scroll bar and navigate to it.

```
85                      "Standard_LRS",
86                      "Standard_GRS"
87                  ]
88              }
89          },
90          {
91              "name": "publicIpAddress",
92              "type": "Microsoft.Network.PublicIpAddressCombo",
93              "label": {
94                  "publicIpAddress": "Public IP Address for the VM",
95                  "domainNameLabel": "DNS Prefix for the public IP Address"
96              },
97              "toolTip": {
98                  "publicIpAddress": "Public IP Address for the VM"
99                  "domainNameLabel": "DNS Prefix for the public IP Address, must be globally unique"
100             },
101             "defaultValue": {
102                 "publicIpAddressName": "[concat(basics('vmName'), '-ip')]",
103                 "domainNameLabel": "[concat(basics('vmName'), '-', take(replace(guid(), '-', ''), 10))]"
104             },
105             "options": {
106                 "hideExisting": false,
107                 "hideNone": false
108             },
109             "constraints": {
110                 "required": {
111                     "domainNameLabel": true
```

If your form doesn't display, and instead you see an icon of a cloud with tear drop, your form has an error, such as a missing property. Open the Web Developer Tools in your browser. The **Console** displays important messages about your interface.

```
[Microsoft_Azure_Compute]  4:39:14 PM Microsoft_Azure_Compute Microsoft_Azure_Compute: Create MultiVm: failed to parse createUIdefinition file.
responseText: {
    "handler": "Microsoft.Compute.MultiVm",
```

Use test script

To test your interface in the portal, copy one of the following scripts to your local machine:

- PowerShell side-load script - Az Module
- PowerShell side-load script - Azure Module
- Azure CLI side-load script

To see your interface file in the portal, run your downloaded script. The script creates a storage account in your Azure subscription, and uploads your createUiDefinition.json file to the storage account. The storage account is created the first time you run the script or if the storage account has been deleted. If the storage account already exists in your Azure subscription, the script reuses it. The script opens the portal and loads your file from the storage account.

Provide a location for the storage account, and specify the folder that has your createUiDefinition.json file.

For PowerShell, use:

```
.\SideLoad-AzCreateUIDefinition.ps1 `
  -StorageResourceGroupLocation southcentralus `
  -ArtifactsStagingDirectory .\100-Marketplace-Sample
```

For Azure CLI, use:

```
./sideload-createuidef.sh \
  -l southcentralus \
  -a .\100-Marketplace-Sample
```

If your createUiDefinition.json file is in the same folder as the script, and you've already created the storage account, you don't need to provide those parameters.

For PowerShell, use:

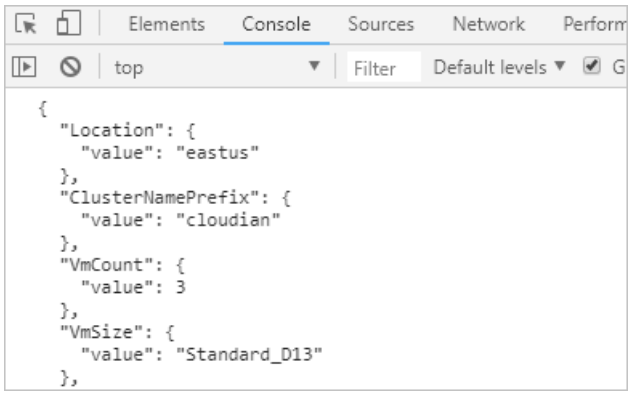
```
.\SideLoad-AzCreateUIDefinition.ps1
```

For Azure CLI, use:

```
./sideload-createuidef.sh
```

The script opens a new tab in your browser. It displays the portal with your interface for creating the managed application.

Provide values for the fields. When finished, you see the values that are passed to the template which can be found in your browser's developer tools console.



You can use these values as the parameter file for testing your deployment template.

If the portal hangs at the summary screen, there might be a bug in the output section. For example, you may have referenced a control that doesn't exist. If a parameter in the output is empty, the parameter might be referencing a property that doesn't exist. For example, the reference to the control is valid, but the property reference isn't valid.

Test your solution files

Now that you've verified your portal interface is working as expected, it's time to validate that your createUiDefinition file is properly integrated with your mainTemplate.json file. You can run a validation script test to test the content of your solution files, including the createUiDefinition file. The script validates the JSON syntax, checks for regex expressions on text fields, and makes sure the output values of the portal interface match the parameters of your template. For information on running this script, see Run static validation checks for templates.

Next steps

After validating your portal interface, learn about making your Azure managed application available in the Marketplace.

Enable and request just-in-time access for Azure Managed Applications

1/2/2020 • 2 minutes to read • Edit Online

Consumers of your managed application may be reluctant to grant you permanent access to the managed resource group. As a publisher of a manager application, you might prefer that consumers know exactly when you need to access the managed resources. To give consumers greater control over granting access to managed resources, Azure Managed Applications provides a feature called just-in-time (JIT) access, which is currently in preview.

JIT access enables you to request elevated access to a managed application's resources for troubleshooting or maintenance. You always have read-only access to the resources, but for a specific time period you can have greater access.

The work flow for granting access is:

1. You add a managed application to the marketplace and specify that JIT access is available.
2. During deployment, the consumer enables JIT access for that instance of the managed application.
3. After deployment, the consumer can change the settings for JIT access.
4. You send a request for access when you need to troubleshoot or update the managed resources.
5. The consumer approves your request.

This article focuses on the actions publishers take to enable JIT access and submit requests. To learn about approving JIT access requests, see Approve just-in-time access in Azure Managed Applications.

Add JIT access step to UI

Your CreateUiDefinition.json file is exactly like the UI file you create for permanent access, except you must include a step that lets consumers enable JIT access. To learn more about publishing your first managed application offering in the Azure Marketplace, see Azure Managed Applications in the Marketplace.

To support JIT capability for your offer, add the following content to your CreateUiDefinition.json file:

In "steps":

```
{
    "name": "jitConfiguration",
    "label": "JIT Configuration",
    "subLabel": {
        "preValidation": "Configure JIT settings for your application",
        "postValidation": "Done"
    },
    "bladeTitle": "JIT Configuration",
    "elements": [
        {
          "name": "jitConfigurationControl",
          "type": "Microsoft.Solutions.JitConfigurator",
          "label": "JIT Configuration"
        }
    ]
}
```

In "outputs":

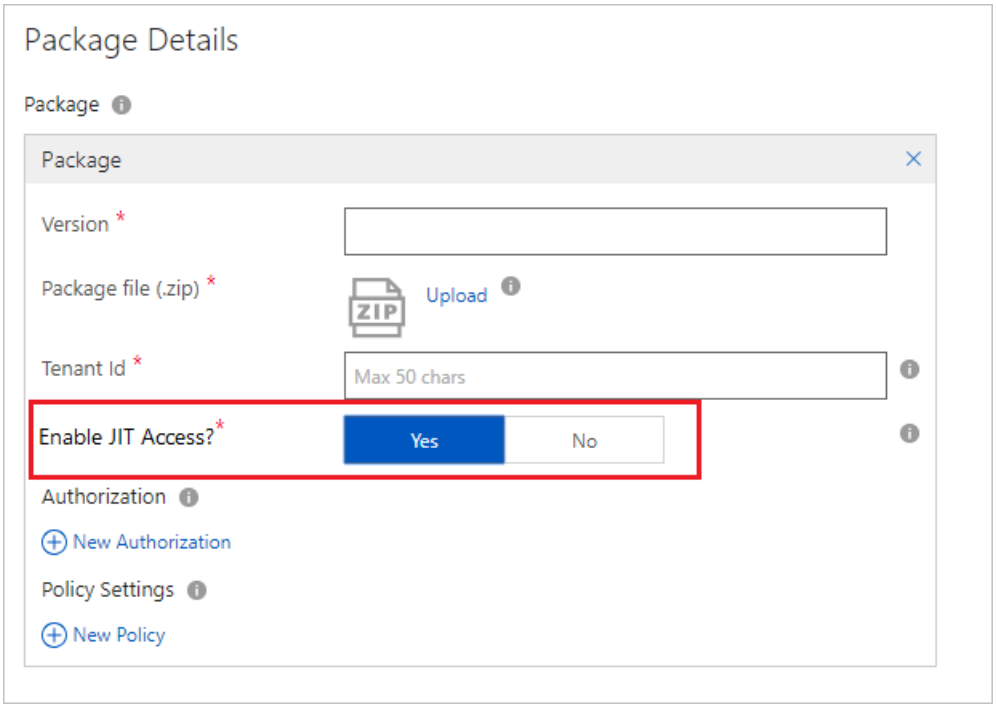
```
"jitAccessPolicy": "[steps('jitConfiguration').jitConfigurationControl]"
```

> **NOTE**
> JIT access is in preview. The schema for JIT configuration could change in future iterations.

Enable JIT access

When defining your offering in the marketplace, make sure you enable JIT access.

1. Sign in to the Cloud Partner publishing portal.
2. Provide values to publish your managed application in the marketplace. Select **Yes** for **Enable JIT Access?**



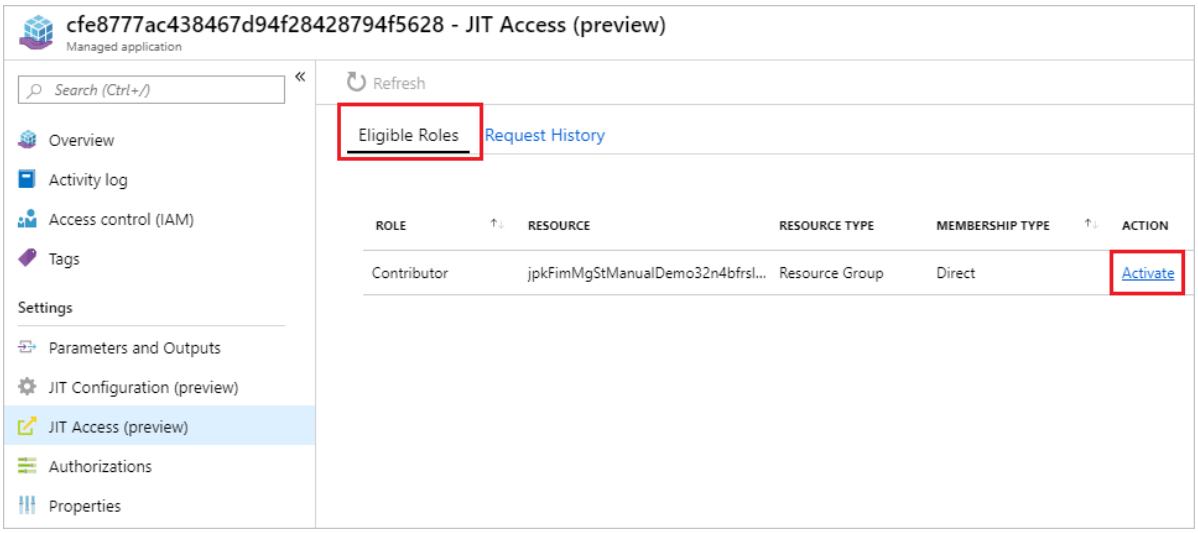
You have added a JIT configuration step to your UI, and have enabled JIT access in the marketplace offering. When consumers deploy your managed application, they can turn on JIT access for their instance.

Request access

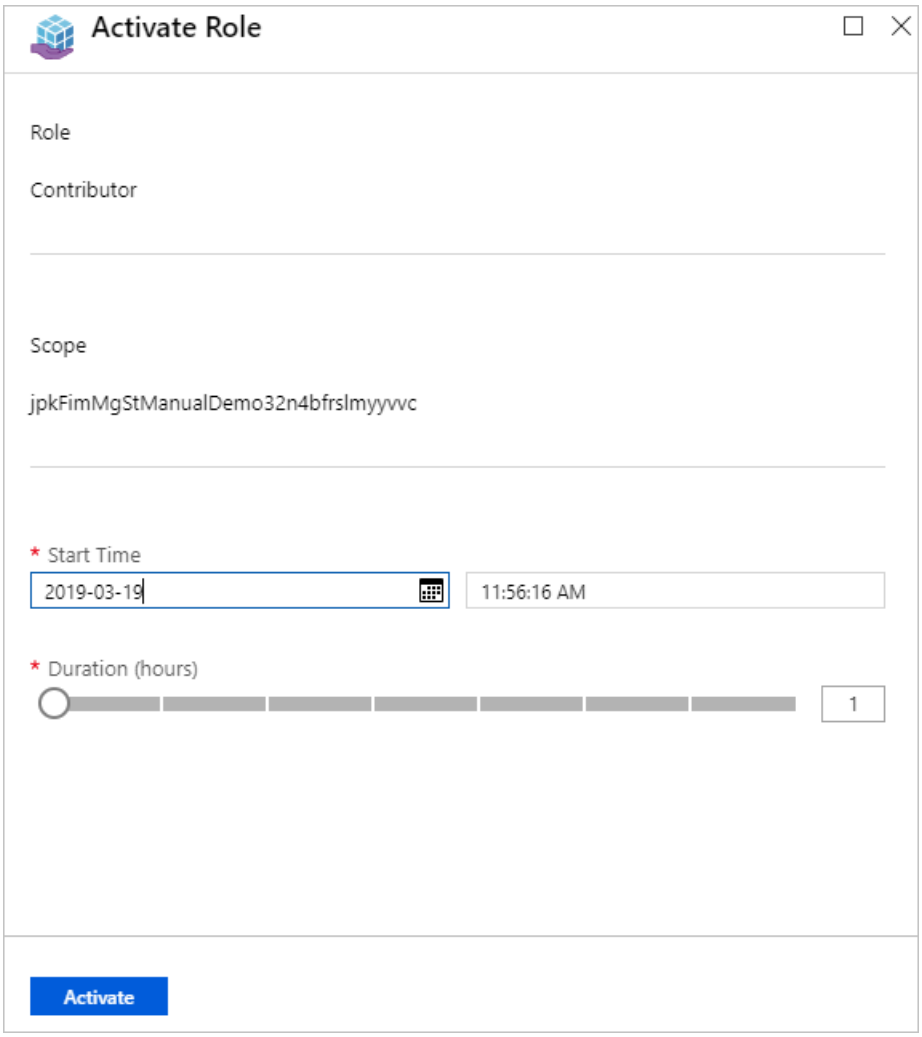
When you need to access the consumer's managed resources, you send a request for a specific role, time and duration. The consumer must then approve the request.

To send a JIT access request:

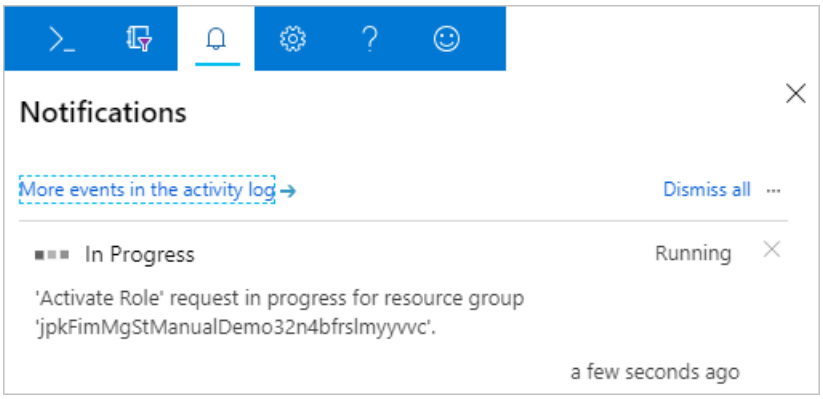
1. Select **JIT Access** for the managed application you need to access.
2. Select **Eligible Roles**, and select **Activate** in the ACTION column for the role you want.



3. On the **Activate Role** form, select a start time and duration for your role to be active. Select **Activate** to send the request.

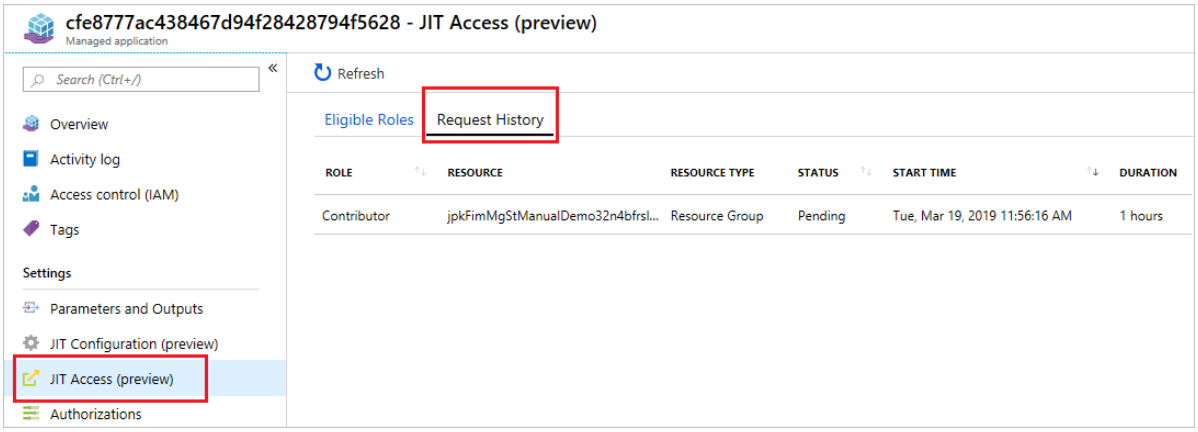


4. View the notifications to see that the new JIT request is successfully sent to the consumer.



Now, you must wait for the consumer to approve your request.

5. To view the status of all JIT requests for a managed application, select **JIT Access** and **Request History**.



Known issues

The principal ID of the account requesting JIT access must be explicitly included in the managed application definition. The account can't only be included through a group that is specified in the package. This limitation will be fixed in a future release.

Next steps

To learn about approving requests for JIT access, see Approve just-in-time access in Azure Managed Applications.

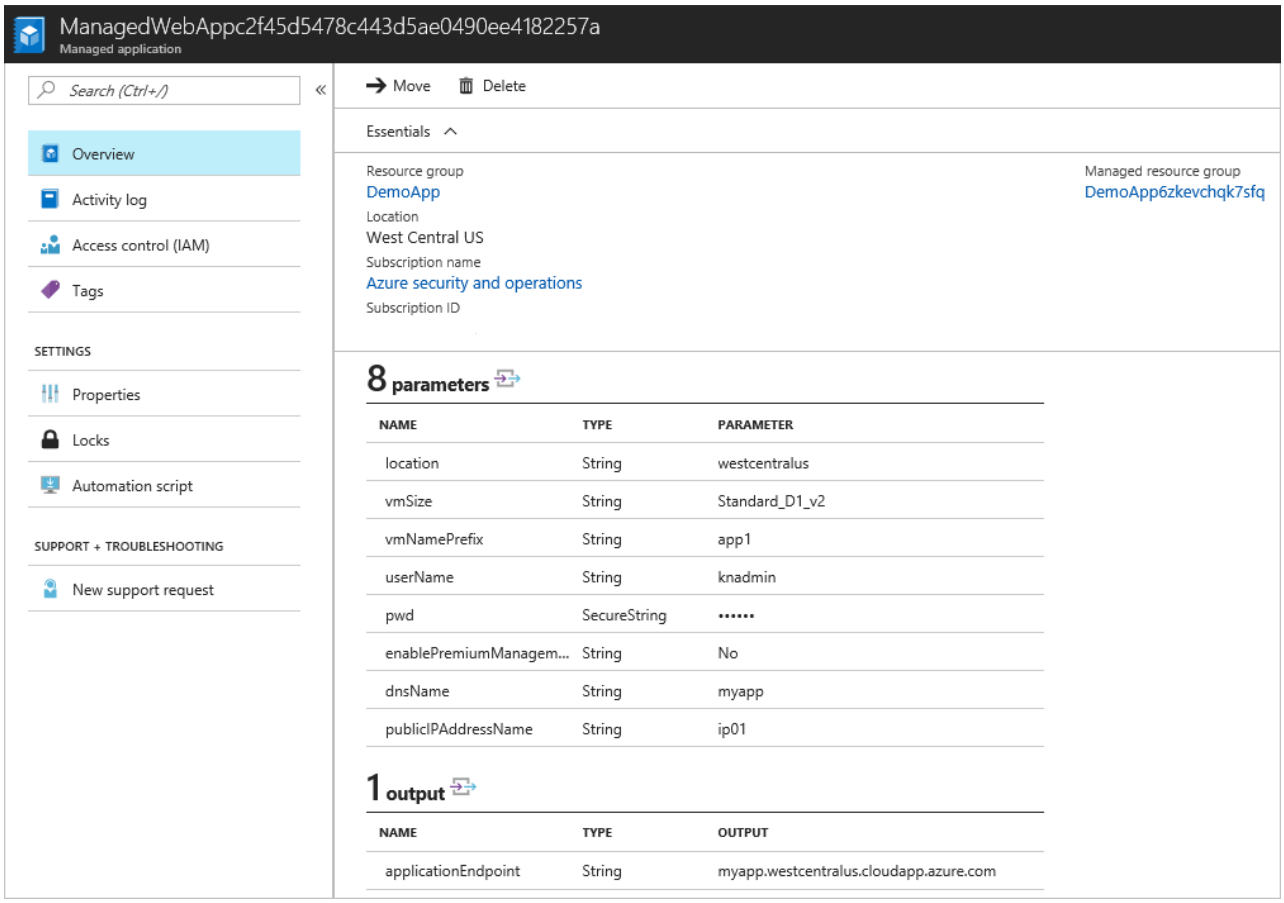
Work with resources in the managed resource group for Azure managed application

1/2/2020 • 2 minutes to read • Edit Online

This article describes how to update resources that are deployed as part of a managed application. As the publisher of a managed application, you have access to the resources in the managed resource group. To update these resources, you need to find the managed resource group associated with a managed application, and access the resource in that resource group.

This article assumes you have deployed the managed application in the Managed Web Application (IaaS) with Azure management services sample project. That managed application includes a **Standard_D1_v2** virtual machine. If you have not deployed that managed application, you can still use this article to become familiar with the steps for updating a managed resource group.

The following image shows the deployed managed application.



In this article, you use Azure CLI to:

- Identify the managed application
- Identify the managed resource group
- Identify the virtual machine resource(s) in the managed resource group
- Change the VM size (either to a smaller size if not utilized, or a larger to support more load)
- Assign a policy to the managed resource group that specifies the allowed locations

Get managed application and managed resource group

To get the managed applications in a resource group, use:

```
az managedapp list --query "[?contains(resourceGroup,'DemoApp')]"
```

To get the ID of the managed resource group, use:

```
az managedapp list --query "[?contains(resourceGroup,'DemoApp')].{ managedResourceGroup:managedResourceGroupId
}"
```

Resize VMs in managed resource group

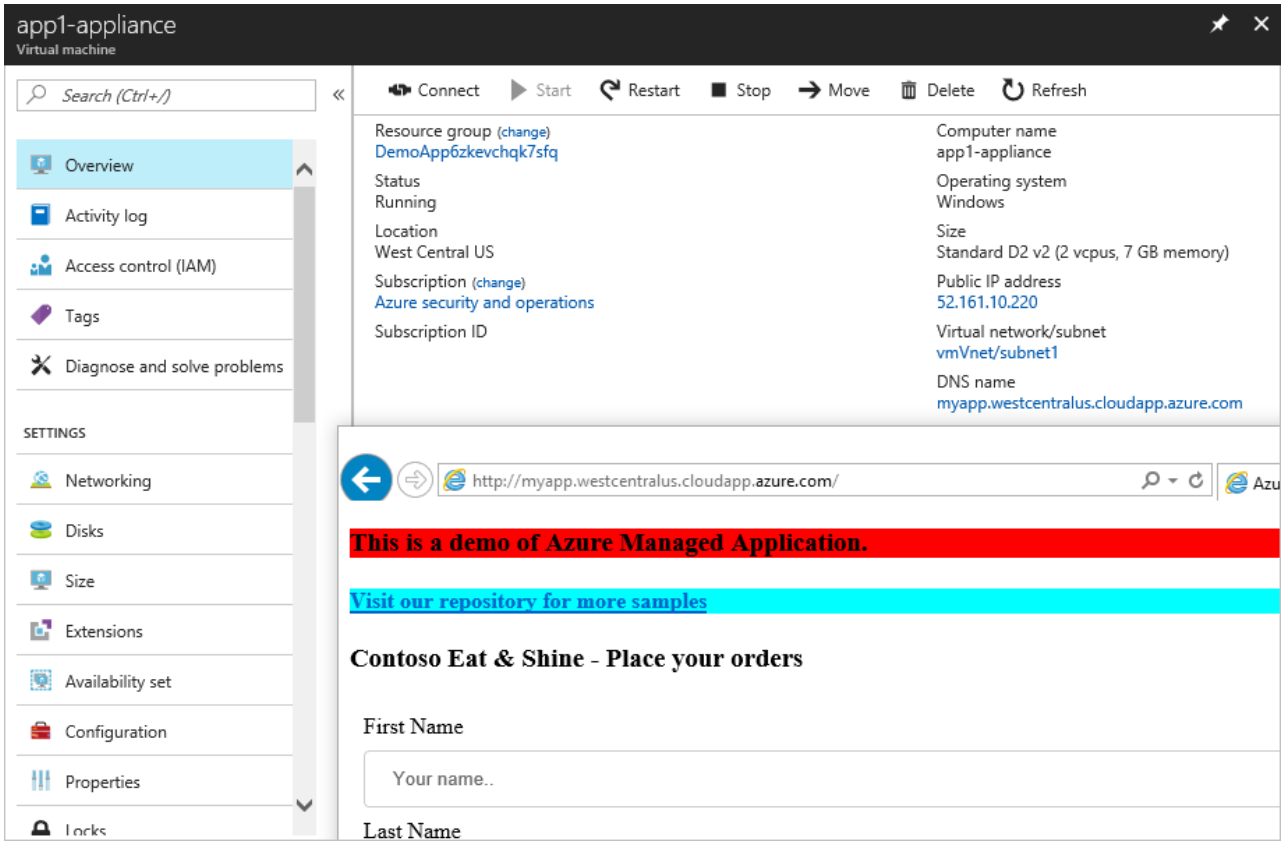
To see the virtual machines in the managed resource group, provide the name of the managed resource group.

```
az vm list -g DemoApp6zkevchqk7sfq --query "[].
{VMName:name,OSType:storageProfile.osDisk.osType,VMSize:hardwareProfile.vmSize}"
```

To update the size of the VMs, use:

```
az vm resize --size Standard_D2_v2 --ids $(az vm list -g DemoApp6zkevchqk7sfq --query "[].id" -o tsv)
```

After the operation completes, verify the application is running on Standard D2 v2.



Apply policy to managed resource group

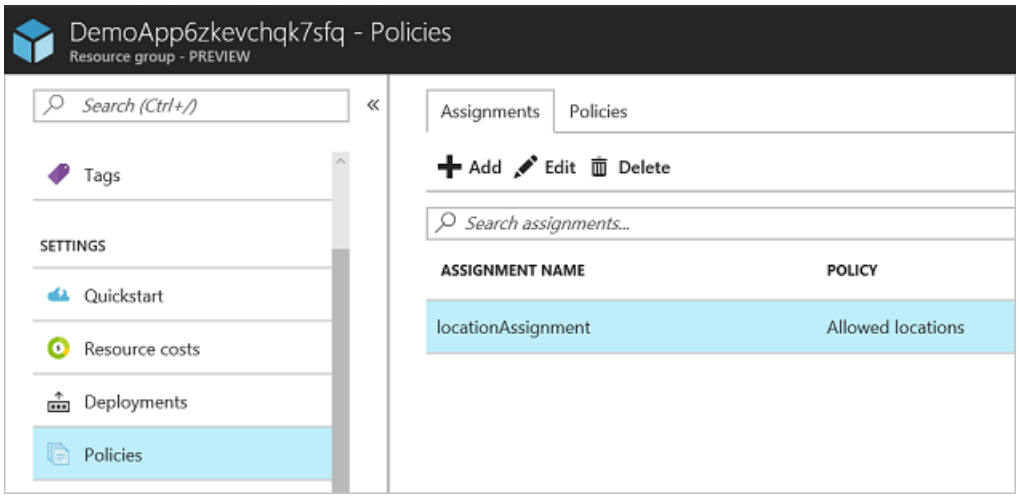
Get the managed resource group and assignment a policy at that scope. The policy **e56962a6-4747-49cd-b67b-bf8b01975c4c** is a built-in policy for specifying allowed locations.

```
managedGroup=$(az managedapp show --name <app-name> --resource-group DemoApp --query managedResourceGroupId --
output tsv)

az policy assignment create --name locationAssignment --policy e56962a6-4747-49cd-b67b-bf8b01975c4c --scope
$managedGroup --params '{
                            "listofallowedLocations": {
                                "value": [
                                    "northeurope",
                                    "westeurope"
                                ]
                            }
                        }'
```

To see the allowed locations, use:

```
az policy assignment show --name locationAssignment --scope $managedGroup --query
parameters.listofallowedLocations.value
```

The policy assignment appears in the portal.



Next steps

- For an introduction to managed applications, see Managed application overview.
- For sample projects, see Sample projects for Azure managed applications.

Publish a service catalog application through Azure portal

1/2/2020 • 2 minutes to read • Edit Online

You can use the Azure portal to publish managed applications that are intended for members of your organization. For example, an IT department can publish managed applications that ensure compliance with organizational standards. These managed applications are available through the service catalog, not the Azure marketplace.

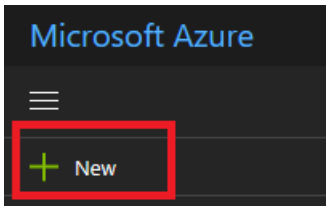
Prerequisites

When publishing a managed application, you specify an identity to manage the resources. We recommend you specify an Azure Active Directory user group. To create an Azure Active Directory user group, see Create a group and add members in Azure Active Directory.

The .zip file that contains the managed application definition must be available through a URI. We recommend that you upload your .zip file to a storage blob.

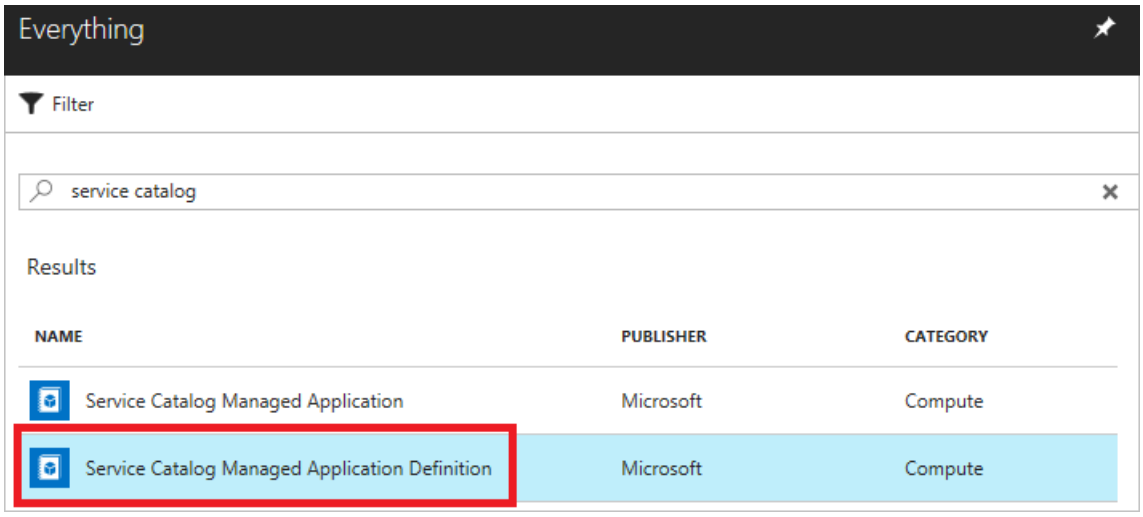
Create managed application with portal

1. In the upper left corner, select **+ New**.

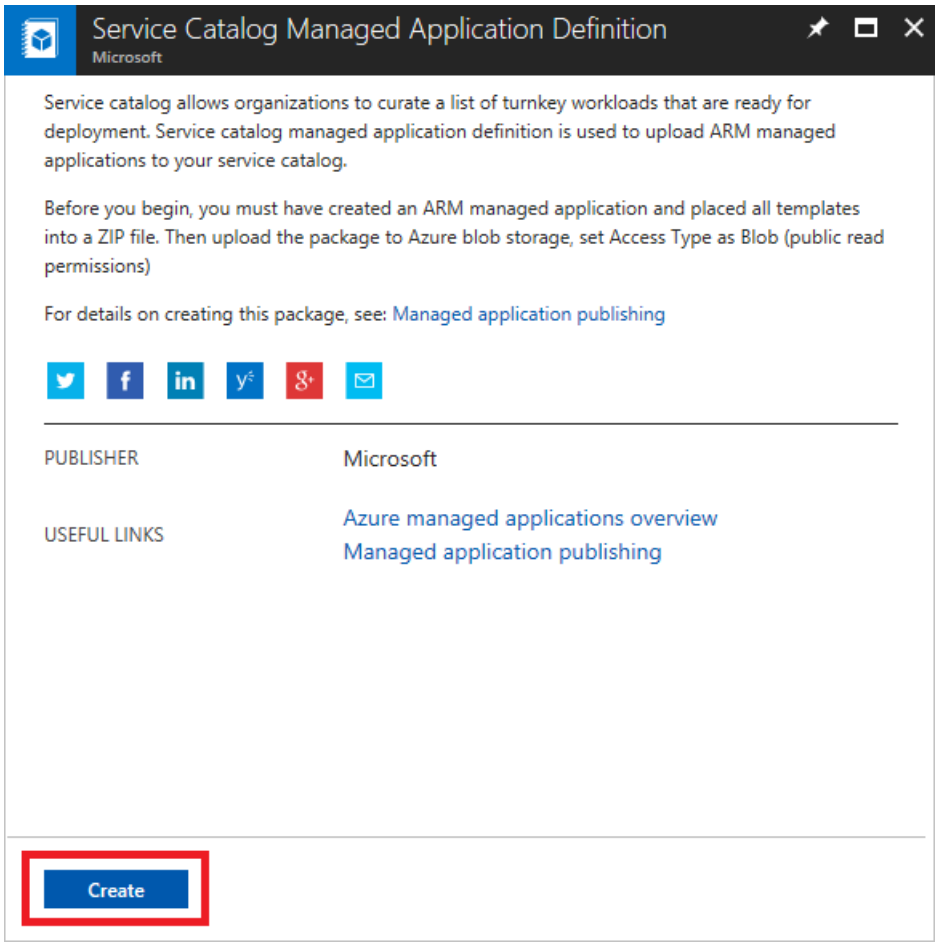


2. Search for **service catalog**.

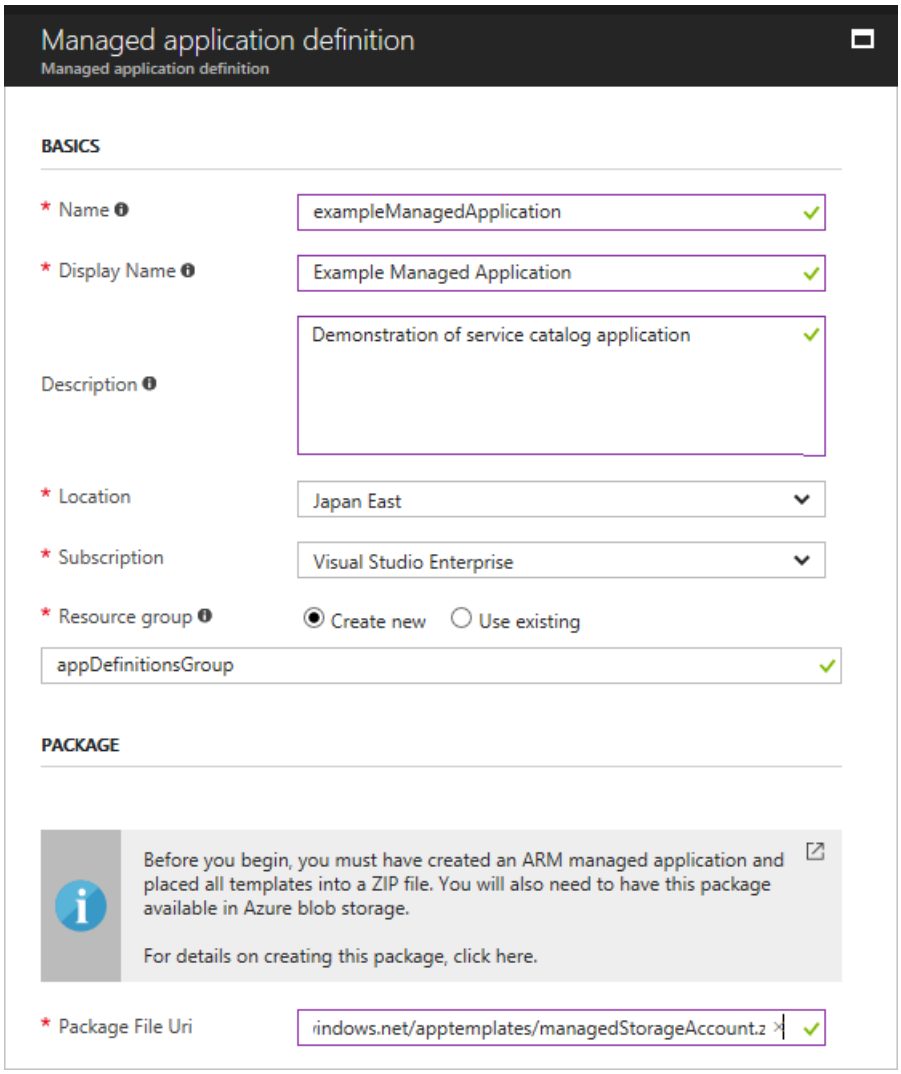
3. In the results, scroll until you find **Service Catalog Managed Application Definition**. Select it.



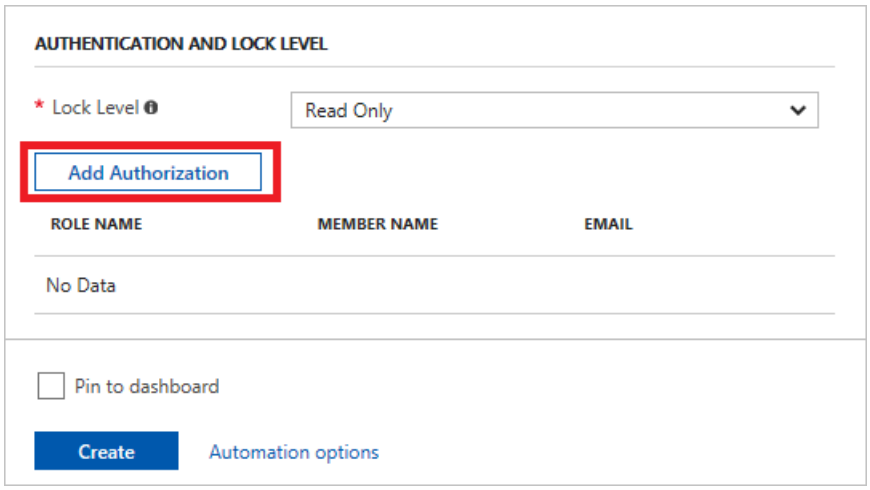
4. Select **Create** to start the process of creating the managed application definition.



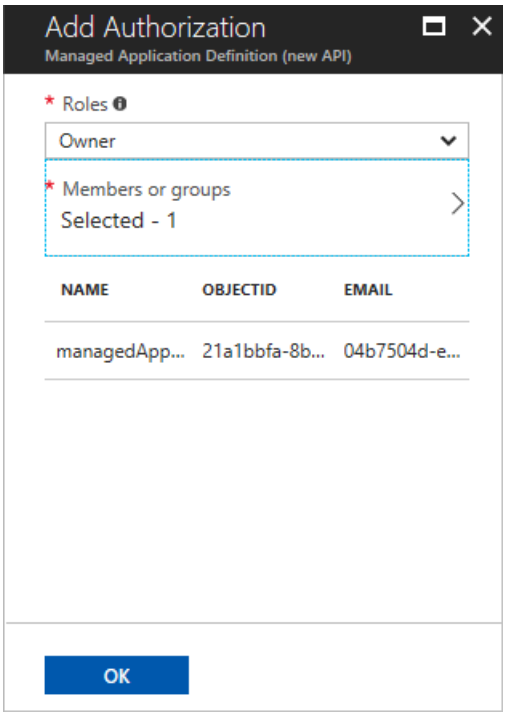
5. Provide values for name, display name, description, location, subscription, and resource group. For package file URI, provide the path to the zip file you created.



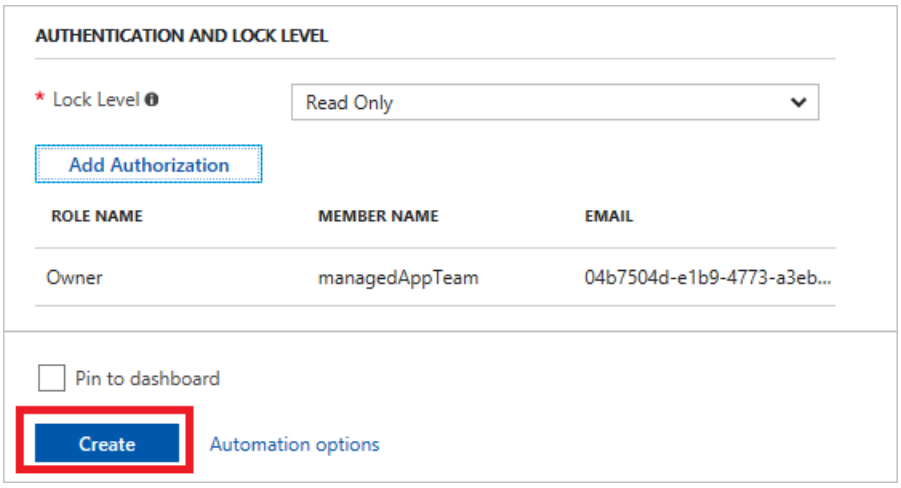
6. When you get to the Authentication and Lock Level section, select **Add Authorization**.



7. Select an Azure Active Directory group to manage the resources, and select **OK**.



8. When you have provided all the values, select **Create**.



Next steps

- For an introduction to managed applications, see Managed application overview.
- For managed application examples, see Sample projects for Azure managed applications.
- To learn how to create a UI definition file for a managed application, see Get started with CreateUiDefinition.

Access Key Vault secret when deploying Azure Managed Applications

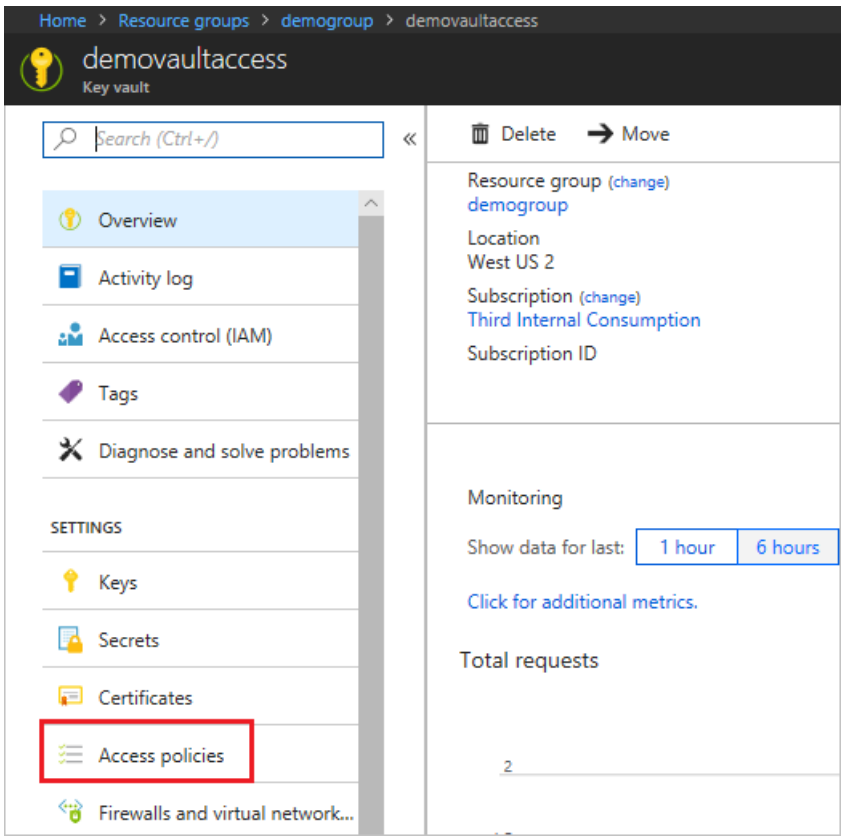
2/7/2020 • 2 minutes to read • Edit Online

When you need to pass a secure value (like a password) as a parameter during deployment, you can retrieve the value from an Azure Key Vault. To access the Key Vault when deploying Managed Applications, you must grant access to the **Appliance Resource Provider** service principal. The Managed Applications service uses this identity to run operations. To successfully retrieve a value from a Key Vault during deployment, the service principal must be able to access the Key Vault.

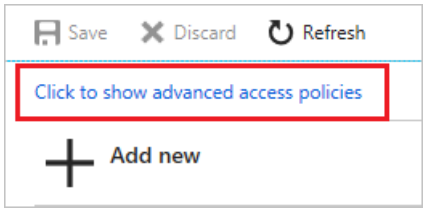
This article describes how to configure the Key Vault to work with Managed Applications.

Enable template deployment

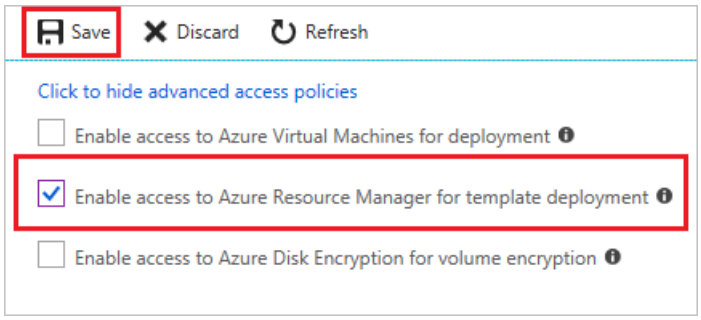
1. In the portal, select your Key Vault.
2. Select **Access policies**.



3. Select **Click to show advanced access policies**.

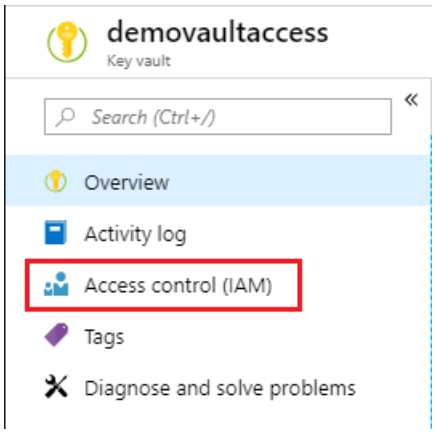


4. Select **Enable access to Azure Resource Manager for template deployment**. Then, select **Save**.

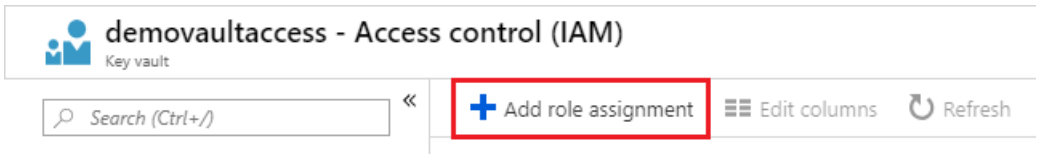


Add service as contributor

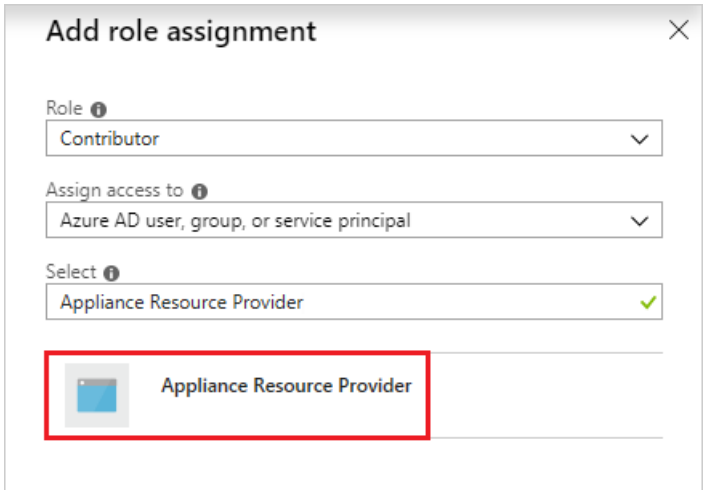
1. Select **Access control (IAM)**.



2. Select **Add role assignment**.



3. Select **Contributor** for the role. Search for **Appliance Resource Provider** and select it from the available options.



4. Select **Save**.

Reference Key Vault secret

To pass a secret from a Key Vault to a template in your Managed Application, you must use a linked or nested template and reference the Key Vault in the parameters for the linked or nested template. Provide the resource ID of the Key Vault and the name of the secret.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "The location where the resources will be deployed."
      }
    },
    "vaultName": {
      "type": "string",
      "metadata": {
        "description": "The name of the keyvault that contains the secret."
      }
    },
    "secretName": {
      "type": "string",
      "metadata": {
        "description": "The name of the secret."
      }
    },
    "vaultResourceGroupName": {
      "type": "string",
      "metadata": {
        "description": "The name of the resource group that contains the keyvault."
      }
    },
    "vaultSubscription": {
      "type": "string",
      "defaultValue": "[subscription().subscriptionId]",
      "metadata": {
        "description": "The name of the subscription that contains the keyvault."
      }
    }
  },
  "resources": [
    {
      "type": "Microsoft.Resources/deployments",
      "apiVersion": "2018-05-01",
      "name": "dynamicSecret",
      "properties": {
        "mode": "Incremental",
        "expressionEvaluationOptions": {
          "scope": "inner"
        },
        "template": {
          "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
          "contentVersion": "1.0.0.0",
          "parameters": {
            "adminLogin": {
              "type": "string"
            },
            "adminPassword": {
              "type": "securestring"
            },
            "location": {
              "type": "string"
            }
          },
          "variables": {
            "sqlServerName": "[concat('sql-', uniqueString(resourceGroup().id, 'sql'))]"
          },
          "resources": [
            {
              "type": "Microsoft.Sql/servers",
              "apiVersion": "2018-06-01-preview",
```

```
                "name": "[variables('sqlServerName')]",
                "location": "[parameters('location')]",
                "properties": {
                  "administratorLogin": "[parameters('adminLogin')]",
                  "administratorLoginPassword": "[parameters('adminPassword')]"
                }
              }
            ],
            "outputs": {
              "sqlFQDN": {
                "type": "string",
                "value": "[reference(variables('sqlServerName')).fullyQualifiedDomainName]"
              }
            }
          },
          "parameters": {
            "location": {
              "value": "[parameters('location')]"
            },
            "adminLogin": {
              "value": "ghuser"
            },
            "adminPassword": {
              "reference": {
                "keyVault": {
                  "id": "[resourceId(parameters('vaultSubscription'), parameters('vaultResourceGroupName'),
'Microsoft.KeyVault/vaults', parameters('vaultName'))]"
                },
                "secretName": "[parameters('secretName')]"
              }
            }
          }
        }
      }
    ],
    "outputs": {
    }
  }
```

Next steps

You've configured your Key Vault to be accessible during deployment of a Managed Application.

- For information about passing a value from a Key Vault as a template parameter, see Use Azure Key Vault to pass secure parameter value during deployment.
- For managed application examples, see Sample projects for Azure managed applications.
- To learn how to create a UI definition file for a managed application, see Get started with CreateUiDefinition.

Azure Managed Application with Managed Identity

1/2/2020 • 8 minutes to read • Edit Online

> **NOTE**
>
> Managed Identity support for Managed Applications is currently in preview. Please use the 2018-09-01-preview api version to utilize Managed Identity.

Learn how to configure a Managed Application to contain a Managed Identity. Managed Identity can be used to allow the customer to grant the Managed Application access to additional existing resources. The identity is managed by the Azure platform and does not require you to provision or rotate any secrets. For more about managed identities in Azure Active Directory (AAD), see Managed identities for Azure resources.

Your application can be granted two types of identities:

- A **system-assigned identity** is tied to your application and is deleted if your app is deleted. An app can only have one system-assigned identity.
- A **user-assigned identity** is a standalone Azure resource that can be assigned to your app. An app can have multiple user-assigned identities.

How to use Managed Identity

Managed Identity enables many scenarios for Managed Applications. Some common scenarios that can be solved are:

- Deploying a Managed Application linked to existing Azure resources. An example is deploying an Azure virtual machine (VM) within the Managed Application that is attached to an existing network interface.
- Granting the Managed Application and publisher access to Azure resources outside the **managed resource group**.
- Providing an operational identity of Managed Applications for Activity Log and other services within Azure.

Adding Managed Identity

Creating a Managed Application with a Managed Identity requires an additional property to be set on the Azure resource. The following example shows a sample **identity** property:

```
{
"identity": {
    "type": "SystemAssigned, UserAssigned",
    "userAssignedIdentities": {
        "/subscriptions/00000000-0000-0000-0000-
000000000000/resourceGroups/testRG/providers/Microsoft.ManagedIdentity/userassignedidentites/myuserassignedide
ntity": {}
    }
}
```

There are two common ways to create a Managed Application with **identity**: CreateUIDefinition.json and Azure Resource Manager templates. For simple single create scenarios, CreateUIDefinition should be used to enable Managed Identity, because it provides a richer experience. However, when dealing with advanced or complex systems that require automated or multiple Managed Application deployments, templates can be used.

Using CreateUIDefinition

A Managed Application can be configured with Managed Identity through the CreateUIDefinition.json. In the outputs section, the key `managedIdentity` can be used to override the identity property of the Managed Application template. The sample bellow will enable **system-assigned** identity on the Managed Application. More complex identity objects can be formed by using CreateUIDefinition elements to ask the consumer for inputs. These inputs can be used to construct Managed Applications with **user-assigned identity**.

```
"outputs": {
    "managedIdentity": "[parse('{\"Type\":\"SystemAssigned\"}')]"
}
```

When to use CreateUIDefinition for Managed Identity

Below are some recommendations on when to use CreateUIDefinition for enabling Managed Identity on Managed Applications.

- The Managed Application creation goes through the Azure portal or marketplace.
- The Managed Identity requires complex consumer input.
- The Managed Identity is needed on creation of the Managed Application.

SystemAssigned CreateUIDefinition

A basic CreateUIDefinition that enables the SystemAssigned identity for the Managed Application.

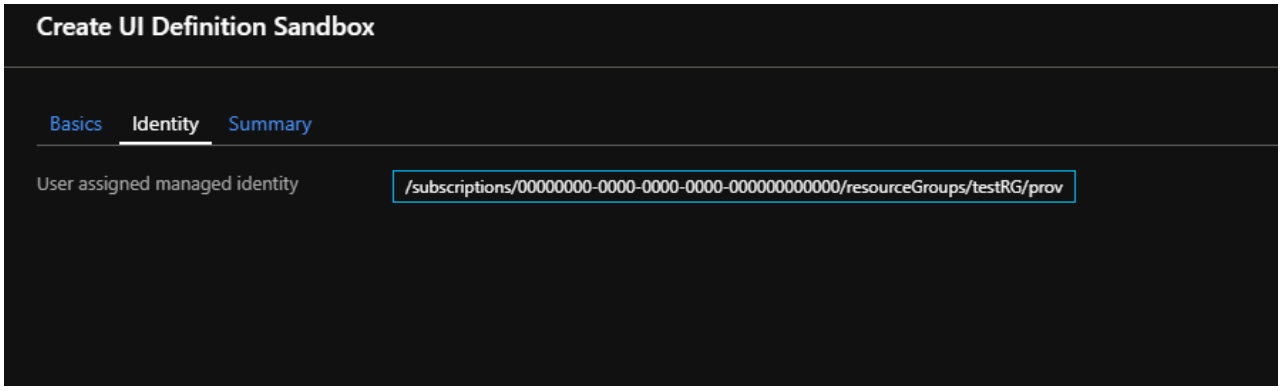
```
{
  "$schema": "https://schema.management.azure.com/schemas/0.1.2-preview/CreateUIDefinition.MultiVm.json#",
  "handler": "Microsoft.Azure.CreateUIDef",
  "version": "0.1.2-preview",
    "parameters": {
        "basics": [
            {}
        ],
        "steps": [
        ],
        "outputs": {
            "managedIdentity": "[parse('{\"Type\":\"SystemAssigned\"}')]"
        }
    }
}
```

UserAssigned CreateUIDefinition

A basic CreateUIDefinition that takes a **user-assigned identity** resource as input and enables the UserAssigned identity for the Managed Application.

```
{
  "$schema": "https://schema.management.azure.com/schemas/0.1.2-preview/CreateUIDefinition.MultiVm.json#",
  "handler": "Microsoft.Azure.CreateUIDef",
  "version": "0.1.2-preview",
    "parameters": {
        "basics": [
            {}
        ],
        "steps": [
            {
                "name": "manageIdentity",
                "label": "Identity",
                "subLabel": {
                    "preValidation": "Manage Identities",
                    "postValidation": "Done"
                },
                "bladeTitle": "Identity",
                "elements": [
                    {
                        "name": "userAssignedText",
                        "type": "Microsoft.Common.TextBox",
                        "label": "User assigned managed identity",
                        "defaultValue": "/subscriptions/00000000-0000-0000-0000-
000000000000/resourceGroups/testRG/providers/Microsoft.ManagedIdentity/userassignedidentites/myuserassignedide
ntity",
                        "visible": true
                    }
                ]
            }
        ],
        "outputs": {
            "managedIdentity": "[parse(concat('{\"Type\":\"UserAssigned\",\"UserAssignedIdentities\":
{',string(steps('manageIdentity').userAssignedText),':{}}}'))]"
        }
    }
}
```

The CreateUIDefinition.json above generates a create user experience that has a textbox for a consumer to enter the **user-assigned identity** Azure resource ID. The generated experience would look like:



Using Azure Resource Manager templates

> **NOTE**
>
> Marketplace Managed Application templates are automatically generated for customers going through the Azure portal create experience. For these scenarios, the `managedIdentity` output key on the CreateUIDefinition must be used to enabled identity.

The Managed Identity can also be enabled through Azure Resource Manager templates. The sample bellow will enable **system-assigned** identity on the Managed Application. More complex identity objects can be formed by

using Azure Resource Manager template parameters to provide inputs. These inputs can be used to construct Managed Applications with **user-assigned identity**.

When to use Azure Resource Manager templates for Managed Identity

Below are some recommendations on when to use Azure Resource Manager templates for enabling Managed Identity on Managed Applications.

- Managed Applications can be programmatically deployed based on a template.
- Custom role assignments for the Managed Identity are needed to provision the Managed Application.
- The Managed Application does not need the Azure portal and marketplace creation flow.

SystemAssigned template

A basic Azure Resource Manager template that deploys a Managed Application with **system-assigned** identity.

```
"resources": [
    {
        "type": "Microsoft.Solutions/applications",
        "name": "[parameters('applicationName')]",
        "apiVersion": "2018-09-01-preview",
        "location": "[parameters('location')]",
        "identity": {
            "type": "SystemAssigned"
        },
        "properties": {
            "ManagedResourceGroupId": "[parameters('managedByResourceGroupId')]",
            "parameters": { }
        }
    }
]
```

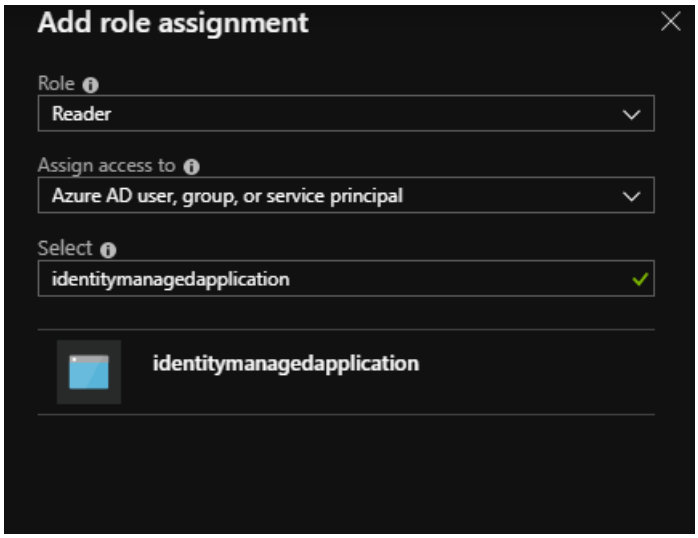
UserAssigned template

A basic Azure Resource Manager template that deploys a Managed Application with a **user-assigned identity**.

```
"resources": [
    {
      "type": "Microsoft.ManagedIdentity/userAssignedIdentities",
      "name": "[parameters('managedIdentityName')]",
      "apiVersion": "2018-11-30",
      "location": "[parameters('location')]"
    },
    {
        "type": "Microsoft.Solutions/applications",
        "name": "[parameters('applicationName')]",
        "apiVersion": "2018-09-01-preview",
        "location": "[parameters('location')]",
        "identity": {
            "type": "UserAssigned",
            "userAssignedIdentities": {
                "
[resourceID('Microsoft.ManagedIdentity/userAssignedIdentities/',parameters('managedIdentityName'))]": {}
            }
        },
        "properties": {
            "ManagedResourceGroupId": "[parameters('managedByResourceGroupId')]",
            "parameters": { }
        }
    }
]
```

Granting access to Azure resources

Once a Managed Application is granted an identity, it can be granted access to existing azure resources. This process can be done through the Access control (IAM) interface in the Azure portal. The name of the Managed Application or **user-assigned identity** can be searched to add a role assignment.



Linking existing Azure resources

> **NOTE**
>
> A **user-assigned identity** must be configured before deploying the Managed Application. In addition, linked resource deployment of Managed Applications is only supported for the **marketplace** kind.

Managed Identity can also be used to deploy a Managed Application that requires access to existing resources during its deployment. When the Managed Application is provisioned by the customer, **user-assigned identities** can be added to provide additional authorizations to the **mainTemplate** deployment.

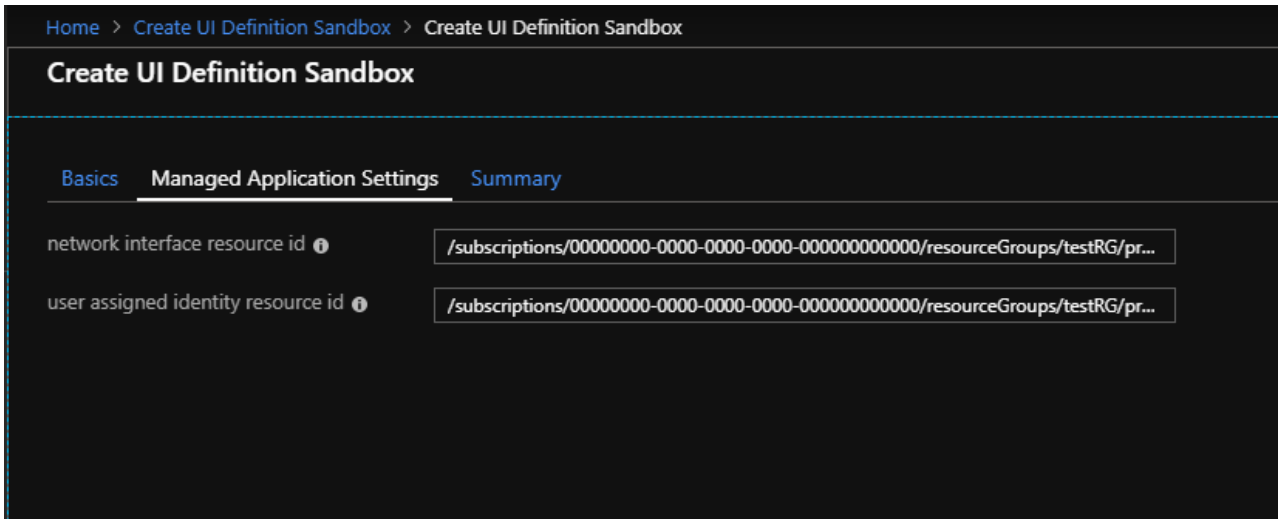
Authoring the CreateUIDefinition with a linked resource

When linking the deployment of the Managed Application to existing resources, both the existing Azure resource and a **user-assigned identity** with the applicable role assignment on that resource must be provided.

A sample CreateUIDefinition that requires two inputs: a network interface resource ID and a user assigned identity resource id.

```
{
    "$schema": "https://schema.management.azure.com/schemas/0.1.2-preview/CreateUIDefinition.MultiVm.json#",
    "handler": "Microsoft.Compute.MultiVm",
    "version": "0.1.2-preview",
    "parameters": {
        "basics": [
            {}
        ],
        "steps": [
            {
                "name": "managedApplicationSetting",
                "label": "Managed Application Settings",
                "subLabel": {
                    "preValidation": "Managed Application Settings",
                    "postValidation": "Done"
                },
                "bladeTitle": "Managed Application Settings",
                "elements": [
                    {
                        "name": "networkInterfaceId",
                        "type": "Microsoft.Common.TextBox",
                        "label": "network interface resource id",
                        "defaultValue": "/subscriptions/00000000-0000-0000-0000-
000000000000/resourceGroups/testRG/providers/Microsoft.Network/networkInterfaces/existingnetworkinterface",
                        "toolTip": "Must represent the identity as an Azure Resource Manager resource
identifer format ex.
/subscriptions/sub1/resourcegroups/myGroup/providers/Microsoft.Network/networkInterfaces/networkinterface1",
                        "visible": true
                    },
                    {
                        "name": "userAssignedId",
                        "type": "Microsoft.Common.TextBox",
                        "label": "user assigned identity resource id",
                        "defaultValue": "/subscriptions/00000000-0000-0000-0000-
000000000000/resourceGroups/testRG/providers/Microsoft.ManagedIdentity/userassignedidentites/myuserassignedide
ntity",
                        "toolTip": "Must represent the identity as an Azure Resource Manager resource
identifer format ex.
/subscriptions/sub1/resourcegroups/myGroup/providers/Microsoft.ManagedIdentity/userAssignedIdentities/identity
1",
                        "visible": true
                    }
                ]
            }
        ],
        "outputs": {
            "existingNetworkInterfaceId": "[steps('managedApplicationSetting').networkInterfaceId]",
            "managedIdentity": "[parse(concat('{\"Type\":\"UserAssigned\",\"UserAssignedIdentities\":
{',string(steps('managedApplicationSetting').userAssignedId),':{}}}'))]"
        }
    }
}
```

This CreateUIDefinition.json generates a create user experience that has two fields. The first field allows the user to enter in the Azure resource ID for the resource being linked to the Managed Application deployment. The second is for a consumer to enter the **user-assigned identity** Azure resource ID, which has access to the linked Azure resource. The generated experience would look like:



Authoring the mainTemplate with a linked resource

In addition to updating the CreateUIDefinition, the main template also needs to be updated to accept the passed in linked resource ID. The main template can be updated to accept the new output by adding a new parameter. Since the `managedIdentity` output overrides the value on the generated Managed Application template, it is not passed to the main template and should not be included in the parameters section.

A sample main template that sets the network profile to an existing network interface provided by the CreateUIDefinition.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
      "existingNetworkInterfaceId": { "type": "string" }
    },
    "variables": {
    },
    "resources": [
        {
            "apiVersion": "2016-04-30-preview",
            "type": "Microsoft.Compute/virtualMachines",
            "name": "myLinkedResourceVM",
            "location": "[resourceGroup().location]",
            "properties": {
                …,
                "networkProfile": {
                    "networkInterfaces": [
                        {
                            "id": "[parameters('existingNetworkInterfaceId')]"
                        }
                    ]
                }
            }
        }
    ]
}
```

Consuming the Managed Application with a linked resource

Once the Managed Application package is created, the Managed Application can be consumed through the Azure portal. Before it can be consumed, there are several prerequisite steps.

- An instance of the required linked Azure resource must be created.
- The **user-assigned identity** must be created and given role assignments to the linked resource.
- The existing linked resource ID and the **user-assigned identity** ID are provided to the CreateUIDefinition.

Accessing the Managed Identity token

The token of the Managed Application can now be accessed through the `listTokens` api from the publisher tenant. An example request might look like:

```
POST
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroup}/providers/Microsoft
.Solutions/applications/{applicationName}/listTokens?api-version=2018-09-01-preview HTTP/1.1

{
    "authorizationAudience": "https://management.azure.com/",
    "userAssignedIdentities": [

"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroup}/providers/Microsoft.ManagedIdentity/userAssign
edIdentities/{userAssignedIdentityName}"
    ]
}
```

Request Body Parameters:

PARAMETER	REQUIRED	DESCRIPTION
authorizationAudience	*no*	The App ID URI of the target resource. It also is the `aud` (audience) claim of the issued token. The default value is "https://management.azure.com/"
userAssignedIdentities	*no*	The list of user-assigned managed identities to retrieve a token for. If not specified, `listTokens` will return the token for the system-assigned managed identity.

A sample response might look like:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "value": [
        {
            "access_token": "eyJ0eXAi…",
            "expires_in": "2…",
            "expires_on": "1557…",
            "not_before": "1557…",
            "authorizationAudience": "https://management.azure.com/",
            "resourceId":
"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroup}/providers/Microsoft.Solutions/applications/{ap
plicationName}",
            "token_type": "Bearer"
        }
    ]
}
```

The response will contain an array of tokens under the `value` property:

PARAMETER	DESCRIPTION
access_token	The requested access token.

PARAMETER	DESCRIPTION
expires_in	The number of seconds the access token will be valid.
expires_on	The timespan when the access token expires. This is represented as the number of seconds from epoch.
not_before	The timespan when the access token takes effect. This is represented as the number of seconds from epoch.
authorizationAudience	The `aud` (audience) the access token was request for. This is the same as what was provided in the `listTokens` request.
resourceId	The Azure resource ID for the issued token. This is either the managed application ID or the user-assigned identity ID.
token_type	The type of the token.

Next steps

How to configure a Managed Application with a Custom Provider

Azure managed applications with notifications

1/23/2020 • 4 minutes to read • Edit Online

Azure managed application notifications allow publishers to automate actions based on lifecycle events of the managed application instances. Publishers can specify custom notification webhook endpoints to receive event notifications about new and existing managed application instances. Publishers can set up custom workflows at the time of application provisioning, updates, and deletion.

Getting started

To start receiving managed applications, spin up a public HTTPS endpoint and specify it when you publish the service catalog application definition or Azure Marketplace offer.

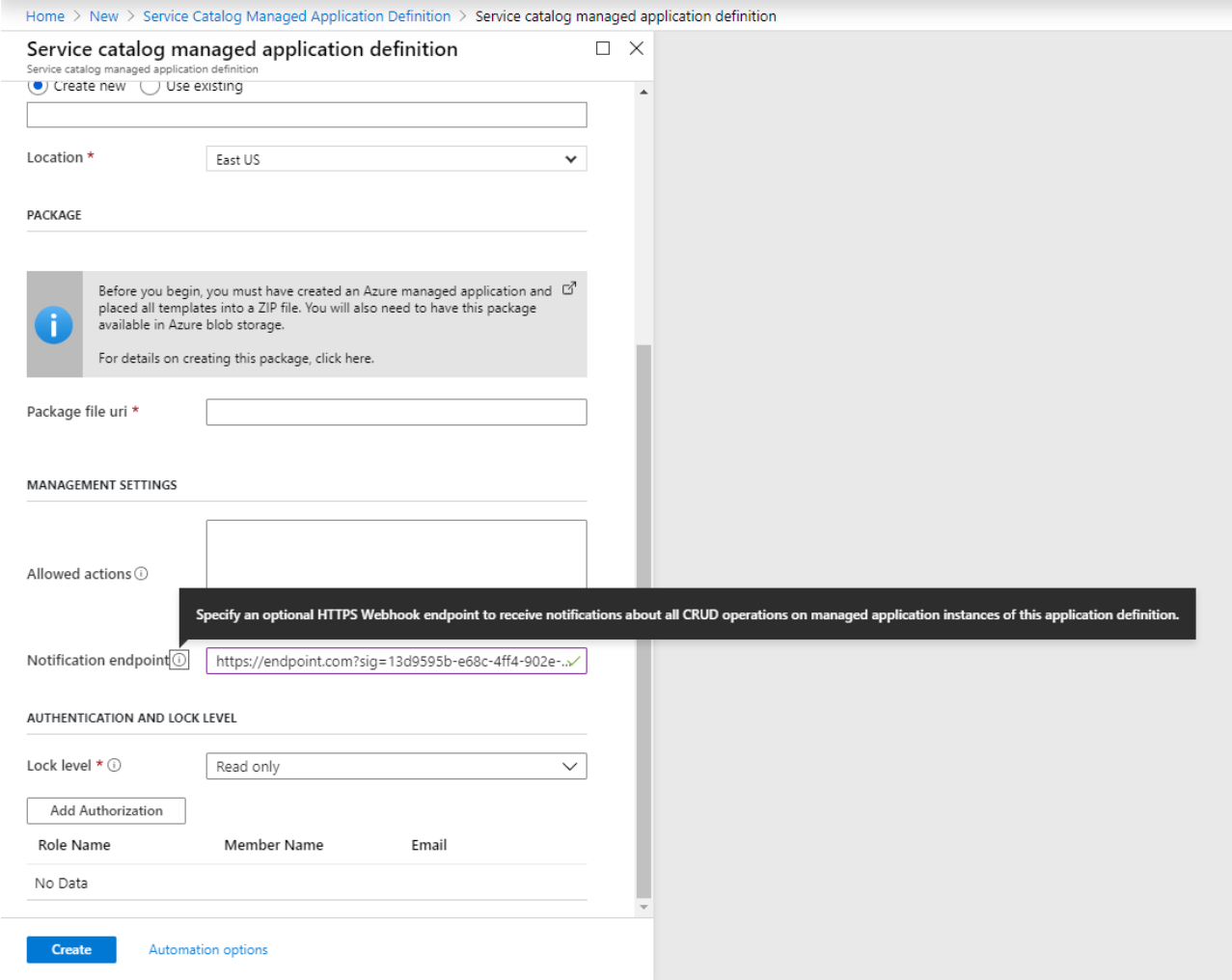
Here are the recommended steps to get started quickly:

1. Spin up a public HTTPS endpoint that logs the incoming POST requests and returns `200 OK`.
2. Add the endpoint to the service catalog application definition or Azure Marketplace offer as explained later in this article.
3. Create a managed application instance that references the application definition or Azure Marketplace offer.
4. Validate that the notifications are being received.
5. Enable authorization as explained in the **Endpoint authentication** section of this article.
6. Follow the instructions in the **Notification schema** section of this article to parse the notification requests and implement your business logic based on the notification.

Add service catalog application definition notifications

Azure portal

To get started, see Publish a service catalog application through Azure portal.



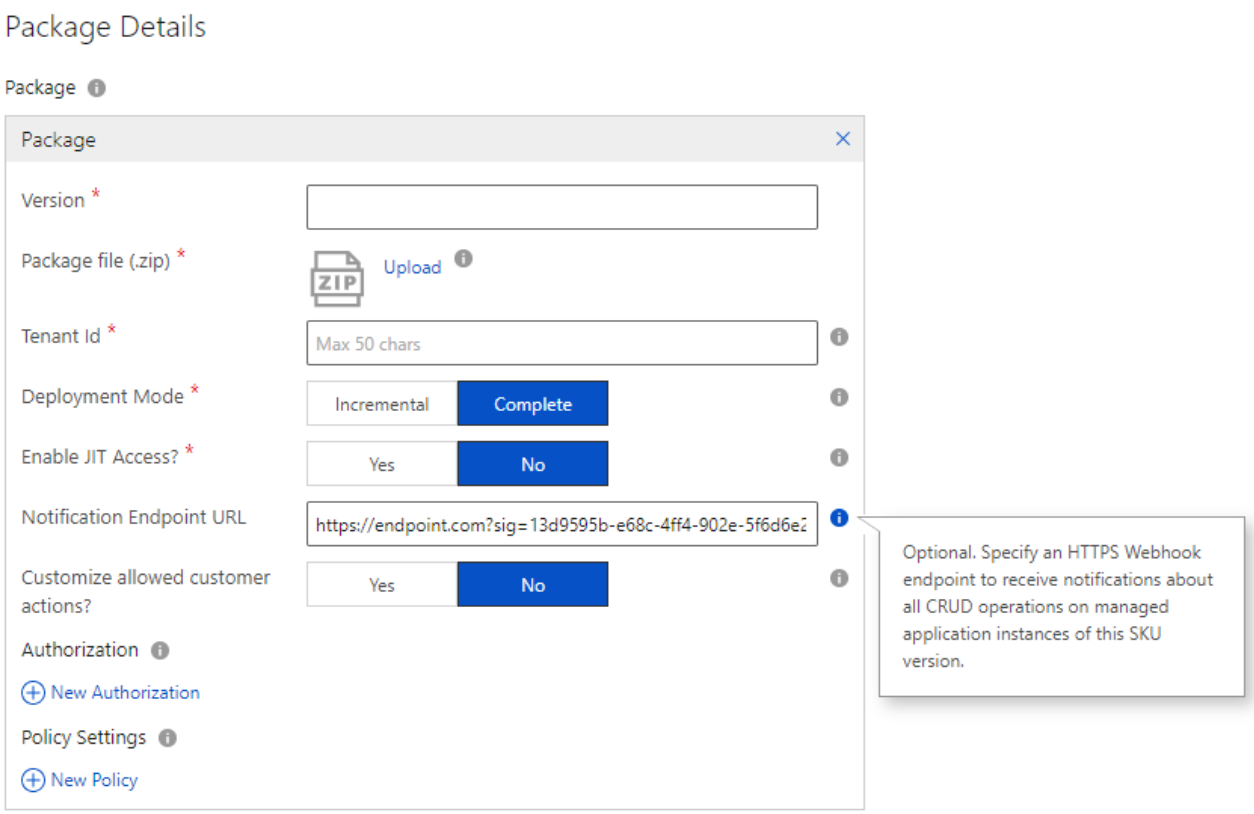
REST API

> **NOTE**
>
> Currently, you can supply only one endpoint in the `notificationEndpoints` in the application definition properties.

```
{
  "properties": {
    "isEnabled": true,
    "lockLevel": "ReadOnly",
    "displayName": "Sample Application Definition",
    "description": "Notification-enabled application definition.",
    "notificationPolicy": {
        "notificationEndpoints": [
            {
                "uri": "https://isv.azurewebsites.net:1214?sig=unique_token"
            }
        ]
    },
    "authorizations": [
      {
        "principalId": "d6b7fbd3-4d99-43fe-8a7a-f13aef11dc18",
        "roleDefinitionId": "8e3af657-a8ff-443c-a75c-2fe8c4bcb635"
      },
    ...
```

Add Azure Marketplace managed application notifications

For more information, see Create an Azure application offer.



Event triggers

The following table describes all the possible combinations of EventType and ProvisioningState and their triggers:

EVENTTYPE	PROVISIONINGSTATE	TRIGGER FOR NOTIFICATION
PUT	Accepted	Managed resource group has been created and projected successfully after application PUT (before the deployment inside the managed resource group is kicked off).
PUT	Succeeded	Full provisioning of the managed application succeeded after a PUT.
PUT	Failed	Failure of PUT of application instance provisioning at any point.
PATCH	Succeeded	After a successful PATCH on the managed application instance to update tags, JIT access policy, or managed identity.
DELETE	Deleting	As soon as the user initiates a DELETE of a managed app instance.
DELETE	Deleted	After the full and successful deletion of the managed application.
DELETE	Failed	After any error during the deprovisioning process that blocks the deletion.

Notification schema

When you spin up your webhook endpoint to handle notifications, you'll need to parse the payload to get important properties to then act upon the notification. Service catalog and Azure Marketplace managed application notifications provide many of the same properties. Two small differences are outlined in the table that follows the samples.

Service catalog application notification schema

Here's a sample service catalog notification after the successful provisioning of a managed application instance:

```
POST https://{your_endpoint_URI}/resource?{optional_parameter}={optional_parameter_value} HTTP/1.1

{
    "eventType": "PUT",
    "applicationId":
"subscriptions/<subId>/resourceGroups/<rgName>/providers/Microsoft.Solutions/applications/<applicationName>",
    "eventTime": "2019-08-14T19:20:08.1707163Z",
    "provisioningState": "Succeeded",
    "applicationDefinitionId":
"subscriptions/<subId>/resourceGroups/<rgName>/providers/Microsoft.Solutions/applicationDefinitions/<appDefName
>"
}
```

If the provisioning fails, a notification with the error details will be sent to the specified endpoint.

```
POST https://{your_endpoint_URI}/resource?{optional_parameter}={optional_parameter_value} HTTP/1.1

{
    "eventType": "PUT",
    "applicationId":
"subscriptions/<subId>/resourceGroups/<rgName>/providers/Microsoft.Solutions/applications/<applicationName>",
    "eventTime": "2019-08-14T19:20:08.1707163Z",
    "provisioningState": "Failed",
    "applicationDefinitionId":
"subscriptions/<subId>/resourceGroups/<rgName>/providers/Microsoft.Solutions/applicationDefinitions/<appDefName
>",
    "error": {
        "code": "ErrorCode",
        "message": "error message",
        "details": [
            {
                "code": "DetailedErrorCode",
                "message": "error message"
            }
        ]
    }
}
```

Azure Marketplace application notification schema

Here's a sample service catalog notification after the successful provisioning of a managed application instance:

```
POST https://{your_endpoint_URI}/resource?{optional_parameter}={optional_parameter_value} HTTP/1.1

{
    "eventType": "PUT",
    "applicationId":
"subscriptions/<subId>/resourceGroups/<rgName>/providers/Microsoft.Solutions/applications/<applicationName>",
    "eventTime": "2019-08-14T19:20:08.1707163Z",
    "provisioningState": "Succeeded",
    "billingDetails": {
        "resourceUsageId":"<resourceUsageId>"
    },
    "plan": {
        "publisher": "publisherId",
        "product": "offer",
        "name": "skuName",
        "version": "1.0.1"
    }
}
```

If the provisioning fails, a notification with the error details will be sent to the specified endpoint.

```
POST https://{your_endpoint_URI}/resource?{optional_parameter}={optional_parameter_value} HTTP/1.1

{
    "eventType": "PUT",
    "applicationId":
"subscriptions/<subId>/resourceGroups/<rgName>/providers/Microsoft.Solutions/applications/<applicationName>",
    "eventTime": "2019-08-14T19:20:08.1707163Z",
    "provisioningState": "Failed",
    "billingDetails": {
        "resourceUsageId":"<resourceUsageId>"
    },
    "plan": {
        "publisher": "publisherId",
        "product": "offer",
        "name": "skuName",
        "version": "1.0.1"
    },
    "error": {
        "code": "ErrorCode",
        "message": "error message",
        "details": [
            {
                "code": "DetailedErrorCode",
                "message": "error message"
            }
        ]
    }
}
```

PARAMETER	DESCRIPTION
eventType	The type of event that triggered the notification. (For example, PUT, PATCH, DELETE.)
applicationId	The fully qualified resource identifier of the managed application for which the notification was triggered.
eventTime	The timestamp of the event that triggered the notification. (Date and time in UTC ISO 8601 format.)

PARAMETER	DESCRIPTION
provisioningState	The provisioning state of the managed application instance. (For example, Succeeded, Failed, Deleting, Deleted.)
error	*Specified only when the provisioningState is Failed*. Contains the error code, message, and details of the issue that caused the failure.
applicationDefinitionId	*Specified only for service catalog managed applications*. Represents the fully qualified resource identifier of the application definition for which the managed application instance was provisioned.
plan	*Specified only for Azure Marketplace managed applications*. Represents the publisher, offer, SKU, and version of the managed application instance.
billingDetails	*Specified only for Azure Marketplace managed applications*. The billing details of the managed application instance. Contains the resourceUsageId that you can use to query Azure Marketplace for usage details.

Endpoint authentication

To secure the webhook endpoint and ensure the authenticity of the notification:

1. Provide a query parameter on top of the webhook URI, like this: https://your-endpoint.com?sig=Guid. With each notification, check that the query parameter `sig` has the expected value `Guid`.
2. Issue a GET on the managed application instance by using applicationId. Validate that the provisioningState matches the provisioningState of the notification to ensure consistency.

Notification retries

The Managed Application Notification service expects a `200 OK` response from the webhook endpoint to the notification. The notification service will retry if the webhook endpoint returns an HTTP error code greater than or equal to 500, if it returns an error code of 429, or if the endpoint is temporarily unreachable. If the webhook endpoint doesn't become available within 10 hours, the notification message will be dropped and the retries will stop.

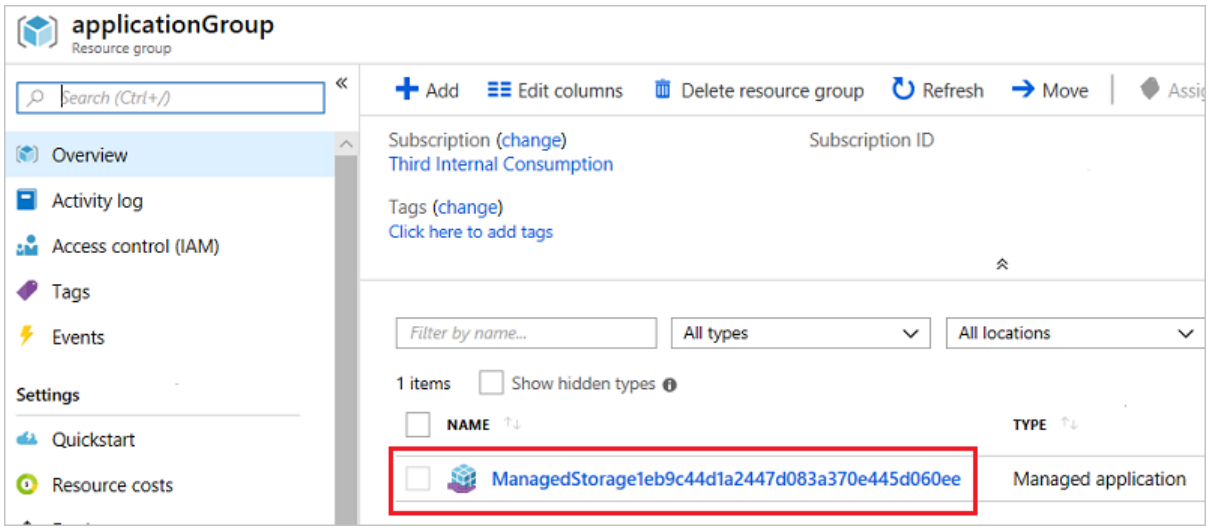
Monitor a deployed instance of a managed application

1/2/2020 • 2 minutes to read • Edit Online

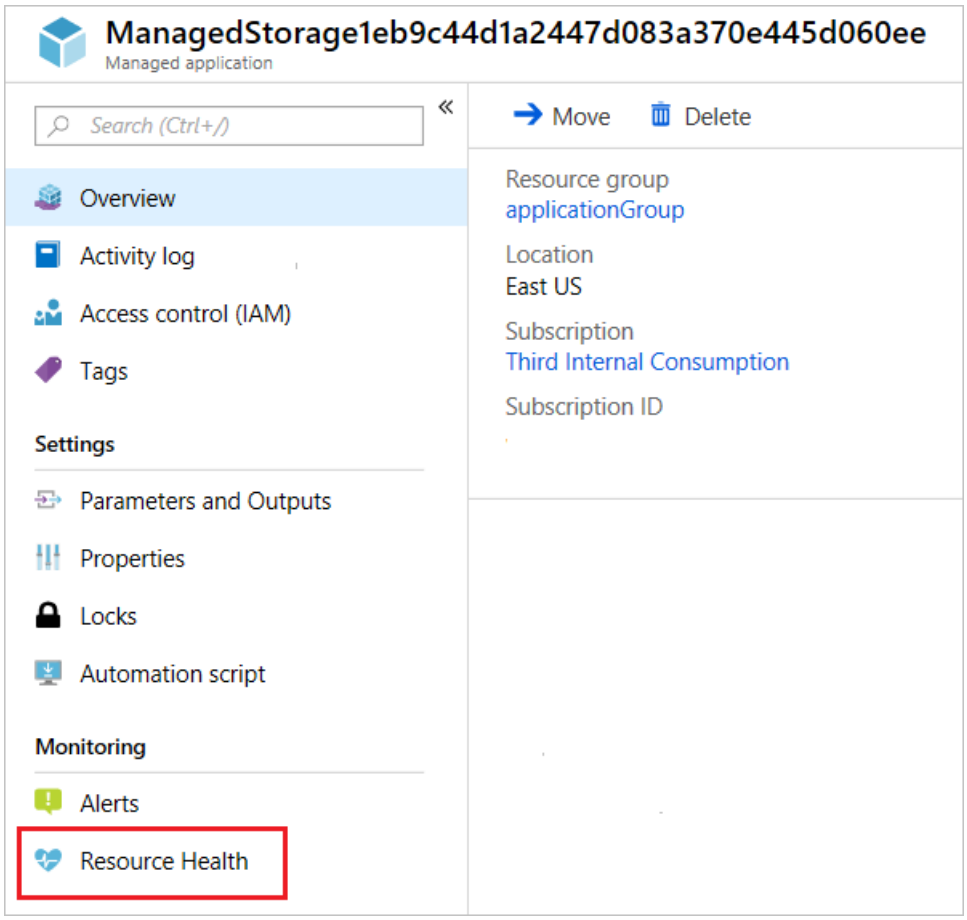
After you've deployed a managed application to your Azure subscription, you may want to check the status of the application. This article shows options in the Azure portal for checking the status. You can monitor the availability of the resources in your managed application. You can also set up and view alerts.

View resource health

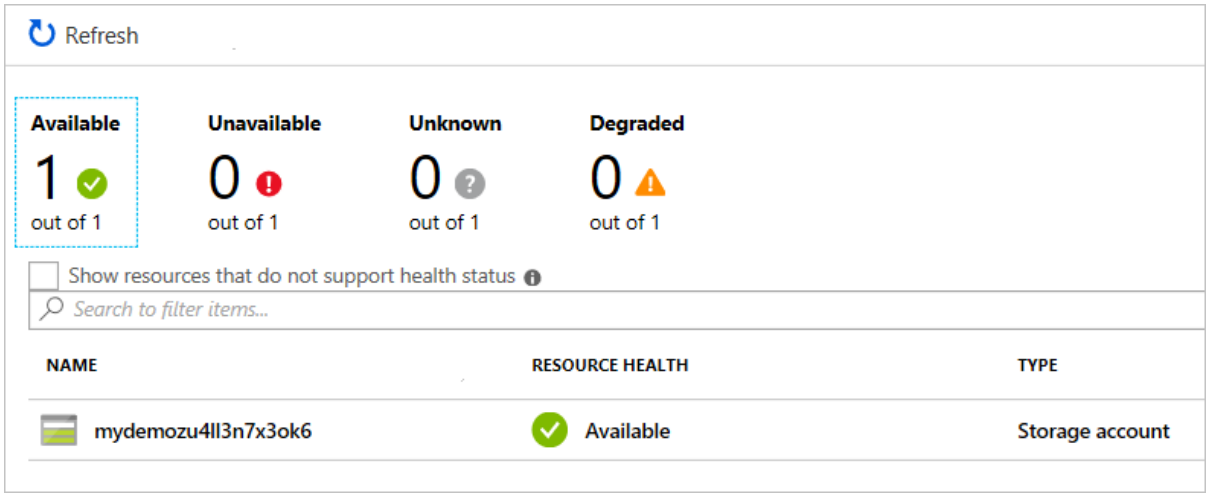
1. Select your managed application instance.



2. Select **Resource Health**.

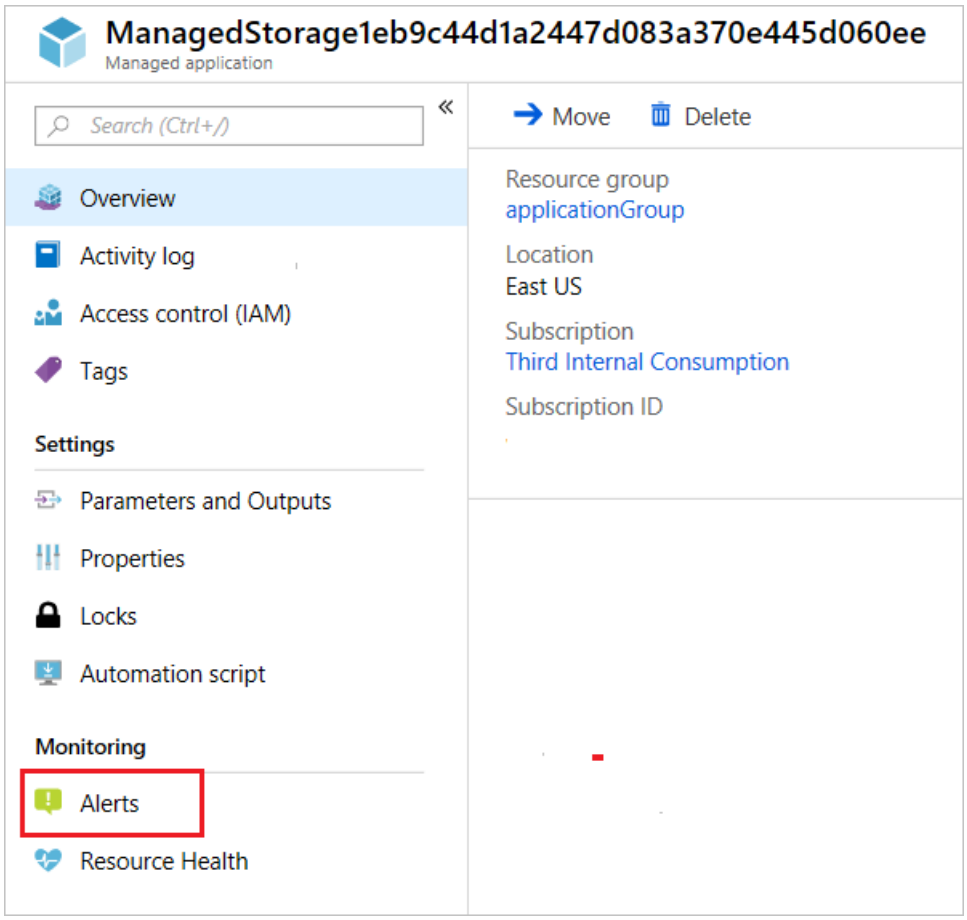


3. View the availability of the resources in your managed application.

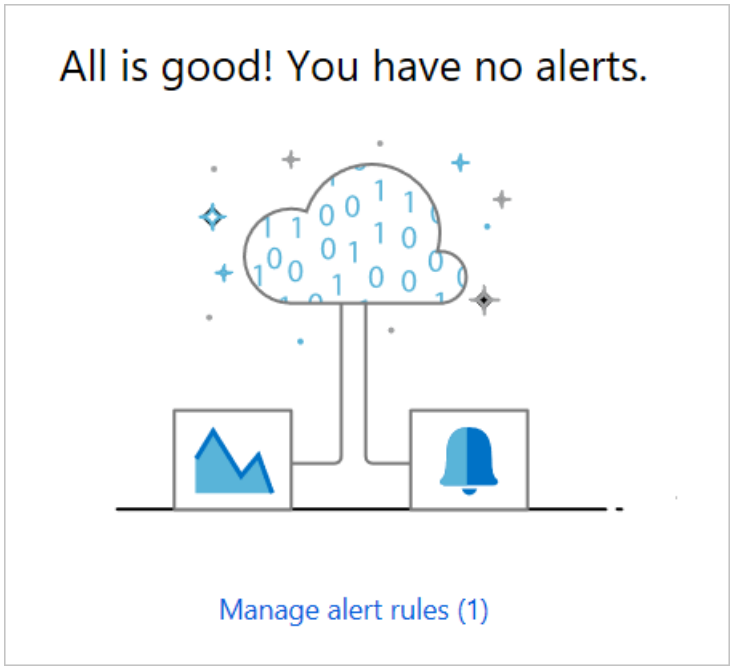


View alerts

1. Select **Alerts**.



2. If you have alert rules configured, you see information about alerts that have been raised.



3. To add alert rules, select **+ New alert rule**.

You can create alerts for your managed application instance or the resources in the managed application. For information about creating alerts, see Overview of alerts in Microsoft Azure.

Next steps

- For managed application examples, see Sample projects for Azure managed applications.
- To deploy a managed application, see Deploy service catalog app through Azure portal.

Configure and approve just-in-time access for Azure Managed Applications

1/2/2020 • 3 minutes to read • Edit Online

As a consumer of a managed application, you might not be comfortable giving the publisher permanent access to the managed resource group. To give you greater control over granting access to managed resources, Azure Managed Applications provides a feature called just-in-time (JIT) access, which is currently in preview. It enables you to approve when and for how long the publisher has access to the resource group. The publisher can make required updates during that time, but when that time is over, the publisher's access expires.

The work flow for granting access is:

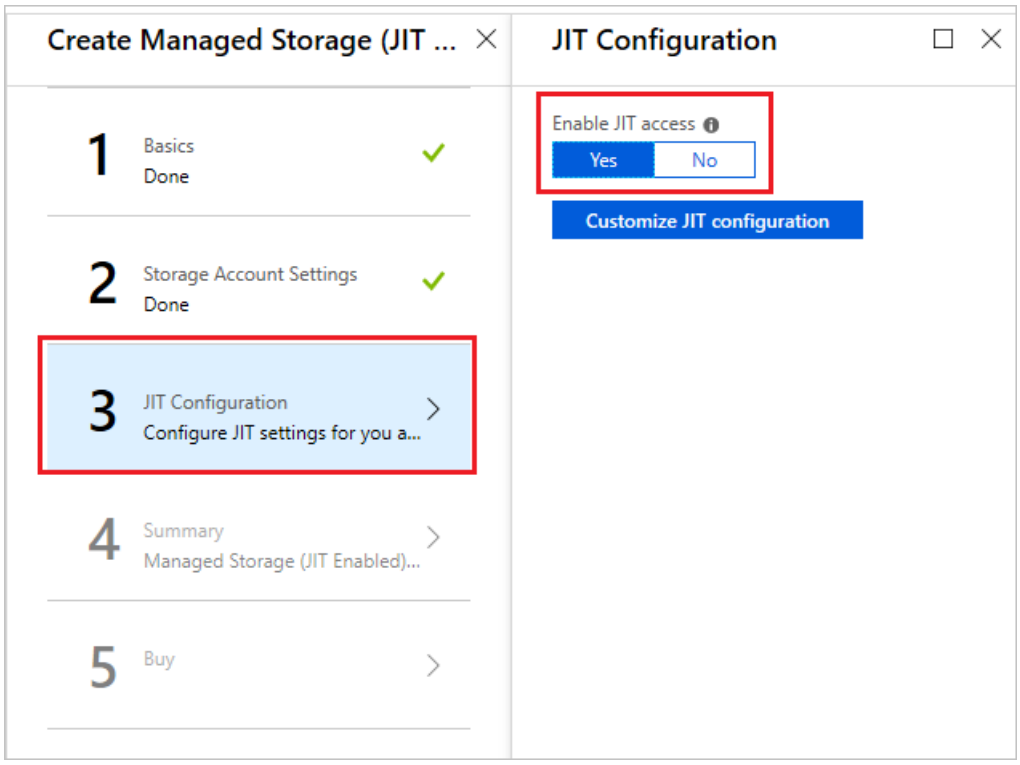
1. The publisher adds a managed application to the marketplace and specifies that JIT access is available.
2. During deployment, you enable JIT access for your instance of the managed application.
3. After deployment, you can change the settings for JIT access.
4. The publisher sends a request for access.
5. You approve the request.

This article focuses on the actions consumers take to enable JIT access and approve requests. To learn about publishing a managed application with JIT access, see Request just-in-time access in Azure Managed Applications.

> **NOTE**
>
> To use just-in-time access, you must have a Azure Active Directory P2 license.

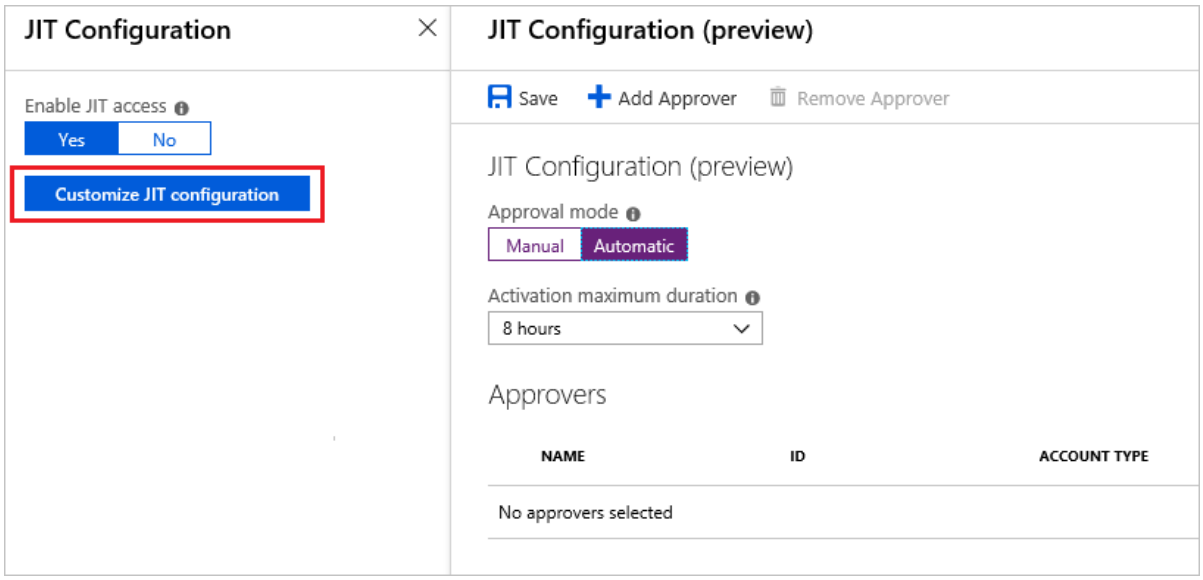
Enable during deployment

1. Sign in to the Azure portal.
2. Find a marketplace entry for a managed application with JIT enabled. Select **Create**.
3. While providing values for the new managed application, the **JIT Configuration** step allows you to enable or disable JIT access for the managed application. Select **Yes** for **Enable JIT Access**. This option is selected by default for managed applications that defined with JIT enabled in the marketplace.



You can only enable JIT access during deployment. If you select **No**, the publisher gets permanent access to the managed resource group. You can't enable JIT access later.

4. To change the default approval settings, select **Customize JIT Configuration**.



By default, a managed application with JIT enabled has the following settings:

- Approval mode – automatic
- Maximum access duration – 8 hours
- Approvers – none

When the approval mode is set to **automatic**, the approvers receive a notification for each request but the request is automatically approved. When set to **manual**, the approvers receive a notification for each request, and one of them must approve it.

The activation maximum duration specifies the maximum amount of time a publisher can request for access to the managed resource group.

The approvers list is the Azure Active Directory users that can approve of JIT access requests. To add an approver, select **Add Approver** and search for the user.

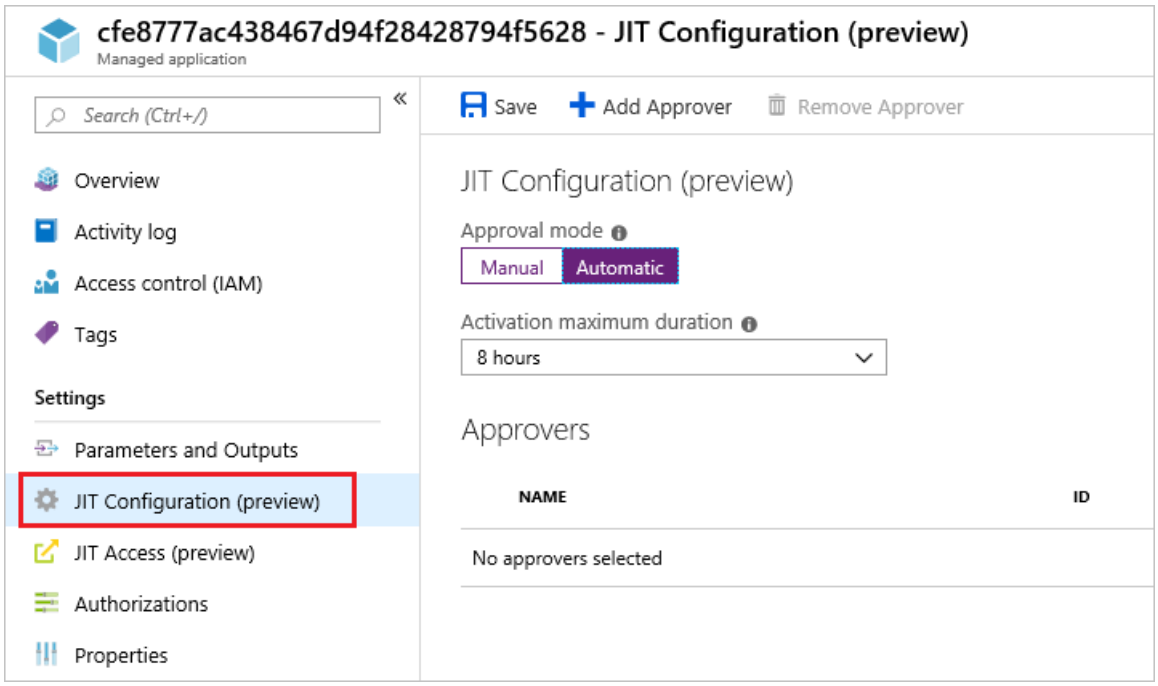
After updating the setting, select **Save**.

Update after deployment

You can change the values for how requests are approved. However, if you didn't enable JIT access during deployment, you can't enable it later.

To change the settings for a deployed managed application:

1. In the portal, select the manage application.
2. Select **JIT Configuration** and change the settings as needed.



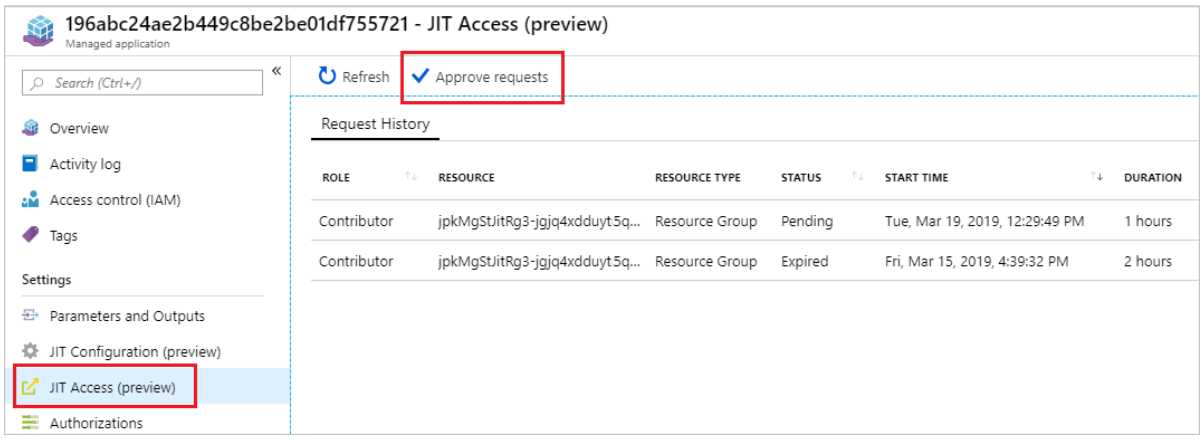
3. When done, select **Save**.

Approve requests

When the publisher requests access, you're notified of the request. You can approve JIT access requests either directly through the managed application, or across all managed applications through the Azure AD Privileged Identity Management service. To use just-in-time access, you must have a Azure Active Directory P2 license.

To approve requests through the managed application:

1. Select **JIT Access** for the managed application, and select **Approve Requests**.

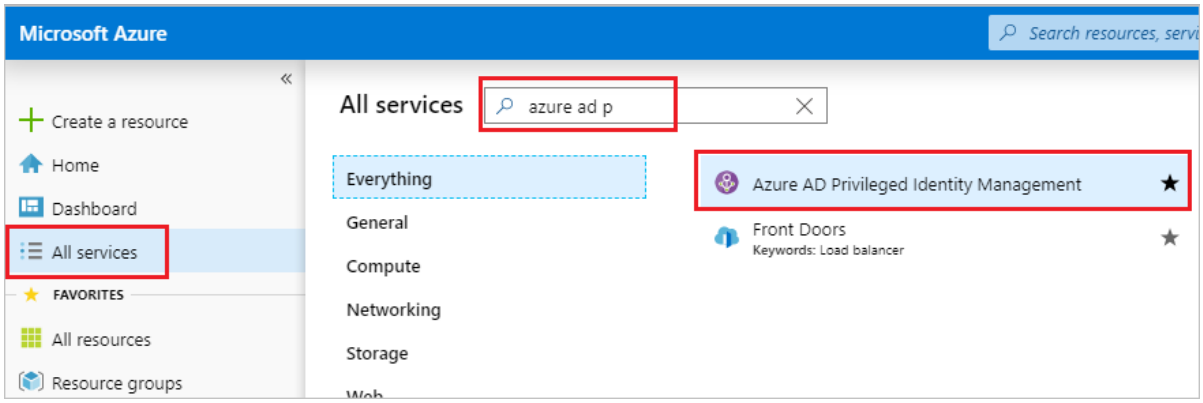


2. Select the request to approve.

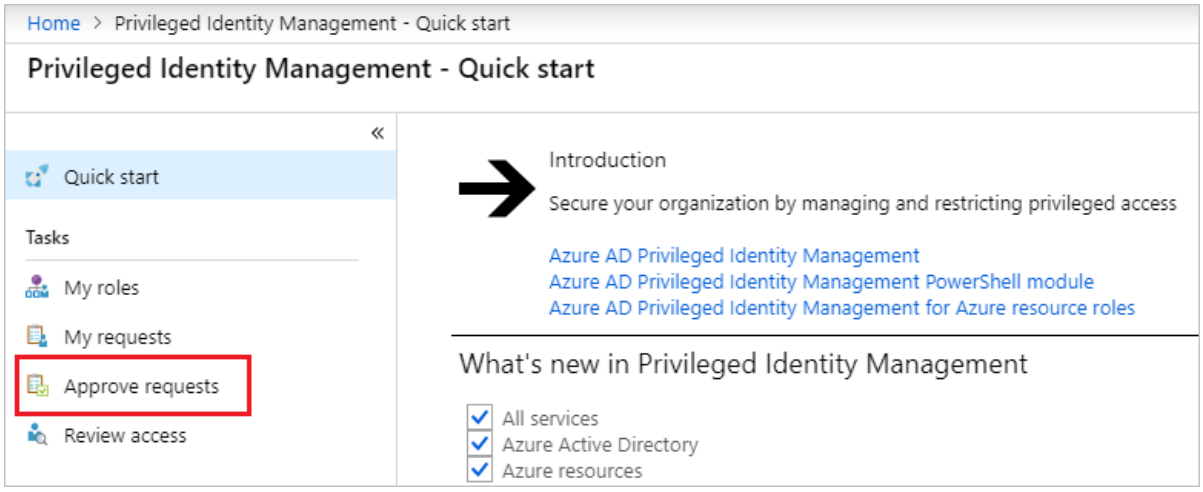
3. In the form, provide the reason for the approval and select **Approve**.

To approve requests through Azure AD Privileged Identity Management:

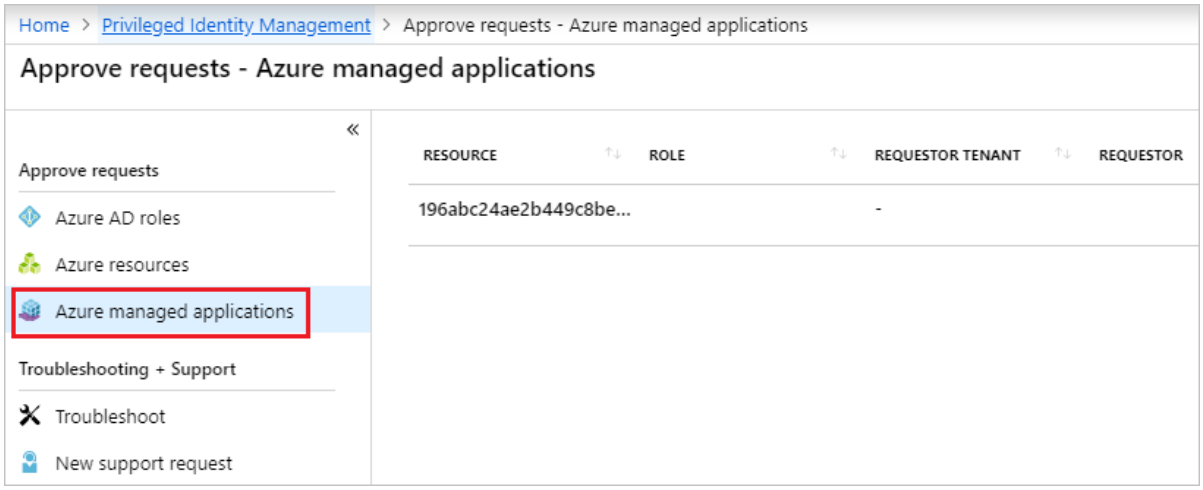
1. Select **All services** and begin searching for **Azure AD Privileged Identity Management**. Select it from the available options.



2. Select **Approve requests**.



3. Select **Azure managed applications**, and select the request to approve.



Next steps

To learn about publishing a managed application with JIT access, see Request just-in-time access in Azure Managed Applications.

Reference: User interface elements artifact

1/2/2020 • 2 minutes to read • Edit Online

This article is a reference for a *createUiDefinition.json* artifact in Azure Managed Applications. For more information about authoring user interface elements, see Create user interface elements.

User interface elements

The following JSON shows an example of *createUiDefinition.json* file for Azure Managed Applications:

```
{
  "$schema": "https://schema.management.azure.com/schemas/0.1.2-preview/CreateUIDefinition.MultiVm.json#",
  "handler": "Microsoft.Azure.CreateUIDef",
  "version": "0.1.2-preview",
  "parameters": {
    "basics": [
      {}
    ],
    "steps": [
      {
        "name": "applicationSettings",
        "label": "Application Settings",
        "subLabel": {
          "preValidation": "Configure your application settings",
          "postValidation": "Done"
        },
        "bladeTitle": "Application Settings",
        "elements": [
          {
            "name": "funcname",
            "type": "Microsoft.Common.TextBox",
            "label": "Name of the function to be created",
            "toolTip": "Name of the function to be created",
            "visible": true,
            "constraints": {
              "required": true
            }
          },
          {
            "name": "storagename",
            "type": "Microsoft.Common.TextBox",
            "label": "Name of the storage to be created",
            "toolTip": "Name of the storage to be created",
            "visible": true,
            "constraints": {
              "required": true
            }
          },
          {
            "name": "zipFileBlobUri",
            "type": "Microsoft.Common.TextBox",
            "defaultValue": "https://github.com/Azure/azure-quickstart-templates/tree/master/101-custom-rp-
with-function/artifacts/functionzip/functionpackage.zip",
            "label": "The Uri to the uploaded function zip file",
            "toolTip": "The Uri to the uploaded function zip file",
            "visible": true
          }
        ]
      }
    ],
    "outputs": {
      "funcname": "[steps('applicationSettings').funcname]",
      "storageName": "[steps('applicationSettings').storagename]",
      "zipFileBlobUri": "[steps('applicationSettings').zipFileBlobUri]"
    }
  }
}
```

Next steps

- Tutorial: Create managed application with custom actions and resources
- Reference: Deployment template artifact
- Reference: View definition artifact

Reference: Deployment template artifact

1/2/2020 • 2 minutes to read • Edit Online

This article is a reference for a *mainTemplate.json* artifact in Azure Managed Applications. For more information about authoring deployment template, see Azure Resource Manager templates.

Deployment template

The following JSON shows an example of *mainTemplate.json* file for Azure Managed Applications:

```
{
  "$schema": "http://schema.management.azure.com/schemas/2014-04-01-preview/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "location": {
      "type": "string",
      "defaultValue": "eastus",
      "allowedValues": [
        "australiaeast",
        "eastus",
        "westeurope"
      ],
      "metadata": {
        "description": "Location for the resources."
      }
    },
    "funcname": {
      "type": "string",
      "metadata": {
        "description": "Name of the Azure Function that hosts the code. Must be globally unique"
      },
      "defaultValue": ""
    },
    "storageName": {
      "type": "string",
      "metadata": {
        "description": "Name of the storage account that hosts the function. Must be globally unique. The
field can contain only lowercase letters and numbers. Name must be between 3 and 24 characters"
      },
      "defaultValue": ""
    },
    "zipFileBlobUri": {
      "type": "string",
      "defaultValue": "https://github.com/Azure/azure-quickstart-templates/tree/master/101-custom-rp-with-
function/artifacts/functionzip/functionpackage.zip",
      "metadata": {
        "description": "The Uri to the uploaded function zip file"
      }
    }
  },
  "variables": {
    "customrpApiversion": "2018-09-01-preview",
    "customProviderName": "public",
    "serverFarmName": "functionPlan"
  },
  "resources": [
    {
      "type": "Microsoft.Web/serverfarms",
      "apiVersion": "2016-09-01",
      "name": "[variables('serverFarmName')]",
      "location": "[parameters('location')]",
```

```
      "sku": {
        "name": "Y1",
        "tier": "Dynamic",
        "size": "Y1",
        "family": "Y",
        "capacity": 0
      },
      "kind": "functionapp",
      "properties": {
        "name": "[variables('serverFarmName')]",
        "perSiteScaling": false,
        "reserved": false,
        "targetWorkerCount": 0,
        "targetWorkerSizeId": 0
      }
    },
    {
      "type": "Microsoft.Web/sites",
      "kind": "functionapp",
      "name": "[parameters('funcname')]",
      "apiVersion": "2018-02-01",
      "location": "[parameters('location')]",
      "dependsOn": [
        "[resourceId('Microsoft.Storage/storageAccounts', parameters('storageName'))]",
        "[resourceId('Microsoft.Web/serverfarms', variables('serverFarmName'))]"
      ],
      "identity": {
        "type": "SystemAssigned"
      },
      "properties": {
        "name": "[parameters('funcname')]",
        "siteConfig": {
          "appSettings": [
            {
              "name": "AzureWebJobsDashboard",
              "value": "
[concat('DefaultEndpointsProtocol=https;AccountName=',parameters('storageName'),';AccountKey=',listKeys(resour
ceId('Microsoft.Storage/storageAccounts', parameters('storageName')), '2015-05-01-preview').key1)]"
            },
            {
              "name": "AzureWebJobsStorage",
              "value": "
[concat('DefaultEndpointsProtocol=https;AccountName=',parameters('storageName'),';AccountKey=',listKeys(resour
ceId('Microsoft.Storage/storageAccounts', parameters('storageName')), '2015-05-01-preview').key1)]"
            },
            {
              "name": "FUNCTIONS_EXTENSION_VERSION",
              "value": "~2"
            },
            {
              "name": "AzureWebJobsSecretStorageType",
              "value": "Files"
            },
            {
              "name": "WEBSITE_CONTENTAZUREFILECONNECTIONSTRING",
              "value": "
[concat('DefaultEndpointsProtocol=https;AccountName=',parameters('storageName'),';AccountKey=',listKeys(resour
ceId('Microsoft.Storage/storageAccounts', parameters('storageName')), '2015-05-01-preview').key1)]"
            },
            {
              "name": "WEBSITE_CONTENTSHARE",
              "value": "[concat(toLower(parameters('funcname')), 'b86e')]"
            },
            {
              "name": "WEBSITE_NODE_DEFAULT_VERSION",
              "value": "6.5.0"
            },
            {
              "name": "WEBSITE_RUN_FROM_PACKAGE",
```

```
                "value": "[parameters('zipFileBlobUri')]"
              }
            ]
          },
          "clientAffinityEnabled": false,
          "reserved": false,
          "serverFarmId": "[resourceId('Microsoft.Web/serverfarms', variables('serverFarmName'))]"
        }
      },
      {
        "type": "Microsoft.Storage/storageAccounts",
        "name": "[parameters('storageName')]",
        "apiVersion": "2018-02-01",
        "kind": "StorageV2",
        "location": "[parameters('location')]",
        "sku": {
          "name": "Standard_LRS"
        }
      },
      {
        "apiVersion": "[variables('customrpApiversion')]",
        "type": "Microsoft.CustomProviders/resourceProviders",
        "name": "[variables('customProviderName')]",
        "location": "[parameters('location')]",
        "properties": {
          "actions": [
            {
              "name": "ping",
              "routingType": "Proxy",
              "endpoint": "[listSecrets(resourceId('Microsoft.Web/sites/functions', parameters('funcname'),
'HttpTrigger1'), '2018-02-01').trigger_url]"
            },
            {
              "name": "users/contextAction",
              "routingType": "Proxy",
              "endpoint": "[listSecrets(resourceId('Microsoft.Web/sites/functions', parameters('funcname'),
'HttpTrigger1'), '2018-02-01').trigger_url]"
            }
          ],
          "resourceTypes": [
            {
              "name": "users",
              "routingType": "Proxy,Cache",
              "endpoint": "[listSecrets(resourceId('Microsoft.Web/sites/functions', parameters('funcname'),
'HttpTrigger1'), '2018-02-01').trigger_url]"
            }
          ]
        },
        "dependsOn": [
          "[concat('Microsoft.Web/sites/',parameters('funcname'))]"
        ]
      }
    ],
    "outputs": {}
  }
```

Next steps

- Tutorial: Create managed application with custom actions and resources
- Reference: User interface elements artifact
- Reference: View definition artifact

Reference: View definition artifact

1/2/2020 • 2 minutes to read • Edit Online

This article is a reference for a *viewDefinition.json* artifact in Azure Managed Applications. For more information about authoring views configuration, see View definition artifact.

View definition

The following JSON shows an example of *viewDefinition.json* file for Azure Managed Applications:

```
{
  "views": [{
    "kind": "Overview",
    "properties": {
      "header": "Welcome to your Demo Azure Managed Application",
      "description": "This Managed application with Custom Provider is for demo purposes only.",
      "commands": [{
          "displayName": "Ping Action",
          "path": "/customping",
          "icon": "LaunchCurrent"
      }]
    }
  },
  {
    "kind": "CustomResources",
    "properties": {
      "displayName": "Users",
      "version": "1.0.0.0",
      "resourceType": "users",
      "createUIDefinition": {
        "parameters": {
          "steps": [{
            "name": "add",
            "label": "Add user",
            "elements": [{
              "name": "name",
              "label": "User's Full Name",
              "type": "Microsoft.Common.TextBox",
              "defaultValue": "",
              "toolTip": "Provide a full user name.",
              "constraints": { "required": true }
            },
            {
              "name": "location",
              "label": "User's Location",
              "type": "Microsoft.Common.TextBox",
              "defaultValue": "",
              "toolTip": "Provide a Location.",
              "constraints": { "required": true }
            }]
          }],
          "outputs": {
            "name": "[steps('add').name]",
            "properties": {
              "FullName": "[steps('add').name]",
              "Location": "[steps('add').location]"
            }
          }
        }
      },
      "commands": [{
        "displayName": "Custom Context Action",
        "path": "users/contextAction",
        "icon": "Start"
      }],
      "columns": [
        { "key": "properties.FullName", "displayName": "Full Name" },
        { "key": "properties.Location", "displayName": "Location", "optional": true }
      ]
    }
  }]
}
```

Next steps

- Tutorial: Create managed application with custom actions and resources
- Reference: User interface elements artifact
- Reference: Deployment template artifact

CreateUiDefinition functions

1/2/2020 • 8 minutes to read • Edit Online

This section contains the signatures for all supported functions of a CreateUiDefinition.

To use a function, surround the declaration with square brackets. For example:

```
"[function()]"
```

Strings and other functions can be referenced as parameters for a function, but strings must be surrounded in single quotes. For example:

```
"[fn1(fn2(), 'foobar')]"
```

Where applicable, you can reference properties of the output of a function by using the dot operator. For example:

```
"[func().prop1]"
```

Referencing functions

These functions can be used to reference outputs from the properties or context of a CreateUiDefinition.

basics

Returns the output values of an element that is defined in the Basics step.

The following example returns the output of the element named `foo` in the Basics step:

```
"[basics('foo')]"
```

steps

Returns the output values of an element that is defined in the specified step. To get the output values of elements in the Basics step, use `basics()` instead.

The following example returns the output of the element named `bar` in the step named `foo`:

```
"[steps('foo').bar]"
```

location

Returns the location selected in the Basics step or the current context.

The following example could return `"westus"`:

```
"[location()]"
```

String functions

These functions can only be used with JSON strings.

concat

Concatenates one or more strings.

For example, if the output value of `element1` if `"bar"`, then this example returns the string `"foobar!"`:

```
"[concat('foo', steps('step1').element1, '!')]"
```

substring

Returns the substring of the specified string. The substring starts at the specified index and has the specified length.

The following example returns `"ftw"`:

```
"[substring('azure-ftw!!!1one', 6, 3)]"
```

replace

Returns a string in which all occurrences of the specified string in the current string are replaced with another string.

The following example returns `"Everything is awesome!"`:

```
"[replace('Everything is terrible!', 'terrible', 'awesome')]"
```

guid

Generates a globally unique string (GUID).

The following example could return `"c7bc8bdc-7252-4a82-ba53-7c468679a511"`:

```
"[guid()]"
```

toLower

Returns a string converted to lowercase.

The following example returns `"foobar"`:

```
"[toLower('FOOBAR')]"
```

toUpper

Returns a string converted to uppercase.

The following example returns `"FOOBAR"`:

```
"[toUpper('foobar')]"
```

Collection functions

These functions can be used with collections, like JSON strings, arrays and objects.

contains

Returns `true` if a string contains the specified substring, an array contains the specified value, or an object

contains the specified key.

Example 1: string

The following example returns `true`:

```
"[contains('foobar', 'foo')]"
```

Example 2: array

Assume `element1` returns `[1, 2, 3]`. The following example returns `false`:

```
"[contains(steps('foo').element1, 4)]"
```

Example 3: object

Assume `element1` returns:

```
{
  "key1": "foobar",
  "key2": "raboof"
}
```

The following example returns `true`:

```
"[contains(steps('foo').element1, 'key1')]"
```

length

Returns the number of characters in a string, the number of values in an array, or the number of keys in an object.

Example 1: string

The following example returns `6`:

```
"[length('foobar')]"
```

Example 2: array

Assume `element1` returns `[1, 2, 3]`. The following example returns `3`:

```
"[length(steps('foo').element1)]"
```

Example 3: object

Assume `element1` returns:

```
{
  "key1": "foobar",
  "key2": "raboof"
}
```

The following example returns `2`:

```
"[length(steps('foo').element1)]"
```

empty

Returns `true` if the string, array, or object is null or empty.

Example 1: string

The following example returns `true`:

```
"[empty('')]"
```

Example 2: array

Assume `element1` returns `[1, 2, 3]`. The following example returns `false`:

```
"[empty(steps('foo').element1)]"
```

Example 3: object

Assume `element1` returns:

```
{
  "key1": "foobar",
  "key2": "raboof"
}
```

The following example returns `false`:

```
"[empty(steps('foo').element1)]"
```

Example 4: null and undefined

Assume `element1` is `null` or undefined. The following example returns `true`:

```
"[empty(steps('foo').element1)]"
```

first

Returns the first character of the specified string; first value of the specified array; or the first key and value of the specified object.

Example 1: string

The following example returns `"f"`:

```
"[first('foobar')]"
```

Example 2: array

Assume `element1` returns `[1, 2, 3]`. The following example returns `1`:

```
"[first(steps('foo').element1)]"
```

Example 3: object

Assume `element1` returns:

```
{
  "key1": "foobar",
  "key2": "raboof"
}
```

The following example returns `{"key1": "foobar"}`:

```
"[first(steps('foo').element1)]"
```

last

Returns the last character of the specified string, the last value of the specified array, or the last key and value of the specified object.

Example 1: string

The following example returns `"r"`:

```
"[last('foobar')]"
```

Example 2: array

Assume `element1` returns `[1, 2, 3]`. The following example returns `2`:

```
"[last(steps('foo').element1)]"
```

Example 3: object

Assume `element1` returns:

```
{
  "key1": "foobar",
  "key2": "raboof"
}
```

The following example returns `{"key2": "raboof"}`:

```
"[last(steps('foo').element1)]"
```

take

Returns a specified number of contiguous characters from the start of the string, a specified number of contiguous values from the start of the array, or a specified number of contiguous keys and values from the start of the object.

Example 1: string

The following example returns `"foo"`:

```
"[take('foobar', 3)]"
```

Example 2: array

Assume `element1` returns `[1, 2, 3]`. The following example returns `[1, 2]`:

```
"[take(steps('foo').element1, 2)]"
```

Example 3: object

Assume `element1` returns:

```
{
  "key1": "foobar",
  "key2": "raboof"
}
```

The following example returns `{"key1": "foobar"}`:

```
"[take(steps('foo').element1, 1)]"
```

skip

Bypasses a specified number of elements in a collection, and then returns the remaining elements.

Example 1: string

The following example returns `"bar"`:

```
"[skip('foobar', 3)]"
```

Example 2: array

Assume `element1` returns `[1, 2, 3]`. The following example returns `[3]`:

```
"[skip(steps('foo').element1, 2)]"
```

Example 3: object

Assume `element1` returns:

```
{
  "key1": "foobar",
  "key2": "raboof"
}
```

The following example returns `{"key2": "raboof"}`:

```
"[skip(steps('foo').element1, 1)]"
```

Logical functions

These functions can be used in conditionals. Some functions may not support all JSON data types.

equals

Returns `true` if both parameters have the same type and value. This function supports all JSON data types.

The following example returns `true`:

```
"[equals(0, 0)]"
```

The following example returns `true`:

```
"[equals('foo', 'foo')]"
```

The following example returns `false`:

```
"[equals('abc', ['a', 'b', 'c'])]"
```

less

Returns `true` if the first parameter is strictly less than the second parameter. This function supports parameters

only of type number and string.

The following example returns `true`:

```
"[less(1, 2)]"
```

The following example returns `false`:

```
"[less('9', '10')]"
```

lessOrEquals

Returns `true` if the first parameter is less than or equal to the second parameter. This function supports parameters only of type number and string.

The following example returns `true`:

```
"[lessOrEquals(2, 2)]"
```

greater

Returns `true` if the first parameter is strictly greater than the second parameter. This function supports parameters only of type number and string.

The following example returns `false`:

```
"[greater(1, 2)]"
```

The following example returns `true`:

```
"[greater('9', '10')]"
```

greaterOrEquals

Returns `true` if the first parameter is greater than or equal to the second parameter. This function supports parameters only of type number and string.

The following example returns `true`:

```
"[greaterOrEquals(2, 2)]"
```

and

Returns `true` if all the parameters evaluate to `true`. This function supports two or more parameters only of type Boolean.

The following example returns `true`:

```
"[and(equals(0, 0), equals('foo', 'foo'), less(1, 2))]"
```

The following example returns `false`:

```
"[and(equals(0, 0), greater(1, 2))]"
```

or

Returns `true` if at least one of the parameters evaluates to `true`. This function supports two or more parameters only of type Boolean.

The following example returns `true`:

```
"[or(equals(0, 0), equals('foo', 'foo'), less(1, 2))]"
```

The following example returns `true`:

```
"[or(equals(0, 0), greater(1, 2))]"
```

not

Returns `true` if the parameter evaluates to `false`. This function supports parameters only of type Boolean.

The following example returns `true`:

```
"[not(false)]"
```

The following example returns `false`:

```
"[not(equals(0, 0))]"
```

coalesce

Returns the value of the first non-null parameter. This function supports all JSON data types.

Assume `element1` and `element2` are undefined. The following example returns `"foobar"`:

```
"[coalesce(steps('foo').element1, steps('foo').element2, 'foobar')]"
```

Conversion functions

These functions can be used to convert values between JSON data types and encodings.

int

Converts the parameter to an integer. This function supports parameters of type number and string.

The following example returns `1`:

```
"[int('1')]"
```

The following example returns `2`:

```
"[int(2.9)]"
```

float

Converts the parameter to a floating-point. This function supports parameters of type number and string.

The following example returns `1.0`:

```
"[float('1.0')]"
```

The following example returns `2.9`:

```
"[float(2.9)]"
```

string

Converts the parameter to a string. This function supports parameters of all JSON data types.

The following example returns `"1"`:

```
"[string(1)]"
```

The following example returns `"2.9"`:

```
"[string(2.9)]"
```

The following example returns `"[1,2,3]"`:

```
"[string([1,2,3])]"
```

The following example returns `"{"foo":"bar"}"`:

```
"[string({\"foo\":\"bar\"})]"
```

bool

Converts the parameter to a Boolean. This function supports parameters of type number, string, and Boolean. Similar to Booleans in JavaScript, any value except `0` or `'false'` returns `true`.

The following example returns `true`:

```
"[bool(1)]"
```

The following example returns `false`:

```
"[bool(0)]"
```

The following example returns `true`:

```
"[bool(true)]"
```

The following example returns `true`:

```
"[bool('true')]"
```

parse

Converts the parameter to a native type. In other words, this function is the inverse of `string()`. This function

supports parameters only of type string.

The following example returns `1`:

```
"[parse('1')]"
```

The following example returns `true`:

```
"[parse('true')]"
```

The following example returns `[1,2,3]`:

```
"[parse('[1,2,3]')]"
```

The following example returns `{"foo":"bar"}`:

```
"[parse('{\"foo\":\"bar\"}')]"
```

encodeBase64

Encodes the parameter to a base-64 encoded string. This function supports parameters only of type string.

The following example returns `"Zm9vYmFy"`:

```
"[encodeBase64('foobar')]"
```

decodeBase64

Decodes the parameter from a base-64 encoded string. This function supports parameters only of type string.

The following example returns `"foobar"`:

```
"[decodeBase64('Zm9vYmFy')]"
```

encodeUriComponent

Encodes the parameter to a URL encoded string. This function supports parameters only of type string.

The following example returns `"https%3A%2F%2Fportal.azure.com%2F"`:

```
"[encodeUriComponent('https://portal.azure.com/')]"
```

decodeUriComponent

Decodes the parameter from a URL encoded string. This function supports parameters only of type string.

The following example returns `"https://portal.azure.com/"`:

```
"[decodeUriComponent('https%3A%2F%2Fportal.azure.com%2F')]"
```

Math functions

add

Adds two numbers, and returns the result.

The following example returns `3`:

```
"[add(1, 2)]"
```

sub

Subtracts the second number from the first number, and returns the result.

The following example returns `1`:

```
"[sub(3, 2)]"
```

mul

Multiplies two numbers, and returns the result.

The following example returns `6`:

```
"[mul(2, 3)]"
```

div

Divides the first number by the second number, and returns the result. The result is always an integer.

The following example returns `2`:

```
"[div(6, 3)]"
```

mod

Divides the first number by the second number, and returns the remainder.

The following example returns `0`:

```
"[mod(6, 3)]"
```

The following example returns `2`:

```
"[mod(6, 4)]"
```

min

Returns the small of the two numbers.

The following example returns `1`:

```
"[min(1, 2)]"
```

max

Returns the larger of the two numbers.

The following example returns `2`:

```
"[max(1, 2)]"
```

range

Generates a sequence of integral numbers within the specified range.

The following example returns `[1,2,3]`:

```
"[range(1, 3)]"
```

rand

Returns a random integral number within the specified range. This function does not generate cryptographically secure random numbers.

The following example could return `42`:

```
"[rand(-100, 100)]"
```

floor

Returns the largest integer less than or equal to the specified number.

The following example returns `3`:

```
"[floor(3.14)]"
```

ceil

Returns the largest integer greater than or equal to the specified number.

The following example returns `4`:

```
"[ceil(3.14)]"
```

Date functions

utcNow

Returns a string in ISO 8601 format of the current date and time on the local computer.

The following example could return `"1990-12-31T23:59:59.000Z"`:

```
"[utcNow()]"
```

addSeconds

Adds an integral number of seconds to the specified timestamp.

The following example returns `"1991-01-01T00:00:00.000Z"`:

```
"[addSeconds('1990-12-31T23:59:60Z', 1)]"
```

addMinutes

Adds an integral number of minutes to the specified timestamp.

The following example returns `"1991-01-01T00:00:59.000Z"`:

```
"[addMinutes('1990-12-31T23:59:59Z', 1)]"
```

addHours

Adds an integral number of hours to the specified timestamp.

The following example returns `"1991-01-01T00:59:59.000Z"`:

```
"[addHours('1990-12-31T23:59:59Z', 1)]"
```

Next steps

- For an introduction to Azure Resource Manager, see Azure Resource Manager overview.

CreateUiDefinition elements

2/7/2020 • 2 minutes to read • Edit Online

This article describes the schema and properties for all supported elements of a CreateUiDefinition.

Schema

The schema for most elements is as follows:

```
{
  "name": "element1",
  "type": "Microsoft.Common.TextBox",
  "label": "Some text box",
  "defaultValue": "my value",
  "toolTip": "Provide a descriptive name.",
  "constraints": {},
  "options": {},
  "visible": true
}
```

PROPERTY	REQUIRED	DESCRIPTION
name	Yes	An internal identifier to reference a specific instance of an element. The most common usage of the element name is in `outputs`, where the output values of the specified elements are mapped to the parameters of the template. You can also use it to bind the output value of an element to the `defaultValue` of another element.
type	Yes	The UI control to render for the element. For a list of supported types, see Elements.
label	Yes	The display text of the element. Some element types contain multiple labels, so the value could be an object containing multiple strings.
defaultValue	No	The default value of the element. Some element types support complex default values, so the value could be an object.
toolTip	No	The text to display in the tool tip of the element. Similar to `label`, some elements support multiple tool tip strings. Inline links can be embedded using Markdown syntax.

PROPERTY	REQUIRED	DESCRIPTION
constraints	No	One or more properties that are used to customize the validation behavior of the element. The supported properties for constraints vary by element type. Some element types do not support customization of the validation behavior, and thus have no constraints property.
options	No	Additional properties that customize the behavior of the element. Similar to `constraints`, the supported properties vary by element type.
visible	No	Indicates whether the element is displayed. If `true`, the element and applicable child elements are displayed. The default value is `true`. Use logical functions to dynamically control this property's value.

Elements

The documentation for each element contains a UI sample, schema, remarks on the behavior of the element (usually concerning validation and supported customization), and sample output.

- Microsoft.Common.DropDown
- Microsoft.Common.FileUpload
- Microsoft.Common.InfoBox
- Microsoft.Common.OptionsGroup
- Microsoft.Common.PasswordBox
- Microsoft.Common.Section
- Microsoft.Common.TagsByResource
- Microsoft.Common.TextBlock
- Microsoft.Common.TextBox
- Microsoft.Compute.CredentialsCombo
- Microsoft.Compute.SizeSelector
- Microsoft.Compute.UserNameTextBox
- Microsoft.ManagedIdentity.IdentitySelector
- Microsoft.Network.PublicIpAddressCombo
- Microsoft.Network.VirtualNetworkCombo
- Microsoft.Storage.MultiStorageAccountCombo
- Microsoft.Storage.StorageAccountSelector

Next steps

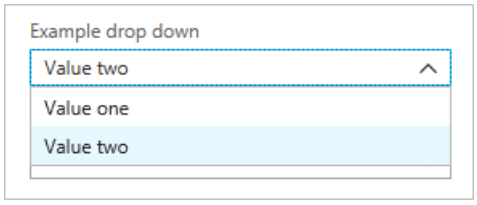
For an introduction to creating UI definitions, see Getting started with CreateUiDefinition.

Microsoft.Common.DropDown UI element

1/2/2020 • 2 minutes to read • Edit Online

A selection control with a dropdown list.

UI sample



Schema

```
{
  "name": "element1",
  "type": "Microsoft.Common.DropDown",
  "label": "Example drop down",
  "defaultValue": "Value two",
  "toolTip": "",
  "constraints": {
    "allowedValues": [
      {
        "label": "Value one",
        "value": "one"
      },
      {
        "label": "Value two",
        "value": "two"
      }
    ],
    "required": true
  },
  "visible": true
}
```

Sample output

```
"two"
```

Remarks

- The label for `constraints.allowedValues` is the display text for an item, and its value is the output value of the element when selected.
- If specified, the default value must be a label present in `constraints.allowedValues`. If not specified, the first item in `constraints.allowedValues` is selected. The default value is **null**.
- `constraints.allowedValues` must have at least one item.
- To emulate a value not being required, add an item with a label and value of `""` (empty string) to `constraints.allowedValues`.

Next steps

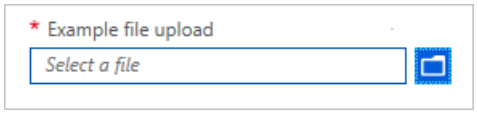
- For an introduction to creating UI definitions, see Getting started with CreateUiDefinition.
- For a description of common properties in UI elements, see CreateUiDefinition elements.

Microsoft.Common.FileUpload UI element

1/2/2020 • 2 minutes to read • Edit Online

A control that allows a user to specify one or more files to upload.

UI sample



Schema

```
{
  "name": "element1",
  "type": "Microsoft.Common.FileUpload",
  "label": "Some file upload",
  "toolTip": "",
  "constraints": {
    "required": true,
    "accept": ".doc,.docx,.xml,application/msword"
  },
  "options": {
    "multiple": false,
    "uploadMode": "file",
    "openMode": "text",
    "encoding": "UTF-8"
  },
  "visible": true
}
```

Sample output

If options.multiple is false and options.uploadMode is file, then the output has the contents of the file as a JSON string:

```
"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua."
```

If options.multiple is true and`options.uploadMode is file, then the output has the contents of the files as a JSON array:

```
[
  "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.",
  "Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.",
  "Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.",
  "Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
]
```

If options.multiple is false and options.uploadMode is url, then the output has a URL as a JSON string:

```
"https://myaccount.blob.core.windows.net/pictures/profile.jpg?sv=2013-08-15&st=2013-08-16&se=2013-08-
17&sr=c&sp=r&rscd=file;%20attachment&rsct=binary
&sig=YWJjZGVmZw%3d%3d&sig=a39%2BYozJhGp6miujGymjRpN8tsrQfLo9Z3i8IRyIpnQ%3d"
```

If options.multiple is true and options.uploadMode is url, then the output has a list of URLs as a JSON array:

```
[
  "https://myaccount.blob.core.windows.net/pictures/profile1.jpg?sv=2013-08-15&st=2013-08-16&se=2013-08-
17&sr=c&sp=r&rscd=file;%20attachment&rsct=binary
&sig=YWJjZGVmZw%3d%3d&sig=a39%2BYozJhGp6miujGymjRpN8tsrQfLo9Z3i8IRyIpnQ%3d",
  "https://myaccount.blob.core.windows.net/pictures/profile2.jpg?sv=2013-08-15&st=2013-08-16&se=2013-08-
17&sr=c&sp=r&rscd=file;%20attachment&rsct=binary
&sig=YWJjZGVmZw%3d%3d&sig=a39%2BYozJhGp6miujGymjRpN8tsrQfLo9Z3i8IRyIpnQ%3d",
  "https://myaccount.blob.core.windows.net/pictures/profile3.jpg?sv=2013-08-15&st=2013-08-16&se=2013-08-
17&sr=c&sp=r&rscd=file;%20attachment&rsct=binary
&sig=YWJjZGVmZw%3d%3d&sig=a39%2BYozJhGp6miujGymjRpN8tsrQfLo9Z3i8IRyIpnQ%3d"
]
```

When testing a CreateUiDefinition, some browsers (like Google Chrome) truncate URLs generated by the Microsoft.Common.FileUpload element in the browser console. You may need to right-click individual links to copy the full URLs.

Remarks

- `constraints.accept` specifies the types of files that are shown in the browser's file dialog. See the HTML5 specification for allowed values. The default value is **null**.
- If `options.multiple` is set to **true**, the user is allowed to select more than one file in the browser's file dialog. The default value is **false**.
- This element supports uploading files in two modes based on the value of `options.uploadMode`. If **file** is specified, the output has the contents of the file as a blob. If **url** is specified, then the file is uploaded to a temporary location, and the output has the URL of the blob. Temporary blobs will be purged after 24 hours. The default value is **file**.
- An uploaded file is protected. The output URL includes a SAS token for accessing the file during deployment.
- The value of `options.openMode` determines how the file is read. If the file is expected to be plain text, specify **text**; else, specify **binary**. The default value is **text**.
- If `options.uploadMode` is set to **file** and `options.openMode` is set to **binary**, the output is base64-encoded.
- `options.encoding` specifies the encoding to use when reading the file. The default value is **UTF-8**, and is used only when `options.openMode` is set to **text**.

Next steps

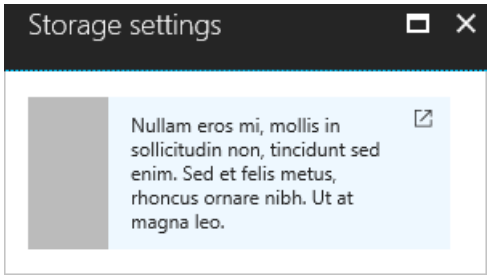
- For an introduction to creating UI definitions, see Getting started with CreateUiDefinition.
- For a description of common properties in UI elements, see CreateUiDefinition elements.

Microsoft.Common.InfoBox UI element

1/2/2020 • 2 minutes to read • Edit Online

A control that adds an information box. The box contains important text or warnings that help users understand the values they're providing. It can also link to a URI for more information.

UI sample



Schema

```
{
  "name": "text1",
  "type": "Microsoft.Common.InfoBox",
  "visible": true,
  "options": {
    "icon": "None",
    "text": "Nullam eros mi, mollis in sollicitudin non, tincidunt sed enim. Sed et felis metus, rhoncus
ornare nibh. Ut at magna leo.",
    "uri": "https://www.microsoft.com"
  }
}
```

Sample output

```
"Nullam eros mi, mollis in sollicitudin non, tincidunt sed enim. Sed et felis metus, rhoncus ornare nibh. Ut
at magna leo."
```

Remarks

- For `icon`, use **None**, **Info**, **Warning**, or **Error**.
- The `uri` property is optional.

Next steps

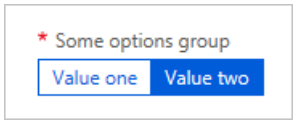
- For an introduction to creating UI definitions, see Getting started with CreateUiDefinition.
- For a description of common properties in UI elements, see CreateUiDefinition elements.

Microsoft.Common.OptionsGroup UI element

1/2/2020 • 2 minutes to read • Edit Online

A selection control with a row of available options.

UI sample



Schema

```
{
  "name": "element1",
  "type": "Microsoft.Common.OptionsGroup",
  "label": "Some options group",
  "defaultValue": "Value two",
  "toolTip": "",
  "constraints": {
    "allowedValues": [
      {
        "label": "Value one",
        "value": "one"
      },
      {
        "label": "Value two",
        "value": "two"
      }
    ],
    "required": true
  },
  "visible": true
}
```

Sample output

```
"two"
```

Remarks

- The label for `constraints.allowedValues` is the display text for an item, and its value is the output value of the element when selected.
- If specified, the default value must be a label present in `constraints.allowedValues`. If not specified, the first item in `constraints.allowedValues` is selected by default. The default value is **null**.
- `constraints.allowedValues` must have at least one item.

Next steps

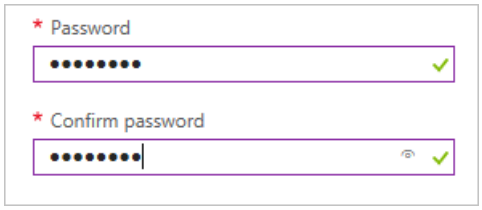
- For an introduction to creating UI definitions, see Getting started with CreateUiDefinition.
- For a description of common properties in UI elements, see CreateUiDefinition elements.

Microsoft.Common.PasswordBox UI element

1/2/2020 • 2 minutes to read • Edit Online

A control that can be used to provide and confirm a password.

UI sample



Schema

```
{
  "name": "element1",
  "type": "Microsoft.Common.PasswordBox",
  "label": {
    "password": "Password",
    "confirmPassword": "Confirm password"
  },
  "toolTip": "",
  "constraints": {
    "required": true,
    "regex": "^[a-zA-Z0-9]{8,}$",
    "validationMessage": "Password must be at least 8 characters long, contain only numbers and letters"
  },
  "options": {
    "hideConfirmation": false
  },
  "visible": true
}
```

Sample output

```
"p4ssw0rd"
```

Remarks

- This element doesn't support the `defaultValue` property.
- For implementation details of `constraints`, see Microsoft.Common.TextBox.
- If `options.hideConfirmation` is set to **true**, the second text box for confirming the user's password is hidden. The default value is **false**.

Next steps

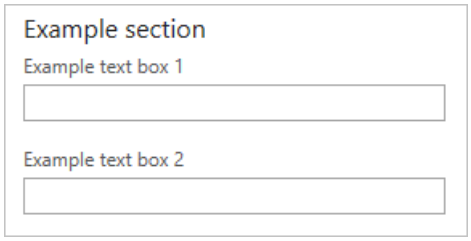
- For an introduction to creating UI definitions, see Getting started with CreateUiDefinition.
- For a description of common properties in UI elements, see CreateUiDefinition elements.

Microsoft.Common.Section UI element

1/2/2020 • 2 minutes to read • Edit Online

A control that groups one or more elements under a heading.

UI sample



Schema

```
{
  "name": "section1",
  "type": "Microsoft.Common.Section",
  "label": "Example section",
  "elements": [
    {
      "name": "text1",
      "type": "Microsoft.Common.TextBox",
      "label": "Example text box 1"
    },
    {
      "name": "text2",
      "type": "Microsoft.Common.TextBox",
      "label": "Example text box 2"
    }
  ],
  "visible": true
}
```

Remarks

- `elements` must have at least one element, and can have all element types except `Microsoft.Common.Section`.
- This element doesn't support the `toolTip` property.

Sample output

To access the output values of elements in `elements`, use the basics() or steps() functions and dot notation:

```
steps('configuration').section1.text1
```

Elements of type `Microsoft.Common.Section` have no output values themselves.

Next steps

- For an introduction to creating UI definitions, see Getting started with CreateUiDefinition.

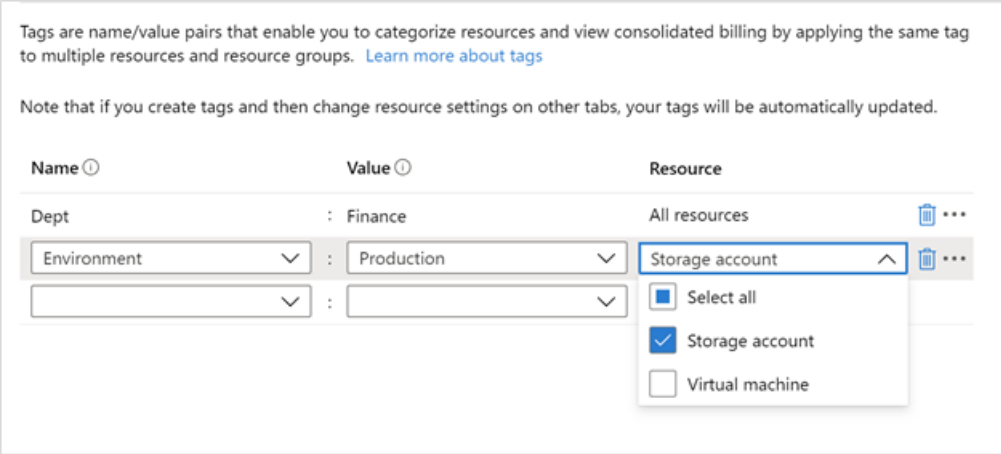
- For a description of common properties in UI elements, see CreateUiDefinition elements.

Microsoft.Common.TagsByResource UI element

1/2/2020 • 2 minutes to read • Edit Online

A control for associating tags with the resources in a deployment.

UI sample



Schema

```
{
  "name": "element1",
  "type": "Microsoft.Common.TagsByResource",
  "resources": [
    "Microsoft.Storage/storageAccounts",
    "Microsoft.Compute/virtualMachines"
  ]
}
```

Sample output

```
{
  "Microsoft.Storage/storageAccounts": {
    "Dept": "Finance",
    "Environment": "Production"
  },
  "Microsoft.Compute/virtualMachines": {
    "Dept": "Finance"
  }
}
```

Remarks

- At least one item in the `resources` array must be specified.
- Each element in `resources` must be a fully qualified resource type. These elements appear in the **Resource** dropdown, and are taggable by the user.
- The output of the control is formatted for easy assignment of tag values in an Azure Resource Manager

template. To receive the control's output in a template, include a parameter in your template as shown in the following example:

```
"parameters": {
  "tagsByResource": { "type": "object", "defaultValue": {} }
}
```

For each resource that can be tagged, assign the tags property to the parameter value for that resource type:

```
{
  "name": "saName1",
  "type": "Microsoft.Storage/storageAccounts",
  "tags": "[ if(contains(parameters('tagsByResource'), 'Microsoft.Storage/storageAccounts'),
parameters('tagsByResource')['Microsoft.Storage/storageAccounts'], json('{}')) ]",
  ...
```

- Use the if function when accessing the tagsByResource parameter. It enables you to assign an empty object when no tags are assigned to the given resource type.

Next steps

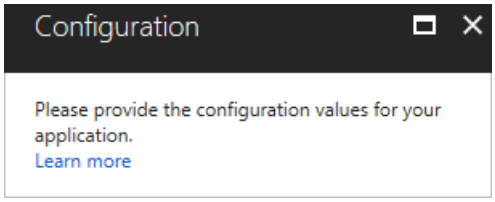
- For an introduction to creating UI definitions, see Getting started with CreateUiDefinition.
- For a description of common properties in UI elements, see CreateUiDefinition elements.

Microsoft.Common.TextBlock UI element

1/2/2020 • 2 minutes to read • Edit Online

A control that can be used to add text to the portal interface.

UI sample



Schema

```
{
  "name": "text1",
  "type": "Microsoft.Common.TextBlock",
  "visible": true,
  "options": {
    "text": "Please provide the configuration values for your application.",
    "link": {
      "label": "Learn more",
      "uri": "https://www.microsoft.com"
    }
  }
}
```

Sample output

```
"Please provide the configuration values for your application. Learn more"
```

Next steps

- For an introduction to creating UI definitions, see Getting started with CreateUiDefinition.
- For a description of common properties in UI elements, see CreateUiDefinition elements.

Microsoft.Common.TextBox UI element

1/2/2020 • 2 minutes to read • Edit Online

A control that can be used to edit unformatted text.

UI sample

Schema

```
{
  "name": "element1",
  "type": "Microsoft.Common.TextBox",
  "label": "Example text box 1",
  "defaultValue": "my text value",
  "toolTip": "Use only allowed characters",
  "constraints": {
    "required": true,
    "regex": "^[a-z0-9A-Z]{1,30}$",
    "validationMessage": "Only alphanumeric characters are allowed, and the value must be 1-30 characters
long."
  },
  "visible": true
}
```

Sample output

```
"my text value"
```

Remarks

- If `constraints.required` is set to **true**, then the text box must have a value to validate successfully. The default value is **false**.
- `constraints.regex` is a JavaScript regular expression pattern. If specified, then the text box's value must match the pattern to validate successfully. The default value is **null**.
- `constraints.validationMessage` is a string to display when the text box's value fails validation. If not specified, then the text box's built-in validation messages are used. The default value is **null**.
- It's possible to specify a value for `constraints.regex` when `constraints.required` is set to **false**. In this scenario, a value isn't required for the text box to validate successfully. If one is specified, it must match the regular expression pattern.

Next steps

- For an introduction to creating UI definitions, see Getting started with CreateUiDefinition.
- For a description of common properties in UI elements, see CreateUiDefinition elements.

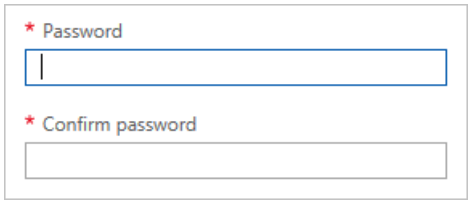
Microsoft.Compute.CredentialsCombo UI element

1/2/2020 • 2 minutes to read • Edit Online

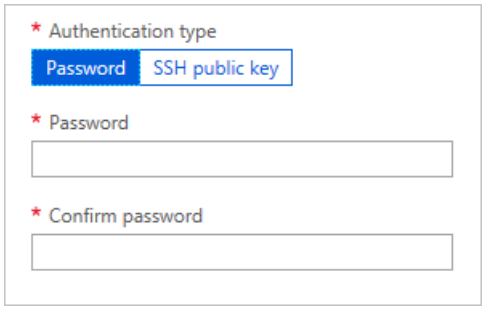
A group of controls with built-in validation for Windows and Linux passwords and SSH public keys.

UI sample

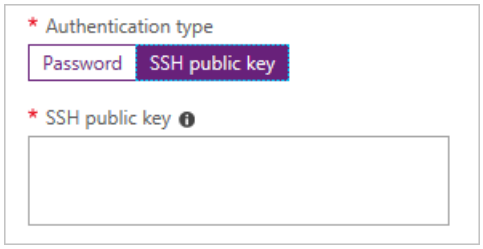
For Windows, users see:



For Linux with password selected, users see:



For Linux with SSH public key selected, users see:



Schema

For Windows, use the following schema:

```
{
  "name": "element1",
  "type": "Microsoft.Compute.CredentialsCombo",
  "label": {
    "password": "Password",
    "confirmPassword": "Confirm password"
  },
  "toolTip": {
    "password": ""
  },
  "constraints": {
    "required": true,
    "customPasswordRegex": "^(?=.*[A-Za-z])(?=.*\\d)[A-Za-z\\d]{12,}$",
    "customValidationMessage": "The password must be alphanumeric, contain at least 12 characters, and have at
least 1 letter and 1 number."
  },
  "options": {
    "hideConfirmation": false
  },
  "osPlatform": "Windows",
  "visible": true
}
```

For **Linux**, use the following schema:

```
{
  "name": "element1",
  "type": "Microsoft.Compute.CredentialsCombo",
  "label": {
    "authenticationType": "Authentication type",
    "password": "Password",
    "confirmPassword": "Confirm password",
    "sshPublicKey": "SSH public key"
  },
  "toolTip": {
    "authenticationType": "",
    "password": "",
    "sshPublicKey": ""
  },
  "constraints": {
    "required": true,
    "customPasswordRegex": "^(?=.*[A-Za-z])(?=.*\\d)[A-Za-z\\d]{12,}$",
    "customValidationMessage": "The password must be alphanumeric, contain at least 12 characters, and have at
least 1 letter and 1 number."
  },
  "options": {
    "hideConfirmation": false,
    "hidePassword": false
  },
  "osPlatform": "Linux",
  "visible": true
}
```

Sample output

If `osPlatform` is **Windows**, or `osPlatform` is **Linux** and the user provided a password instead of an SSH public key, the control returns the following output:

```
{
  "authenticationType": "password",
  "password": "p4ssw0rddem0",
}
```

If `osPlatform` is **Linux** and the user provided an SSH public key, the control returns the following output:

```
{
  "authenticationType": "sshPublicKey",
  "sshPublicKey":
"AAAAB3NzaC1yc2EAAAABIwAAAIEA1on8gxCGJJWSRT4uOrR13mUaUk0hRf4RzxSZ1zRbYYFw8pfGesIFoEuVth4HKyF8k1y4mRUnYHP1XNMNM
Jl1JcEArC2asV8sHf6zSPVffozZ5TT4SfsUu/iKy9lUcCfXzwre4WWZSXXcPff+EHtWshahu3WzBdnGxm5Xoi89zcE=",
}
```

Remarks

- `osPlatform` must be specified, and can be either **Windows** or **Linux**.
- If `constraints.required` is set to **true**, then the password or SSH public key text boxes must have values to validate successfully. The default value is **true**.
- If `options.hideConfirmation` is set to **true**, then the second text box for confirming the user's password is hidden. The default value is **false**.
- If `options.hidePassword` is set to **true**, then the option to use password authentication is hidden. It can be used only when `osPlatform` is **Linux**. The default value is **false**.
- Additional constraints on the allowed passwords can be implemented by using the `customPasswordRegex` property. The string in `customValidationMessage` is displayed when a password fails custom validation. The default value for both properties is **null**.

Next steps

- For an introduction to creating UI definitions, see Getting started with CreateUiDefinition.
- For a description of common properties in UI elements, see CreateUiDefinition elements.

Microsoft.Compute.SizeSelector UI element

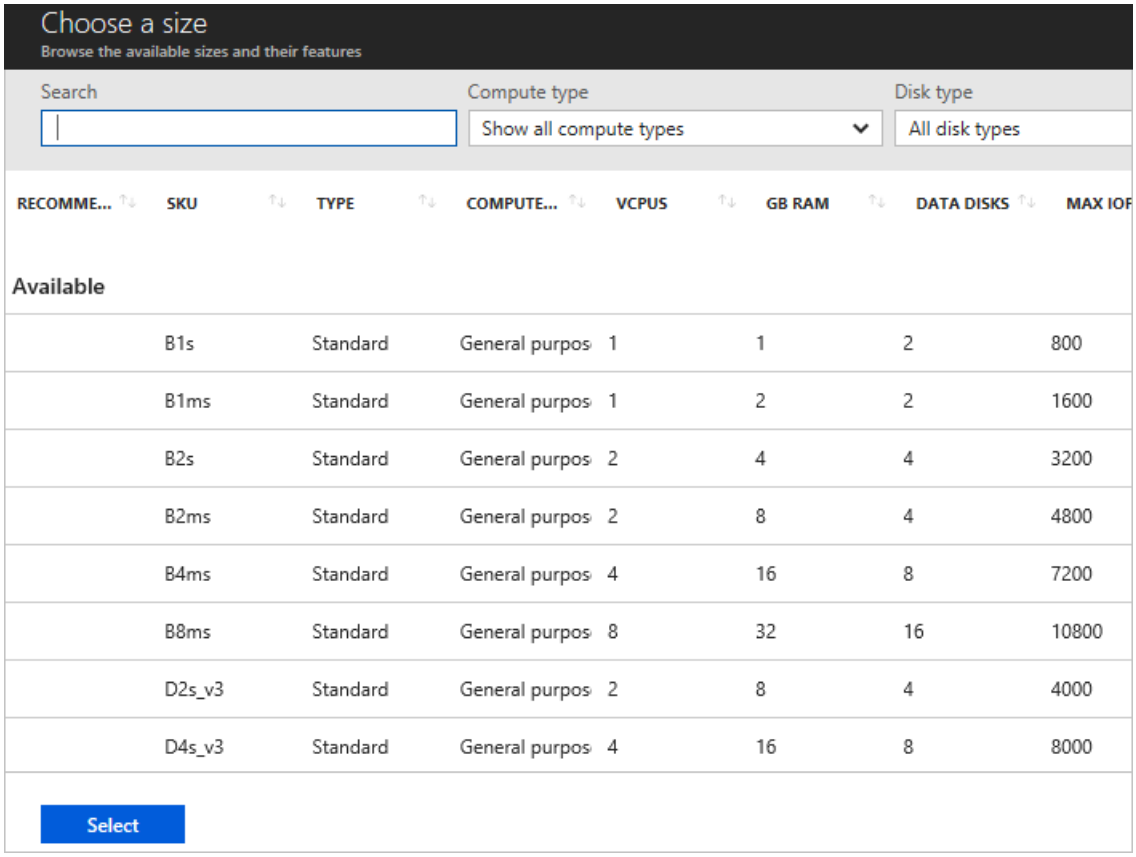
1/2/2020 • 2 minutes to read • Edit Online

A control for selecting a size for one or more virtual machine instances.

UI sample

The user sees a selector with default values from the element definition.

After selecting the control, the user sees an expanded view of the available sizes.



Schema

```
{
  "name": "element1",
  "type": "Microsoft.Compute.SizeSelector",
  "label": "Size",
  "toolTip": "",
  "recommendedSizes": [
    "Standard_D1",
    "Standard_D2",
    "Standard_D3"
  ],
  "constraints": {
    "allowedSizes": [],
    "excludedSizes": [],
    "numAvailabilityZonesRequired": 3,
    "zone": "3"
  },
  "options": {
    "hideDiskTypeFilter": false
  },
  "osPlatform": "Windows",
  "imageReference": {
    "publisher": "MicrosoftWindowsServer",
    "offer": "WindowsServer",
    "sku": "2012-R2-Datacenter"
  },
  "count": 2,
  "visible": true
}
```

Sample output

```
"Standard_D1"
```

Remarks

- `recommendedSizes` should have at least one size. The first recommended size is used as the default. The list of available sizes isn't sorted by the recommended state. The user can select that column to sort by recommended state.
- If a recommended size isn't available in the selected location, the size is automatically skipped. Instead, the next recommended size is used.
- `constraints.allowedSizes` and `constraints.excludedSizes` are both optional, but can't be used simultaneously. The list of available sizes can be determined by calling List available virtual machine sizes for a subscription. Any size not specified in the `constraints.allowedSizes` is hidden, and any size not specified in `constraints.excludedSizes` is shown.
- `osPlatform` must be specified, and can be either **Windows** or **Linux**. It's used to determine the hardware costs of the virtual machines.
- `imageReference` is omitted for first-party images, but provided for third-party images. It's used to determine the software costs of the virtual machines.
- `count` is used to set the appropriate multiplier for the element. It supports a static value, like **2**, or a dynamic value from another element, like `[steps('step1').vmCount]`. The default value is **1**.
- The `numAvailabilityZonesRequired` can be 1, 2, or 3.
- By default, `hideDiskTypeFilter` is **false**. The disk type filter enables the user to see all disk types or only SSD.

Next steps

- For an introduction to creating UI definitions, see Getting started with CreateUiDefinition.
- For a description of common properties in UI elements, see CreateUiDefinition elements.

Microsoft.Compute.UserNameTextBox UI element

1/2/2020 • 2 minutes to read • Edit Online

A text box control with built-in validation for Windows and Linux user names.

UI sample

Schema

```
{
  "name": "element1",
  "type": "Microsoft.Compute.UserNameTextBox",
  "label": "User name",
  "defaultValue": "",
  "toolTip": "",
  "constraints": {
    "required": true,
    "regex": "^[a-z0-9A-Z]{1,30}$",
    "validationMessage": "Only alphanumeric characters are allowed, and the value must be 1-30 characters
long."
  },
  "osPlatform": "Windows",
  "visible": true
}
```

Sample output

```
"Example name"
```

Remarks

- If `constraints.required` is set to **true**, then the text box must have a value to validate successfully. The default value is **true**.
- `osPlatform` must be specified, and can be either **Windows** or **Linux**.
- `constraints.regex` is a JavaScript regular expression pattern. If specified, then the text box's value must match the pattern to validate successfully. The default value is **null**.
- `constraints.validationMessage` is a string to display when the text box's value fails the validation specified by `constraints.regex`. If not specified, then the text box's built-in validation messages are used. The default value is **null**.
- This element has built-in validation that is based on the value specified for `osPlatform`. The built-in validation can be used along with a custom regular expression. If a value for `constraints.regex` is specified, then both the built-in and custom validations are triggered.

Next steps

- For an introduction to creating UI definitions, see Getting started with CreateUiDefinition.
- For a description of common properties in UI elements, see CreateUiDefinition elements.

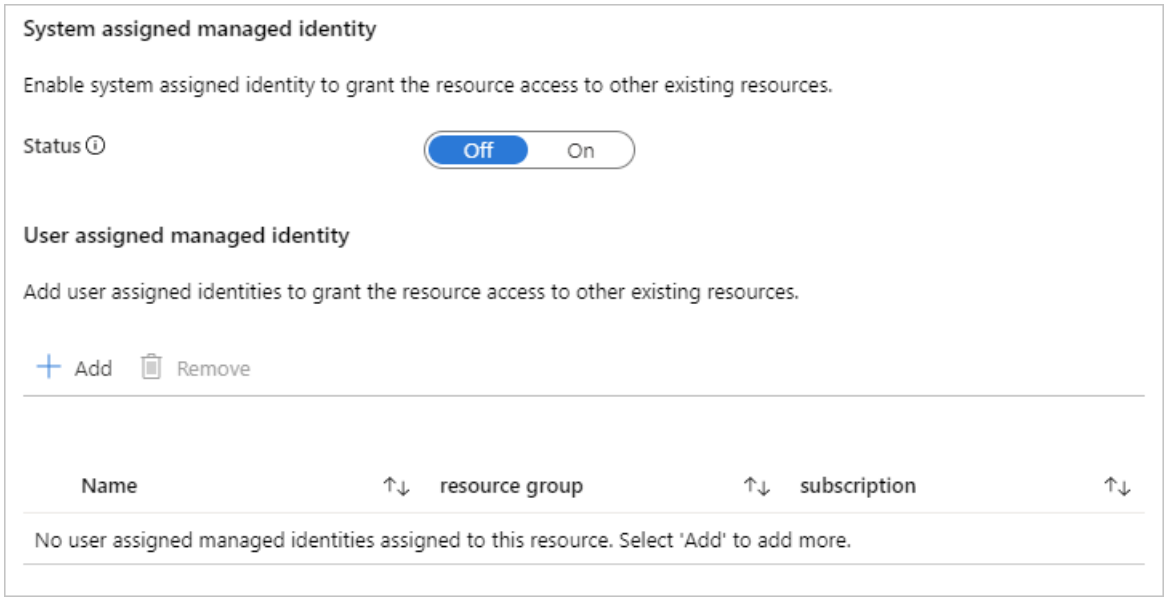
Microsoft.ManagedIdentity.IdentitySelector UI element

2/7/2020 • 2 minutes to read • Edit Online

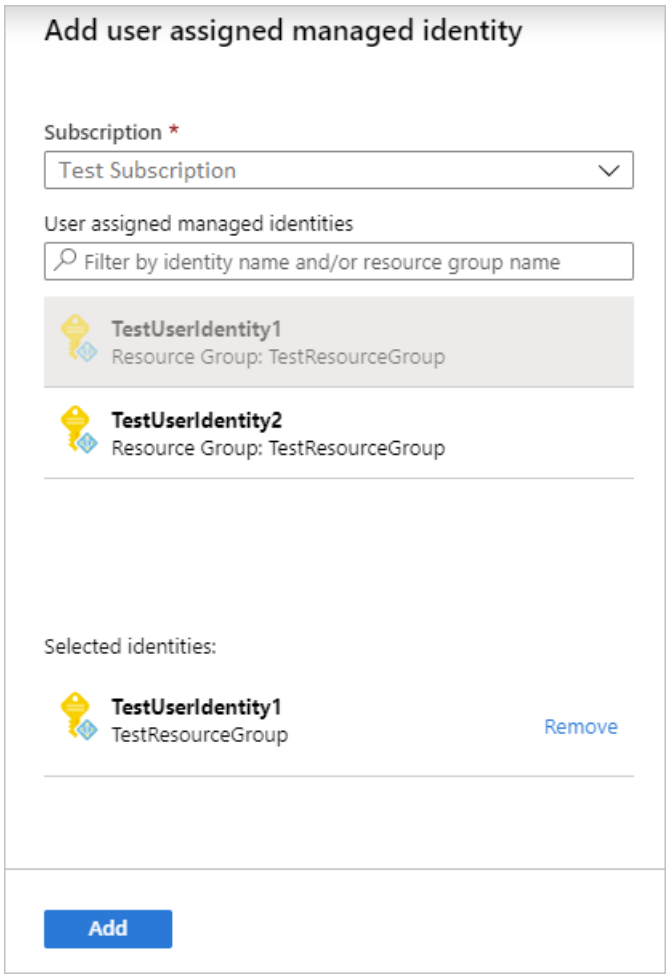
A control for assigning managed identities for a resource in a deployment.

UI sample

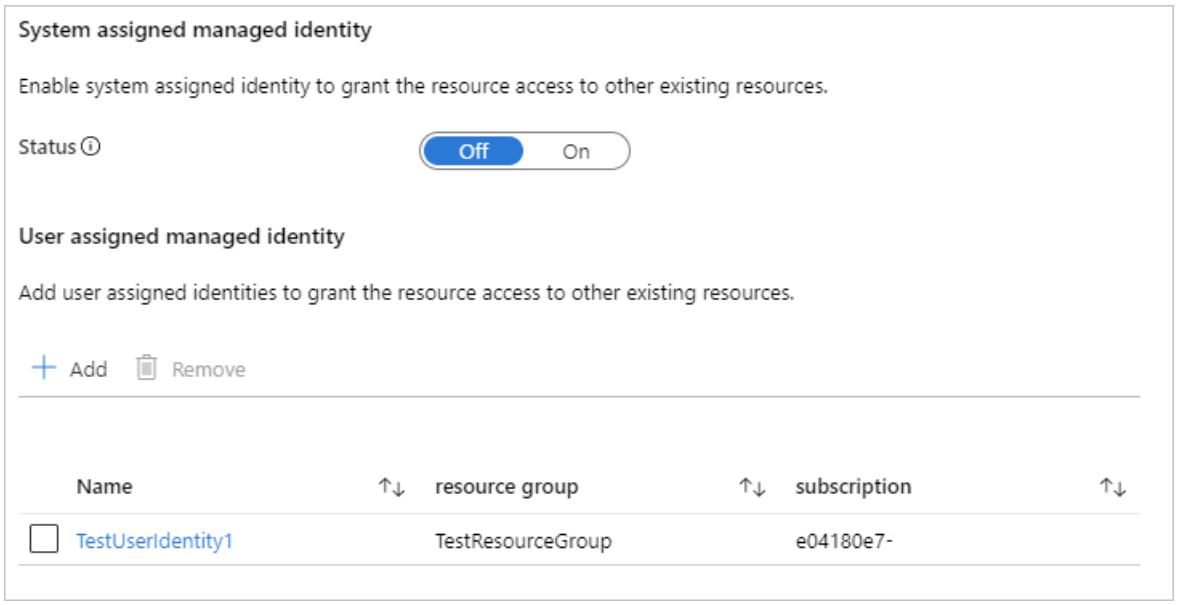
The control consists of the following elements:



When the user selects **Add**, the following form opens. The user can select one or more user-assigned identities for the resource.



The selected identities are displayed in the table. The user can add or delete items from this table.



Schema

```
{
  "name": "identity",
  "type": "Microsoft.ManagedIdentity.IdentitySelector",
  "label": "Managed Identity Configuration",
  "toolTip": {
    "systemAssignedIdentity": "Enable system assigned identity to grant the resource access to other existing
resources.",
    "userAssignedIdentity": "Add user assigned identities to grant the resource access to other existing
resources."
  },
  "defaultValue": {
    "systemAssignedIdentity": "Off"
  },
  "options": {
    "hideSystemAssignedIdentity": false,
    "hideUserAssignedIdentity": false
  },
  "visible": true
}
```

Sample output

```
{
  "identity": {
    "value": {
      "type": "UserAssigned",
      "userAssignedIdentities": {

"/subscriptions/xxxx/resourceGroups/TestResourceGroup/providers/Microsoft.ManagedIdentity/userAssignedIdentiti
es/TestUserIdentity1": {}
      }
    }
  }
}
```

Remarks

- Use **defaultValue.systemAssignedIdentity** to set an initial value for the system assigned identity options control. The default value is **Off**. The following values are allowed:
  - **On** – A system assigned identity is assigned to the resource.
  - **Off** – A system assigned identity isn't assigned to the resource.
  - **OnOnly** – A system assigned identity is assigned to the resource. Users can't edit this value during deployment.
  - **OffOnly** – A system assigned identity isn't assigned to the resource. Users can't edit this value during deployment.
- If **options.hideSystemAssignedIdentity** is set to **true**, the UI to configure the system assigned identity isn't displayed. The default value for this option is **false**.
- If **options.hideUserAssignedIdentity** is set to **true**, the UI to configure the user assigned identity isn't displayed. The resource isn't assigned a user assigned identity. The default value for this option is **false**.

Next steps

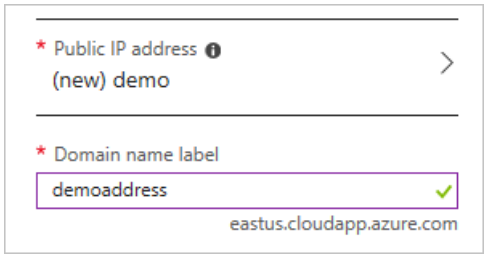
- For an introduction to creating UI definitions, see Getting started with CreateUiDefinition.
- For a description of common properties in UI elements, see CreateUiDefinition elements.

Microsoft.Network.PublicIpAddressCombo UI element

1/2/2020 • 2 minutes to read • Edit Online

A group of controls for selecting a new or existing public IP address.

UI sample



- If the user selects 'None' for public IP address, the domain name label text box is hidden.
- If the user selects an existing public IP address, the domain name label text box is disabled. Its value is the domain name label of the selected IP address.
- The domain name suffix (for example, westus.cloudapp.azure.com) updates automatically based on the selected location.

Schema

```
{
  "name": "element1",
  "type": "Microsoft.Network.PublicIpAddressCombo",
  "label": {
    "publicIpAddress": "Public IP address",
    "domainNameLabel": "Domain name label"
  },
  "toolTip": {
    "publicIpAddress": "",
    "domainNameLabel": ""
  },
  "defaultValue": {
    "publicIpAddressName": "ip01",
    "domainNameLabel": "mydomain"
  },
  "constraints": {
    "required": {
      "domainNameLabel": true
    }
  },
  "options": {
    "hideNone": false,
    "hideDomainNameLabel": false,
    "hideExisting": false,
    "zone": 3
  },
  "visible": true
}
```

Sample output

If the user selects no public IP address, the control returns the following output:

```
{
  "newOrExistingOrNone": "none"
}
```

If the user selects a new or existing IP address, the control returns the following output:

```
{
  "name": "ip01",
  "resourceGroup": "rg01",
  "domainNameLabel": "mydomain",
  "publicIPAllocationMethod": "Dynamic",
  "sku": "Basic",
  "newOrExistingOrNone": "new"
}
```

- When `options.hideNone` is specified as **true**, `newOrExistingOrNone` will only have a value of **new** or **existing**.
- When `options.hideDomainNameLabel` is specified as **true**, `domainNameLabel` is undeclared.

Remarks

- If `constraints.required.domainNameLabel` is set to **true**, the user must provide a domain name label when creating a new public IP address. Existing public IP addresses without a label aren't available for selection.
- If `options.hideNone` is set to **true**, then the option to select **None** for the public IP address is hidden. The default value is **false**.
- If `options.hideDomainNameLabel` is set to **true**, then the text box for domain name label is hidden. The default value is **false**.
- If `options.hideExisting` is true, then the user isn't able to choose an existing public IP address. The default value is **false**.
- For `zone`, only public IP addresses for the specified zone or zone resilient public IP addresses are available.

Next steps

- For an introduction to creating UI definitions, see Getting started with CreateUiDefinition.
- For a description of common properties in UI elements, see CreateUiDefinition elements.

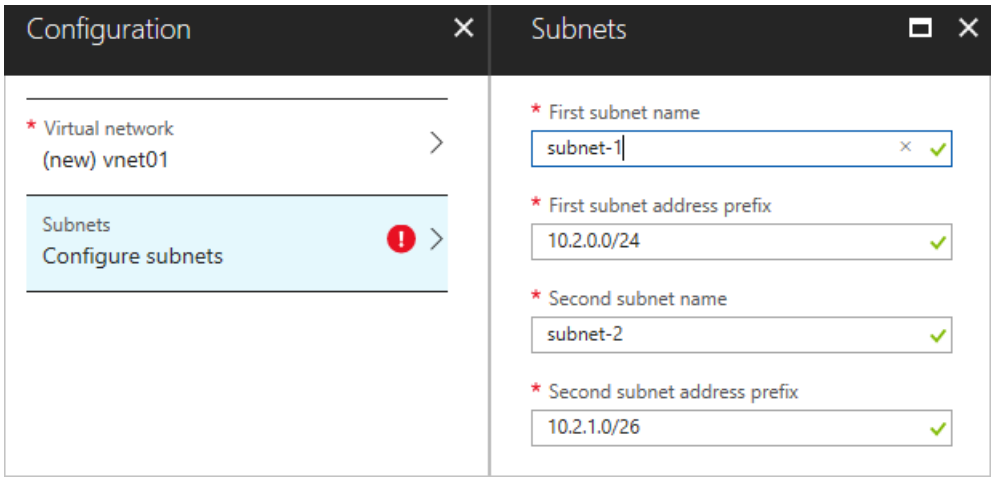
Microsoft.Network.VirtualNetworkCombo UI element

1/2/2020 • 2 minutes to read • Edit Online

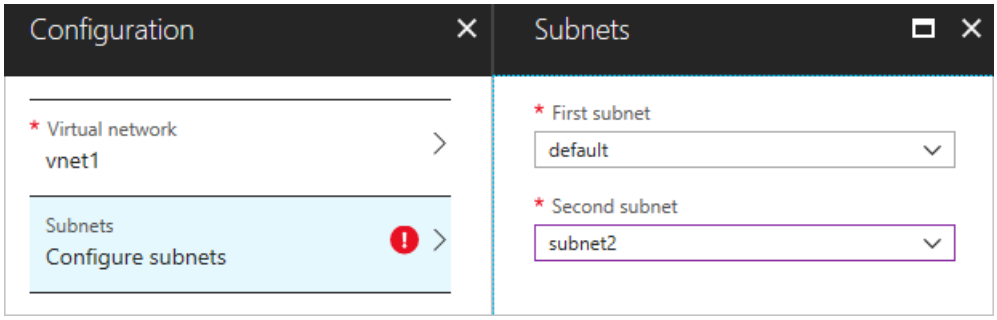
A group of controls for selecting a new or existing virtual network.

UI sample

When the user picks a new virtual network, the user can customize each subnet's name and address prefix. Configuring subnets is optional.



When the user picks an existing virtual network, the user must map each subnet the deployment template requires to an existing subnet. Configuring subnets in this case is required.



Schema

```
{
  "name": "element1",
  "type": "Microsoft.Network.VirtualNetworkCombo",
  "label": {
    "virtualNetwork": "Virtual network",
    "subnets": "Subnets"
  },
  "toolTip": {
    "virtualNetwork": "",
    "subnets": ""
  },
  "defaultValue": {
    "name": "vnet01",
    "addressPrefixSize": "/16"
  },
  "constraints": {
    "minAddressPrefixSize": "/16"
  },
  "options": {
    "hideExisting": false
  },
  "subnets": {
    "subnet1": {
      "label": "First subnet",
      "defaultValue": {
        "name": "subnet-1",
        "addressPrefixSize": "/24"
      },
      "constraints": {
        "minAddressPrefixSize": "/24",
        "minAddressCount": 12,
        "requireContiguousAddresses": true
      }
    },
    "subnet2": {
      "label": "Second subnet",
      "defaultValue": {
        "name": "subnet-2",
        "addressPrefixSize": "/26"
      },
      "constraints": {
        "minAddressPrefixSize": "/26",
        "minAddressCount": 8,
        "requireContiguousAddresses": true
      }
    }
  },
  "visible": true
}
```

Sample output

```
{
  "name": "vnet01",
  "resourceGroup": "rg01",
  "addressPrefixes": ["10.0.0.0/16"],
  "newOrExisting": "new",
  "subnets": {
    "subnet1": {
      "name": "subnet-1",
      "addressPrefix": "10.0.0.0/24",
      "startAddress": "10.0.0.1"
    },
    "subnet2": {
      "name": "subnet-2",
      "addressPrefix": "10.0.1.0/26",
      "startAddress": "10.0.1.1"
    }
  }
}
```

Remarks

- If specified, the first non-overlapping address prefix of size `defaultValue.addressPrefixSize` is determined automatically based on the existing virtual networks in the user's subscription.
- The default value for `defaultValue.name` and `defaultValue.addressPrefixSize` is **null**.
- `constraints.minAddressPrefixSize` must be specified. Any existing virtual networks with an address space smaller than the specified value are unavailable for selection.
- `subnets` must be specified, and `constraints.minAddressPrefixSize` must be specified for each subnet.
- When creating a new virtual network, each subnet's address prefix is calculated automatically based on the virtual network's address prefix and the respective `addressPrefixSize`.
- When using an existing virtual network, any subnets smaller than the respective `constraints.minAddressPrefixSize` are unavailable for selection. Additionally, if specified, subnets that don't have at least `minAddressCount` available addresses are unavailable for selection. The default value is **0**. To ensure that the available addresses are contiguous, specify **true** for `requireContiguousAddresses`. The default value is **true**.
- Creating subnets in an existing virtual network isn't supported.
- If `options.hideExisting` is **true**, the user can't choose an existing virtual network. The default value is **false**.

Next steps

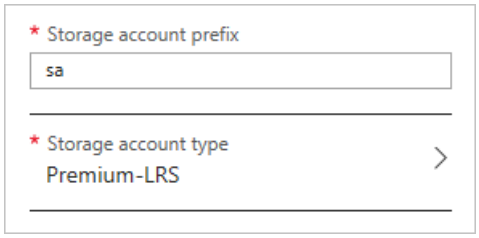
- For an introduction to creating UI definitions, see Getting started with CreateUiDefinition.
- For a description of common properties in UI elements, see CreateUiDefinition elements.

Microsoft.Storage.MultiStorageAccountCombo UI element

1/2/2020 • 2 minutes to read • Edit Online

A group of controls for creating several storage accounts with names that start with a common prefix.

UI sample



Schema

```
{
  "name": "element1",
  "type": "Microsoft.Storage.MultiStorageAccountCombo",
  "label": {
    "prefix": "Storage account prefix",
    "type": "Storage account type"
  },
  "toolTip": {
    "prefix": "",
    "type": ""
  },
  "defaultValue": {
    "prefix": "sa",
    "type": "Premium_LRS"
  },
  "constraints": {
    "allowedTypes": [],
    "excludedTypes": []
  },
  "count": 2,
  "visible": true
}
```

Sample output

```
{
  "prefix": "sa",
  "count": 2,
  "resourceGroup": "rg01",
  "type": "Premium_LRS"
}
```

Remarks

- The value for `defaultValue.prefix` is concatenated with one or more integers to generate the sequence of

storage account names. For example, if `defaultValue.prefix` is **sa** and `count` is **2**, then storage account names **sa1** and **sa2** are generated. Generated storage account names are validated for uniqueness automatically.

- The storage account names are generated lexicographically based on `count`. For example, if `count` is 10, then the storage account names end with two-digit integers (01, 02, 03).
- The default value for `defaultValue.prefix` is **null**, and for `defaultValue.type` is **Premium_LRS**.
- Any type not specified in `constraints.allowedTypes` is hidden, and any type not specified in `constraints.excludedTypes` is shown. `constraints.allowedTypes` and `constraints.excludedTypes` are both optional, but can't be used simultaneously.
- In addition to generating storage account names, `count` is used to set the appropriate multiplier for the element. It supports a static value, like **2**, or a dynamic value from another element, like `[steps('step1').storageAccountCount]`. The default value is **1**.

Next steps

- For an introduction to creating UI definitions, see Getting started with CreateUiDefinition.
- For a description of common properties in UI elements, see CreateUiDefinition elements.

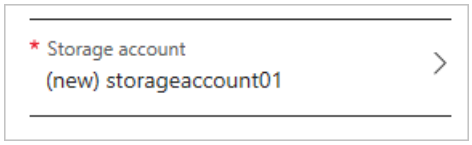
Microsoft.Storage.StorageAccountSelector UI element

1/2/2020 • 2 minutes to read • Edit Online

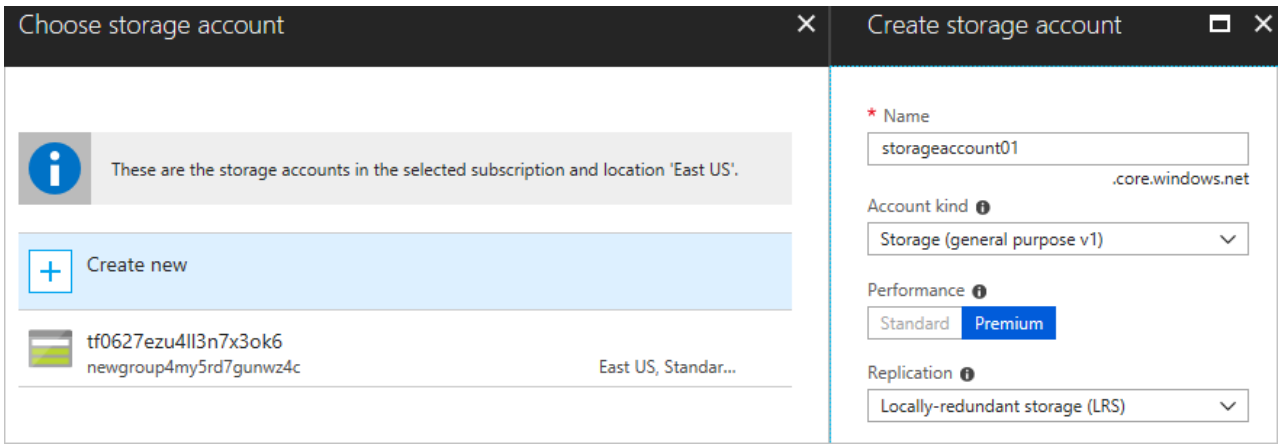
A control for selecting a new or existing storage account.

UI sample

The control shows the default value.



The control enables the user to create a new storage account or select an existing storage account.



Schema

```
{
  "name": "element1",
  "type": "Microsoft.Storage.StorageAccountSelector",
  "label": "Storage account",
  "toolTip": "",
  "defaultValue": {
    "name": "storageaccount01",
    "type": "Premium_LRS"
  },
  "constraints": {
    "allowedTypes": [],
    "excludedTypes": []
  },
  "options": {
    "hideExisting": false
  },
  "visible": true
}
```

Sample output

```
{
  "name": "storageaccount01",
  "resourceGroup": "rg01",
  "type": "Premium_LRS",
  "newOrExisting": "new"
}
```

Remarks

- If specified, `defaultValue.name` is automatically validated for uniqueness. If the storage account name isn't unique, the user must specify a different name or choose an existing storage account.
- The default value for `defaultValue.type` is **Premium_LRS**.
- Any type not specified in `constraints.allowedTypes` is hidden, and any type not specified in `constraints.excludedTypes` is shown. `constraints.allowedTypes` and `constraints.excludedTypes` are both optional, but can't be used simultaneously.
- If `options.hideExisting` is **true**, the user can't choose an existing storage account. The default value is **false**.

Next steps

- For an introduction to creating UI definitions, see Getting started with CreateUiDefinition.
- For a description of common properties in UI elements, see CreateUiDefinition elements.