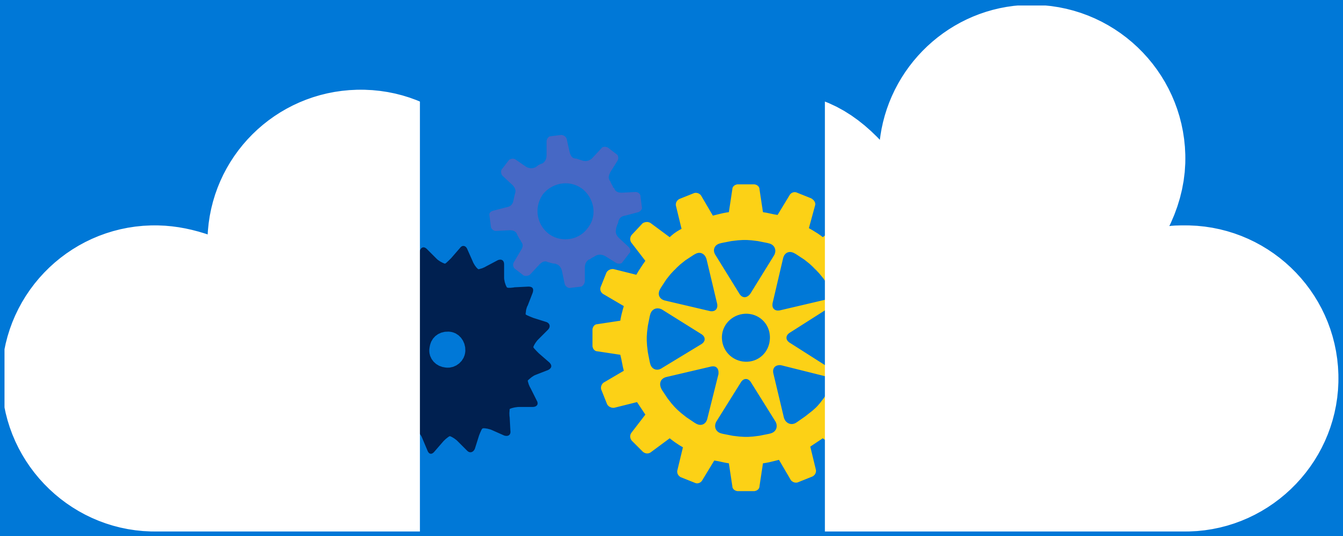
Microsoft



Using Azure Management APIs to get data about your deployed resources

By Abhisek Banerjee, Rangarajan Srirangam, Mandar Inamdar, and Rakesh Patil
Azure Customer Advisory Team (AzureCAT)

June 2017

Contents

Authored by Abhisek Banerjee, Rangarajan Srirangam, Mandar Inamdar, and Rakesh Patil. Edited by Nanette Ray, Tycen Hopkins, and Mark Simms. Reviewed by Mark Simms, Manbeen Kohli, and Mike Wasson.

Overview

This article and the accompanying sample files show you how to collect information about your deployed resources in Azure using data from the Azure management APIs, suitable for reporting and automation. See the GitHub repo at Azure Management API IaaS VM Inventory Sample Scripts.

We walk through an approach based on retrieving the raw information (in this case, a list of virtual machines). Then we use scripts to convert the information into a report-friendly format such as a CSV file that you can easily review in Microsoft Excel or another program. You can extend this extract-transform approach to other services as well.

Azure offers two command environments and two deployment models. In this walk-through, we use PowerShell cmdlets and the Azure command line interface (CLI) to access Azure data, working against Azure Resource Manager resources. The commands vary, particularly if you're working with a classic Azure Service Manager (ASM) deployment. (See the comparison table at the end of this article.) Since a single Azure subscription might include both Resource Manager and Service Manager resources, we include sample files for these combinations.

Work with Azure data

This walk-through uses an example deployment of Azure VMs. Using the sample scripts, we will generate a list of VMs and associated disks that allows you to examine the deployed resources. You can then extrapolate this approach to include other information as needed.

In this example, we want to obtain a list of deployed VMs and ensure that all of them are deployed in availability sets (with at least two VMs in an availability set) for resiliency. The sample scripts will extract the relevant data from the management APIs, and examine:

- Are any VMs not in availability sets? Which aren't in availability sets?
- Do any availability sets have only one VM? Which have a single VM?

This logic can also be modified for different scenarios. For example, although the best practice is to remove single points of failure through redundancy and resilient design, Azure also offers an SLA for single-instance VMs using premium storage for all disks. If you plan to deploy a combination of VMs in availability sets plus single-instance VMs on premium storage, you could modify the script logic to check for either condition.

Step 1: Extract

The first step in our walk-through shows how to execute commands to connect to an Azure subscription and extract the data you want. With just a few commands, you can fetch detailed information about VMs and storage accounts in the JSON format. The idea is to retrieve this information up front so your extraction routine can remain virtually unchanged even as you add more logic later.

Example scripts showing how perform this extraction are included for Windows using PowerShell (**\samplecode\extract\windows\extract.ps1**) and Linux using Bash (**\samplecode\extract\linux\extract.sh**).
We also include sample files containing dumps of information extracted through the extraction commands. These JSON files are named for the deployment mode (ARM or ASM), command line (PS or CLI), and object (VM or SA). For example, **armpssa.json** contains sample data for Azure Resource Manager storage accounts extracted using PowerShell.

NOTE: When creating or modifying your own scripts, consider that each call to an Azure API has a time-based quota; calls to Azure APIs can be made only so many times within a certain period. For details about these limits, see Throttling Resource Manager requests.

If your scripts receive a throttling response, please ensure that they back off for a period of time to prevent affecting other operations against your Azure subscription.

A. Log on

To extract the necessary information about an Azure deployment, you need to execute commands that log on to Azure, choose a deployment mode, select a subscription, and fetch data.

CLI 2.0

In CLI 2.0 you can log on to a subscription using:

```
az login
```

When you run the command without additional options, `azure login` prompts you to continue logging on interactively through a web portal. But multiple scenarios are supported. For more information, see Log in to Azure from the Azure CLI.

CLI 1.0

In CLI 1.0, you can log on to a subscription using:

```
azure login
```

PowerShell

In PowerShell, you can have two different commands for adding an authenticated account:

To add an authenticated account for use with Resource Manager cmdlets, execute:

```
Add-AzureRMAccount
```

This command also provides for multiple login scenarios. For more information, refer to the syntax documentation for Add-AzureRmAccount.

To add an authenticated account for use with Service Management cmdlets, execute:

```
Add-AzureAccount
```

B. Choose deployment mode

In CLI 1.0, you choose a deployment mode using the following command:

```
azure config mode arm
```

You don’t have to set a deployment mode for CLI 2.0 preview, which supports only ARM, or in PowerShell, where the commands that work on resources are distinctly named for ARM and ASM.

C. Select a subscription

Since you might have more than one Azure subscription, you need to choose from among them.

CLI 2.0

In CLI 2.0, you choose a subscription using:

```
az account set --subscription "SubscriptionId"
```

The Subscription ID is not needed in subsequent commands.

CLI 1.0

In CLI 1.0, you pass the subscription ID to each command that works on resources using the parameter:

```
--subscription
```

PowerShell

In PowerShell, you select an ARM subscription using either the `Set-AzureRmContext` or `Select-AzureRmSubscription` command:

```
Set-AzureRmContext -SubscriptionId $SubscriptionID

Select-AzureSubscription -SubscriptionId $SubscriptionID -Current
```

And an ASM subscription:

```
Set-AzureContext -SubscriptionId $SubscriptionID

Select-AzureSubscription -SubscriptionId $SubscriptionID -Current
```

D. Fetch VM and storage account information

The next step is to fetch the VM and storage account details.

CLI 2.0

To fetch VM and storage account details, in CLI 2.0 you execute the following commands:

```
az vm list

az storage account list
```

To write the output of the commands to local files, use the output redirection operator (>). To save the data in the JSON notation, use the `--json` parameter in CLI.

CLI 1.0

In CLI 1.0, the corresponding commands are:

```
azure vm list --subscription $SubscriptionId

azure storage account list --subscription $SubscriptionId
```

PowerShell

In PowerShell, the corresponding commands for ARM resources are:

```
Get-AzureRmVM

Get-AzureRmStorageAccount
```

While the ASM commands are:

```
Get-AzureVM

Get-AzureStorageAccount
```

To write the output of the commands to local files, use the `Out-File` and `FilePath` options. These commands also provide the option of saving the data in the JSON notation (pipe to `ConvertTo-Json` option in PowerShell).

Step 2: Transform

This walk-through includes a sample transform program (\samplecode\transform\transform.py) that converts the extracted data into a common schema that can be used for deeper analysis. By transforming the extracted data, you can independently revise the steps used for extraction and analysis regardless of which deployment model or command set you're working with.

For example, in ARM deployments, the JSON structures produced by PowerShell and CLI commands differ. For classic deployments, PowerShell and CLI outputs differ even more. In addition, certain resources such as Affinity Groups exist only in the classic model, while other, such as Resource Groups, exist only in ARM. The sample creates a common schema in two parts—one for each deployment model.

To illustrate this walk-through, we use CLI 2.0 to create a common, canonical schema. See the \samplecode\transform\transformeddata.json file. For consistency, some of the ASM data has been populated in the fields like their ARM counterpart.

We also added a few useful metadata fields to the CLI structure to illustrate how the transform step can be used to filter the data for analysis. These fields are `version`, `creationDate`, `subscriptionId`, `subscriptionName`, and `ARM` (or `ASM`). You can add more fields as needed or even filter or aggregate some of the fields depending on your analysis needs.

Step 3: Generate inventory

Having extracted the data and transformed it to a common format, the next step in this walk-through is to examine the results. Our sample inventory program (\samplecode\inventory\inventory.py) uses a Python script that reads the transformeddata.json file and produces output showing information about the subscription's VMs and disks in CSV format.

To extend this sample analysis, you can add rules. Each set of rule is a smaller Python program in the \samplecode\inventory\rules folder and configured in the \samplecode\inventory\rulesconfig.json file. To generate an inventory for a subscription, the main inventory program calls the individual programs configured as rules and aggregates their output.

The \samplecode\inventory\rules folder provides the following code samples for analyzing important parameters of VMs and storage accounts in a subscription:

- The vminventory.py file contains sample code that analyzes the virtual machines within a subscription.
- The storageinventory.py file contains sample code that analyzes the storage accounts within a subscription.
- To see sample output of the inventory program for a particular subscription, see the storageAccountInventory.csv and vmInventory.csv files in the \samplecode\sampledata folder.

Run the sample code

You can run the sample code used for this article from Windows or Linux environments.
In Windows, you need the following:

- Latest version of PowerShell
- Latest version of Azure PowerShell modules
- Python 3

In Linux, you need the following:

- Bash
- Azure CLI
- Python 3

Run the sample code in Windows

1. Sign in to a Windows PC configured with the required software.
2. Identify the Azure subscription you want to work with, and keep the Azure subscription ID handy for further use.
3. Extract the sample code to a local folder on the PC.

4. Set PowerShell execution policies to allow execution of unsigned scripts using the following command:

   ```
   Set-ExecutionPolicy -Scope LocalMachine Unrestricted
   ```

5. Run:

   ```
   extract.ps1 <subscriptionId>
   ```

   A prompt appears to log on to the Azure subscription twice, once for ASM mode and again for ARM mode. At the end of extraction, the extracted data is available in the **data/<subscriptionId>** folder as .json files.

6. Run:

   ```
   transform.py <subscriptionId>
   ```

   At the end of the execution, the **transformeddata.json** file is available in the **data/<subscriptionId>** folder.

7. Run:

   ```
   inventory.py <subscriptionId>
   ```

   After the cmdlet executes, the **vminventory.csv** file is available in the **data/<subscriptionId>** folder

8. Review the **vminventory.csv** file for details or use it as the source for further reporting in Excel, Microsoft PowerBI, or similar tools.

Run the sample code in Linux

1. Log on to a Linux PC configured with the required software.
2. Identify the Azure subscription you want to analyze, and keep the Azure subscription ID handy for further use.
3. Extract the sample code to a local folder on the PC.
4. Run:

   ```
   "cd samplecode/extract/linux"
   ```

5. Run:

   ```
   chmod +x extract.sh
   ```

6. Log on to your subscription using either the `#azure login` or `#az login` command. In a browser, go to https://aka.ms/devicelogin and enter the arbitrary code provided on the terminal screen.
7. On the command line, run the script:

   ```
   #./extract.sh <subscriptionId>
   ```

   When the extraction is complete, the extracted data is available in the **data/<subscriptionId;gt;** folder as .json files.

8. Run **transform.py** using this command:

   ```
   #python transform.py <subscriptionId>
   ```

   When the execution is complete, the **transformeddata.json** file is available in the **data/<subscriptionId>** folder.

9. Create the inventory using this command:

   ```
   inventory.py <subscriptionId>
   ```

   The CSV output file is available in the **data/<subscriptionId>** folder.

10. Review the **vminventory.csv** file for details or use it as the source for further reporting in Excel, PowerBI, or similar tools.

Comparing cmdlets in ARM and ASM

You can use CLI 1.0 to work with a deployment based on ARM or classic Azure (ASM). CLI 2.0, released in February 2017, works with ARM only and includes many useful new features. To understand the differences between these deployment models, please refer to Azure Resource Manager vs. classic deployment.

In an Azure Resource Manager deployment, the following commands are comparable:

Task	CLI 1.0	CLI 2.0	PS
Log In	`azure login`	`az login`	`az login`
Add Account	N/A	N/A	`Add-AzureRMAccount`
Set Mode	`azure config mode arm`	N/A	N/A
Set Subscription	N/A	`az account set --subscription "SubscriptionId"`	`az Set-AzureRmContext -SubscriptionId $SubscriptionID`
Get VM List	`azure vm list --subscription $SubscriptionId`	`az vm list`	`Get-AzureRmVM`
Get Storage Accts	`azure storage account list --subscription $SubscriptionId`	`az storage account list`	`Get-AzureRmStorageAccount`

In a classic Azure (ASM) deployment, the following commands are comparable:

Task	CLI 1.0	CLI 2.0	PS
Log In	`azure login`	`az login`	`az login`
Add Account	N/A	N/A	`Add-AzureAccount`
Set Mode	N/A	N/A	N/A
Set Subscription	N/A	`az account set --subscription "SubscriptionId"`	`Select-AzureSubscription -SubscriptionId $SubscriptionID -Current`
Get VM List	`azure vm list --subscription $SubscriptionId`	`az vm list`	`Get-AzureVM`
Get Storage Accts	`azure storage account list --subscription $SubscriptionId`	`az storage account list`	`Get-AzureStorageAccount`

Learn more

For more information about best practices, see the following Azure resources:

- Architectures for running VM Workloads on Azure
- Resiliency technical guidance
- Resiliency checklist
- High availability checklist
- Azure subscription and service limits, quotas, and constraints