Microsoft

Microsoft Official Course

AZ-203T01

Develop Azure Infrastructure as a Service compute solutions

AZ-203T01

Develop Azure Infrastructure as a Service compute solutions

Contents

Welcome to the course

Start Here

Welcome

Welcome to the **Develop Azure Infrastructure as a Service compute solutions** course. This course is part of a series of courses to help you prepare for the **AZ-203: Developing Solutions for Microsoft Azure**[1] certification exam.

Who should take this exam?

Candidates for this exam are Azure Developers who design and build cloud solutions such as applications and services. They participate in all phases of development, from solution design, to development and deployment, to testing and maintenance. They partner with cloud solution architects, cloud DBAs, cloud administrators, and clients to implement the solution.

Candidates should be proficient in developing apps and services by using Azure tools and technologies, including storage, security, compute, and communications.

Candidates must have at least one year of experience developing scalable solutions through all phases of software development and be skilled in at least one cloud-supported programming language.

Exam study areas

AZ-203 includes six study areas, as shown in the table. The percentages indicate the relative weight of each area on the exam. The higher the percentage, the more questions you are likely to see in that area.

AZ-203 Study Areas	Weight
Develop Azure Infrastructure as a Service compute solutions	10-15%
Develop Azure Platform as a Service compute solutions	20-25%

1 https://www.microsoft.com/en-us/learning/exam-az-203.aspx

AZ-203 Study Areas	Weight
Develop for Azure storage	15-20%
Implement Azure security	10-15%
Monitor, troubleshoot, and optimize Azure solutions	15-20%
Connect to and consume Azure, and third-party, services	20-25%

✓ This course will focus on preparing you for the **Develop Azure Infrastructure as a Service compute solutions** area of the AZ-203 certification exam.

Course description

In this course students will gain the knowledge and skills needed to implement Azure IaaS services and features in their development solutions. The course covers provisioning virtual machines, using Batch Service to deploy/maintain resources, and how to create containerized solutions by using Azure Kubernetes Service.

Level: Intermediate

Audience:

- Students in this course are interested in Azure development or in passing the Microsoft Azure Developer Associate certification exam.
- Students should have 1-2 years experience as a developer. This course assumes students know how to code and have a fundamental knowledge of Azure.
- It is recommended that students have some experience with PowerShell or Azure CLI, working in the Azure portal, and with at least one Azure-supported programming language. Most of the examples in this course are presented in C# .NET.

Course Syllabus

Module 1: Implement solutions that use virtual machines

- Provision VMs
- Create ARM templates
- Configure Azure Disk Encryption for VMs

Module 2: Implement batch jobs by using Azure Batch Services

- Azure Batch overview
- Run a batch job by using the Azure CLI and Azure Portal
- Run batch jobs by using code
- Manage batch jobs by using the Batch Service API

Module 3: Create containerized solutions

- Create an Azure Managed Kubernetes Service (AKS) cluster
- Create container images for solutions
- Publish an image to the Azure Container Registry
- Run containers by using Azure Container Instance or AKS

Implement solutions that use virtual machines

Provision VMs

Azure Virtual Machine creation checklist

Provisioning VMs to Azure requires planning. Before you create a single VM be sure you have thought about the following:

- Start with the network
- Name the VM
- Decide the location for the VM
- Determine the size of the VM
- Understanding the pricing model
- Storage for the VM
- Select an operating system

Overview of Azure Virtual Machines

Start with the network

Virtual networks (VNets) are used in Azure to provide private connectivity between Azure Virtual Machines and other Azure services. VMs and services that are part of the same virtual network can access one another. By default, services outside the virtual network cannot connect to services within the virtual network. You can, however, configure the network to allow access to the external service, including your on-premises servers.

This latter point is why you should spend some time thinking about your network configuration. Network addresses and subnets are not trivial to change once you have them set up, and if you plan to connect your private company network to the Azure services, you will want to make sure you consider the topology before putting any VMs into place.

Name the VM

One piece of information people often don't put much thought into is the name of the VM. The VM name is used as the computer name, which is configured as part of the operating system. You can specify a name of up to 15 characters on a Windows VM and 64 characters on a Linux VM.

This name also defines a manageable Azure resource, and it's not trivial to change later. That means you should choose names that are meaningful and consistent, so you can easily identify what the VM does. A good convention is to include the following information in the name:

Element	Example	Notes
Environment	dev, prod, QA	Identifies the environment for the resource
Location	uw (US West), ue (US East)	Identifies the region into which the resource is deployed
Instance	01, 02	For resources that have more than one named instance (web servers, etc.)
Product or Service	service	Identifies the product, application, or service that the resource supports
Role	sql, web, messaging	Identifies the role of the associated resource

For example, `devusc-webvm01` might represent the first development web server hosted in the US South Central location.

Decide the location for the VM

Azure has datacenters all over the world filled with servers and disks. These datacenters are grouped into geographic regions ('West US', 'North Europe', 'Southeast Asia', etc.) to provide redundancy and availability.

When you create and deploy a virtual machine, you must select a region where you want the resources (CPU, storage, etc.) to be allocated. This lets you place your VMs as close as possible to your users to improve performance and to meet any legal, compliance, or tax requirements.

Two other things to think about regarding the location choice. First, the location can limit your available options. Each region has different hardware available and some configurations are not available in all regions. Second, there are price differences between locations. If your workload isn't bound to a specific location, it can be very cost effective to check your required configuration in multiple regions to find the lowest price.

Determine the size of the VM

Once you have the name and location set, you need to decide on the size of your VM. Rather than specify processing power, memory, and storage capacity independently, Azure provides different VM sizes that offer variations of these elements in different sizes. Azure provides a wide range of VM size options allowing you to select the appropriate mix of compute, memory, and storage for what you want to do.

The best way to determine the appropriate VM size is to consider the type of workload your VM needs to run. Based on the workload, you're able to choose from a subset of available VM sizes. Workload options are classified as follows on Azure:

Option	Description
General purpose	General-purpose VMs are designed to have a balanced CPU-to-memory ratio. Ideal for testing and development, small to medium databases, and low to medium traffic web servers.
Compute optimized	Compute optimized VMs are designed to have a high CPU-to-memory ratio. Suitable for medium traffic web servers, network appliances, batch processes, and application servers.
Memory optimized	Memory optimized VMs are designed to have a high memory-to-CPU ratio. Great for relational database servers, medium to large caches, and in-memory analytics.
Storage optimized	Storage optimized VMs are designed to have high disk throughput and IO. Ideal for VMs running databases.
GPU	PU VMs are specialized virtual machines targeted for heavy graphics rendering and video editing. These VMs are ideal options for model training and inferencing with deep learning.
High performance computes	High performance compute is the fastest and most powerful CPU virtual machines with optional high-throughput network interfaces.

What if my size needs change?

Azure allows you to change the VM size when the existing size no longer meets your needs. You can upgrade or downgrade the VM - as long as your current hardware configuration is allowed in the new size. This provides a fully agile and elastic approach to VM management.

The VM size can be changed while the VM is running, as long as the new size is available in the current hardware cluster the VM is running on. The Azure portal makes this obvious by only showing you available size choices. The command line tools will report an error if you attempt to resize a VM to an unavailable size. Changing a running VM size will automatically reboot the machine to complete the request.

If you stop and deallocate the VM, you can then select any size available in your region since this removes your VM from the cluster it was running on.

Warning: Be careful about resizing production VMs - they will be rebooted automatically which can cause a temporary outage and change some configuration settings such as the IP address.

Understanding the pricing model

There are two separate costs the subscription will be charged for every VM: compute and storage. By separating these costs, you scale them independently and only pay for what you need.

Compute costs - Compute expenses are priced on a per-hour basis but billed on a per-minute basis. For example, you are only charged for 55 minutes of usage if the VM is deployed for 55 minutes. You are not charged for compute capacity if you stop and deallocate the VM since this releases the hardware. The hourly price varies based on the VM size and OS you select. The cost for a VM includes the charge for the Windows operating system. Linux-based instances are cheaper because there is no operating system license charge.

Storage costs - You are charged separately for the storage the VM uses. The status of the VM has no relation to the storage charges that will be incurred; even if the VM is stopped/deallocated and you aren't billed for the running VM, you will be charged for the storage used by the disks.

You're able to choose from two payment options for compute costs:

1. **Pay as you go** - With the pay-as-you-go option, you pay for compute capacity by the second, with no long-term commitment or upfront payments. You're able to increase or decrease compute capacity on demand as well as start or stop at any time. Prefer this option if you run applications with short-term or unpredictable workloads that cannot be interrupted. For example, if you are doing a quick test, or developing an app in a VM, this would be the appropriate option.
2. **Reserved Virtual Machine Instances** -The Reserved Virtual Machine Instances (RI) option is an advance purchase of a virtual machine for one or three years in a specified region. The commitment is made up front, and in return, you get up to 72% price savings compared to pay-as-you-go pricing. RIs are flexible and can easily be exchanged or returned for an early termination fee. Prefer this option if the VM has to run continuously, or you need budget predictability, and you can commit to using the VM for at least a year.

Storage for the VM

All Azure virtual machines will have at least two virtual hard disks (VHDs). The first disk stores the operating system, and the second is used as temporary storage. You can add additional disks to store application data; the maximum number is determined by the VM size selection (typically two per CPU). It's common to create one or more data disks, particularly since the OS disk tends to be quite small. Also, separating out the data to different VHDs allows you to manage the security, reliability, and performance of the disk independently.

The data for each VHD is held in **Azure Storage** as page blobs, which allows Azure to allocate space only for the storage you use. It's also how your storage cost is measured; you pay for the storage you are consuming.

What is Azure Storage?

Azure Storage is Microsoft's cloud-based data storage solution. It supports almost any type of data and provides security, redundancy, and scalable access to the stored data. A storage account provides access to objects in Azure Storage for a specific subscription. VMs always have one or more storage accounts to hold each attached virtual disk.

Virtual disks can be backed by either **Standard** or **Premium Storage** accounts. Azure Premium Storage leverages solid-state drives (SSDs) to enable high performance and low latency for VMs running I/O-intensive workloads. Use Azure Premium Storage for production workloads, especially those that are sensitive to performance variations or are I/O intensive. For development or testing, Standard storage is fine.

When you create disks, you will have two options for managing the relationship between the storage account and each VHD. You can choose either **unmanaged disks** or **managed** disks.

Unmanaged disks - With unmanaged disks, you are responsible for the storage accounts that are used to hold the VHDs that correspond to your VM disks. You pay the storage account rates for the amount of space you use. A single storage account has a fixed-rate limit of 20,000 I/O operations/sec. This means that a storage account is capable of supporting 40 standard virtual hard disks at full utilization. If you need to scale out with more disks, then you'll need more storage accounts, which can get complicated.

Managed disks - Managed disks are the newer and recommended disk storage model. They elegantly solve this complexity by putting the burden of managing the storage accounts onto Azure. You specify the size of the disk, up to 4 TB, and Azure creates and manages both the disk and the storage. You don't have to worry about storage account limits, which makes managed disks easier to scale out.

Select an operating system

Azure provides a variety of OS images that you can install into the VM, including several versions of Windows and flavors of Linux. As mentioned earlier, the choice of OS will influence your hourly compute pricing as Azure bundles the cost of the OS license into the price.

If you are looking for more than just base OS images, you can search the Azure Marketplace for more sophisticated install images that include the OS and popular software tools installed for specific scenarios. For example, if you needed a new WordPress site, the standard technology stack would consist of a Linux server, Apache web server, a MySQL database, and PHP. Instead of setting up and configuring each component, you can leverage a Marketplace image and install the entire stack all at once.

Finally, if you can't find a suitable OS image, you can create your disk image with what you need, upload it to Azure storage, and use it to create an Azure VM. Keep in mind that Azure only supports 64-bit operating systems.

Azure Virtual Machine creation and management options

The Azure portal is the easiest way to create resources such as VMs when you are getting started. However, it's not necessarily the most efficient or quickest way to work with Azure, particularly if you need to create several resources together.

Overview of VM creation and management options

Azure portal

The Azure portal provides an easy-to-use browser-based user interface that allows you to create and manage all your Azure resources. For example, you can set up a new database, increase the compute power of your virtual machines, and monitor your monthly costs. It's also a great learning tool, since you can survey all available resources and use guided wizards to create the ones you need.

Once logged in, you're presented with two main areas. The first is a menu with the options to help you create resources, monitor resources, and manage billing. The second is a customizable dashboard that provides you with a snapshot view of all the essential services you've deployed to Azure. You'll most likely find the portal the most comfortable option to use when you start using Azure.

Azure Resource Manager

Let's assume you want to create a copy of a VM with the same settings. You could create a VM image, upload it to Azure, and reference it as the basis for your new VM. This process is inefficient and time-consuming. Azure provides you with the option to create a template from which to create an exact copy of a VM.

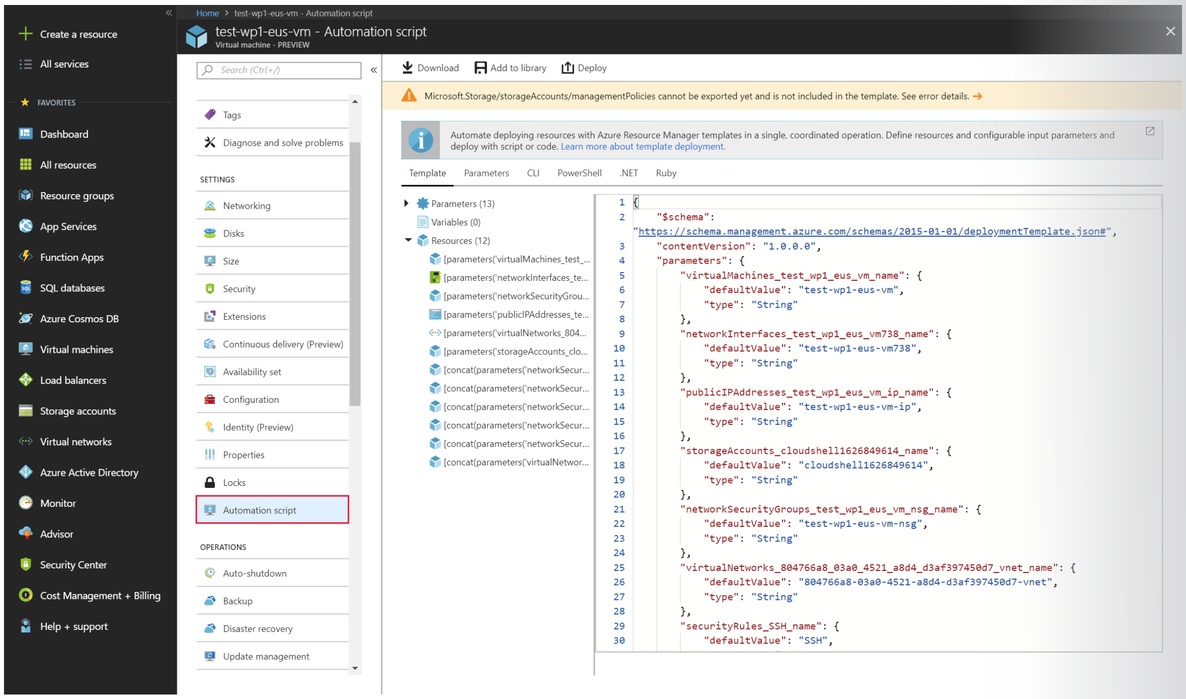
Typically, your Azure infrastructure will contain many resources, many of them related to one another in some way. For example, the VM we created has the virtual machine itself, storage, network interface, web server, and a database - all created together to run the WordPress site. **Azure Resource Manager** makes working with these related resources more efficient. It organizes resources into named resource groups that let you deploy, update, or delete all of the resources together. When we created the WordPress site, we identified the resource group as part of the VM creation, and Resource Manager placed the associated resources into the same group.

Resource Manager also allows you to create templates, which can be used to create and deploy specific configurations.

What are Resource Manager templates?

Resource Manager templates are JSON files that define the resources you need to deploy for your solution.

You can create resource templates from the Settings section for a specific VM by selecting the Automation script option.


Create a resource
All services
FAVORITES
Dashboard
All resources
Resource groups
App Services
Function Apps
SQL databases
Azure Cosmos DB
Virtual machines
Load balancers
Storage accounts
Virtual networks
Azure Active Directory
Monitor
Advisor
Security Center
Cost Management + Billing
Help + support
Home > test-wp1-eus-vm - Automation script
test-wp1-eus-vm - Automation script
Virtual machine - PREVIEW
Search (Ctrl+/)
Tags
Diagnose and solve problems
SETTINGS
Networking
Disks
Size
Security
Extensions
Continuous delivery (Preview)
Availability set
Configuration
Identity (Preview)
Properties
Locks
Automation script
OPERATIONS
Auto-shutdown
Backup
Disaster recovery
Update management
Download
Add to library
Deploy
Template
Parameters
CLI
PowerShell
.NET
Ruby
Parameters (13)
Variables (0)
Resources (12)
[parameters('virtualMachines_test_...
[parameters('networkInterfaces_te...
[parameters('networkSecurityGrou...
[parameters('publicIPAddresses_te...
[parameters('virtualNetworks_804...
[parameters('storageAccounts_clo...
[concat(parameters('networkSecur...
[concat(parameters('networkSecur...
[concat(parameters('networkSecur...
[concat(parameters('networkSecur...
[concat(parameters('networkSecur...
[concat(parameters('virtualNetwor...

Microsoft.Storage/storageAccounts/managementPolicies cannot be exported yet and is not included in the template. See error details. →

Automate deploying resources with Azure Resource Manager templates in a single, coordinated operation. Define resources and configurable input parameters and deploy with script or code. Learn more about template deployment.

```
{
    "$schema":
"https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "virtualMachines_test_wp1_eus_vm_name": {
            "defaultValue": "test-wp1-eus-vm",
            "type": "String"
        },
        "networkInterfaces_test_wp1_eus_vm738_name": {
            "defaultValue": "test-wp1-eus-vm738",
            "type": "String"
        },
        "publicIPAddresses_test_wp1_eus_vm_ip_name": {
            "defaultValue": "test-wp1-eus-vm-ip",
            "type": "String"
        },
        "storageAccounts_cloudshell1626849614_name": {
            "defaultValue": "cloudshell1626849614",
            "type": "String"
        },
        "networkSecurityGroups_test_wp1_eus_vm_nsg_name": {
            "defaultValue": "test-wp1-eus-vm-nsg",
            "type": "String"
        },
        "virtualNetworks_804766a8_03a0_4521_a8d4_d3af397450d7_vnet_name": {
            "defaultValue": "804766a8-03a0-4521-a8d4-d3af397450d7-vnet",
            "type": "String"
        },
        "securityRules_SSH_name": {
            "defaultValue": "SSH",
```

You have the option to save the resource template for later use or immediately deploy a new VM based on this template. For example, you might create a VM from a template in a test environment and find it doesn't quite work to replace your on-premises machine. You can delete the resource group, which deletes all of the resources, tweak the template, and try again. If you only want to make changes to the existing deployed resources, you can change the template used to create it and deploy it again. Resource Manager will change the resources to match the new template.

Once you have it working the way you want it, you can take that template and easily re-create multiple versions of your infrastructure, such as staging and production. You can parameterize fields such as the VM name, network name, storage account name, etc., and load the template repeatedly, using different parameters to customize each environment.

You can use automation scripting tools such as the Azure CLI, Azure PowerShell, or even the Azure REST APIs with your favorite programming language to process resource templates, making this a powerful tool for quickly spinning up your infrastructure.

Azure PowerShell

Creating administration scripts is a powerful way to optimize your workflow. You can automate everyday, repetitive tasks, and once a script has been verified, it will run consistently, likely reducing errors. Azure PowerShell is ideal for one-off interactive tasks and/or the automation of repeated tasks.

PowerShell is a cross-platform shell that provides services like the shell window and command parsing. Azure PowerShell is an optional add-on package that adds the Azure-specific commands (referred to as cmdlets). You can learn more about installing and using Azure PowerShell in a separate training module.

For example, you can use the `New-AzureRmVM` cmdlet to create a new Azure virtual machine.

```
New-AzureRmVm `
    -ResourceGroupName "TestResourceGroup" `
    -Name "test-wp1-eus-vm" `
    -Location "East US" `
    -VirtualNetworkName "test-wp1-eus-network" `
    -SubnetName "default" `
    -SecurityGroupName "test-wp1-eus-nsg" `
    -PublicIpAddressName "test-wp1-eus-pubip" `
    -OpenPorts 80,3389
```

As shown here, you supply various parameters to handle the large number of VM configuration settings available. Most of the parameters have reasonable values; you only need to specify the required parameters.

Azure CLI

Another option for scripting and command-line Azure interaction is the Azure CLI.

The Azure CLI is Microsoft's cross-platform command-line tool for managing Azure resources such as virtual machines and disks from the command line. It's available for macOS, Linux, and Windows, or in the browser using the Cloud Shell. Like Azure PowerShell, the Azure CLI is a powerful way to streamline your administrative workflow. Unlike Azure PowerShell, the Azure CLI does not need PowerShell to function.

For example, you can create an Azure VM with the `az vm create` command.

```
az vm create \
    --resource-group TestResourceGroup \
    --name test-wp1-eus-vm \
    --image win2016datacenter \
    --admin-username jonc \
    --admin-password aReallyGoodPasswordHere
```

The Azure CLI can be used with other scripting languages, for example, Ruby and Python. Both languages are commonly used on non-Windows-based machines where the developer might not be familiar with PowerShell.

Programmatic (APIs)

Generally speaking, both Azure PowerShell and Azure CLI are good options if you have simple scripts to run and want to stick to command-line tools. When it comes to more complex scenarios, where the creation and management of VMs form part of a larger application with complex logic, another approach is needed.

You can interact with every type of resource in Azure programmatically.

Azure REST API

The Azure REST API provides developers with operations categorized by resource as well as the ability to create and manage VMs. Operations are exposed as URIs with corresponding HTTP methods (`GET`, `PUT`, `POST`, `DELETE`, and `PATCH`) and a corresponding response.

The Azure Compute APIs give you programmatic access to virtual machines and their supporting resources. With this API, you have operations to:

- Create and manage availability sets
- Add and manage virtual machine extensions

- Create and manage managed disks, snapshots, and images
- Access the platform images available in Azure
- Retrieve usage information of your resources
- Create and manage virtual machines
- Create and manage virtual machine scale sets

Azure Client SDK

Even though the REST API is platform and language agnostic, most often developers will look toward a higher level of abstraction. The Azure Client SDK encapsulates the Azure REST API, making it much easier for developers to interact with Azure.

The Azure Client SDKs are available for a variety of languages and frameworks, including .NET-based languages such as C#, Java, Node.js, PHP, Python, Ruby, and Go.

Here's an example snippet of C# code to create an Azure VM using the `Microsoft.Azure.Management.Fluent` NuGet package:

```
var azure = Azure
    .Configure()
    .WithLogLevel(HttpLoggingDelegatingHandler.Level.Basic)
    .Authenticate(credentials)
    .WithDefaultSubscription();
// ...
var vmName = "test-wp1-eus-vm";

azure.VirtualMachines.Define(vmName)
    .WithRegion(Region.USEast)
    .WithExistingResourceGroup("TestResourceGroup")
    .WithExistingPrimaryNetworkInterface(networkInterface)
    .WithLatestWindowsImage("MicrosoftWindowsServer", "WindowsServer",
"2012-R2-Datacenter")
    .WithAdminUsername("jonc")
    .WithAdminPassword("aReallyGoodPasswordHere")
    .WithComputerName(vmName)
    .WithSize(VirtualMachineSizeTypes.StandardDS1)
    .Create();
```

Here's the same snippet in Java using the Azure Java SDK:

```
String vmName = "test-wp1-eus-vm";
// ...
VirtualMachine virtualMachine = azure.virtualMachines()
    .define(vmName)
    .withRegion(Region.US_EAST)
    .withExistingResourceGroup("TestResourceGroup")
    .withExistingPrimaryNetworkInterface(networkInterface)
    .withLatestWindowsImage("MicrosoftWindowsServer", "WindowsServer",
"2012-R2-Datacenter")
    .withAdminUsername("jonc")
    .withAdminPassword("aReallyGoodPasswordHere")
```

```
        .withComputerName(vmName)
        .withSize("Standard_DS1")
        .create();
```

Azure VM Extensions

Let's assume you want to configure and install additional software on your virtual machine after the initial deployment. You want this task to use a specific configuration, monitored and executed automatically.

Azure VM extensions are small applications that allow you to configure and automate tasks on Azure VMs after initial deployment. **Azure VM extensions** can be run with the Azure CLI, PowerShell, Azure Resource Manager templates, and the Azure portal.

You bundle extensions with a new VM deployment or run them against an existing system.

Azure Automation Services

Saving time, reducing errors, and increasing efficiency are some of the most significant operational management challenges faced when managing remote infrastructure. If you have a lot of infrastructure services, you might want to consider using higher-level services in Azure to help you operate from a higher level.

Azure Automation allows you to integrate services that allow you to automate frequent, time-consuming, and error-prone management tasks with ease. These services include process automation, configuration management, and update management.

- **Process Management.** Let's assume you have a VM that is monitored for a specific error event. You want to take action and fix the problem as soon as it's reported. Process automation allows you to set up watcher tasks that can respond to events that may occur in your datacenter.
- **Configuration Management.** Perhaps you want to track software updates that become available for the operating system that runs on your VM. There are specific updates you may want to include or exclude. Configuration management allows you to track these updates and take action as required. You use System Center Configuration Manager to manage your company's PC, servers, and mobile devices. You can extend this support to your Azure VMs with Configuration Manager.
- **Update Management.** This is used to manage updates and patches for your VMs. With this service, you're able to assess the status of available updates, schedule installation, and review deployment results to verify updates applied successfully. Update management incorporates services that provide process and configuration management. You enable update management for a VM directly from your Azure Automation account. You can also allow update management for a single virtual machine from the virtual machine blade in the portal.

As you can see, Azure provides a variety of tools to create and administer resources so that you can integrate management operations into a process that works for you.

Manage the availability of your Azure VMs

Often, the success of a services company is directly related to the service level agreements (SLA) the company has with its customers. Your customers expect the services you provide always to be available and their data kept safe. This is something that Microsoft takes very seriously. Azure provides tools you can use to manage availability, data security, and monitoring, so you know your services are always available for your customers.

Administration of an Azure VM isn't limited to managing the operating system or software that runs on the VM. It helps to know which services Azure provides that ensure service availability and support automation. These services help you to plan your organization's business continuity and disaster recovery strategy.

Here, we'll cover an Azure service that helps you improve VM availability, streamlines VM management tasks, and keeps your VM data backed up and safe. Let's start by defining availability.

What is availability?

Availability is the percentage of time a service is available for use.

Let's assume you have a website and you want your customers to be able to access information at all times. Your expectation is 100% availability concerning website access.

Why do I need to think about availability when using Azure?

Azure VMs run on physical servers hosted within the Azure Datacenter. As with most physical devices, there's a chance that there could be a failure. If the physical server fails, the virtual machines hosted on that server will also fail. If this happens, Azure will move the VM to a healthy host server automatically. However, this self-healing migration could take several minutes, during which, the application(s) hosted on that VM will not be available.

The VMs could also be affected by periodic updates initiated by Azure itself. These maintenance events range from software updates to hardware upgrades and are required to improve platform reliability and performance. These events usually are performed without impacting any guest VMs, but sometimes the virtual machines will be rebooted to complete an update or upgrade.

Note: Microsoft does not automatically update your VM's OS or software. You have complete control and responsibility for that. However, the underlying software host and hardware are periodically patched to ensure reliability and high performance at all times.

To ensure your services aren't interrupted and avoid a single point of failure, it's recommended to deploy at least two instances of each VM. This feature is called an availability set.

What is an availability set?

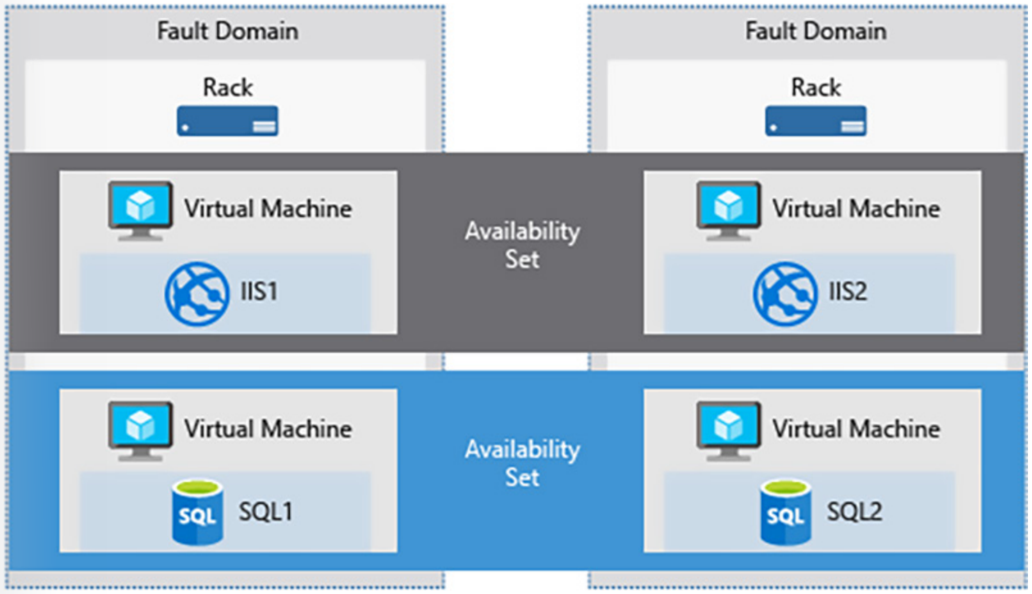
An availability set is a logical feature used to ensure that a group of related VMs are deployed so that they aren't all subject to a single point of failure and not all upgraded at the same time during a host operating system upgrade in the datacenter. VMs placed in an availability set should perform an identical set of functionalities and have the same software installed.

Tip: Microsoft offers a 99.95% external connectivity service level agreement (SLA) for multiple-instance VMs deployed in an availability set. That means that for the SLA to apply, there must be at least two instances of the VM deployed within an availability set.

You can create availability sets through the Azure portal in the disaster recovery section. Also, you can build them using Resource Manager templates, or any of the scripting or API tools. When you place VMs into an availability set, Azure guarantees to spread them across Fault Domains and Update Domains.

What is a fault domain?

A fault domain is a logical group of hardware in Azure that shares a common power source and network switch. You can think of it as a rack within an on-premises datacenter. The first two VMs in an availability set will be provisioned into two different racks so that if the network or the power failed in a rack, only one VM would be affected. Fault domains are also defined for managed disks attached to VMs.



What is an update domain?

An update domain is a logical group of hardware that can undergo maintenance or be rebooted at the same time. Azure will automatically place availability sets into update domains to minimize the impact when the Azure platform introduces host operating system changes. Azure then processes each update domain one at a time.

Availability sets are a powerful feature to ensure the services running in your VMs are always available to your customers. However, they aren't foolproof. What if something happens to the data or the software running on the VM itself? For that, we'll need to look at other disaster recovery and backup techniques.

Failover across locations

You can also replicate your infrastructure across sites to handle regional failover. Azure Site Recovery replicates workloads from a primary site to a secondary location. If an outage happens at your primary site, you can fail over to a secondary location. This failover allows users to continue to access your applications without interruption. You can then fail back to the primary location once it's up and running again. Azure Site Recovery is about replication of virtual or physical machines; it keeps your workloads available in an outage.

While there are many attractive technical features to Site Recovery, there are at least two significant business advantages:

1. Site Recovery enables the use of Azure as a destination for recovery, thus eliminating the cost and complexity of maintaining a secondary physical datacenter.
2. Site Recovery makes it incredibly simple to test failovers for recovery drills without impacting production environments. This makes it easy to test your planned or unplanned failovers. After all, you don't have a good disaster recovery plan if you've never tried to failover.

The recovery plans you create with Site Recovery can be as simple or as complex as your scenario requires. They can include custom PowerShell scripts, Azure Automation runbooks, or manual intervention steps. You can leverage the recovery plans to replicate workloads to Azure, easily enabling new opportunities for migration, temporary bursts during surge periods, or development and testing of new applications.

Azure Site Recovery works with Azure resources, or Hyper-V, VMware, and physical servers in your on-premises infrastructure and can be a key part of your organization's business continuity and disaster recovery (BCDR) strategy by orchestrating the replication, failover, and recovery of workloads and applications if the primary location fails.

Create an Azure VM by using the Azure portal

Azure virtual machines (VMs) can be created through the Azure portal. This method provides a browser-based user interface to create VMs and their associated resources. This tutorial shows you how to use the Azure portal to deploy a virtual machine (VM) in Azure that runs Windows Server 2016. To see your VM in action, you then RDP to the VM and install the IIS web server.

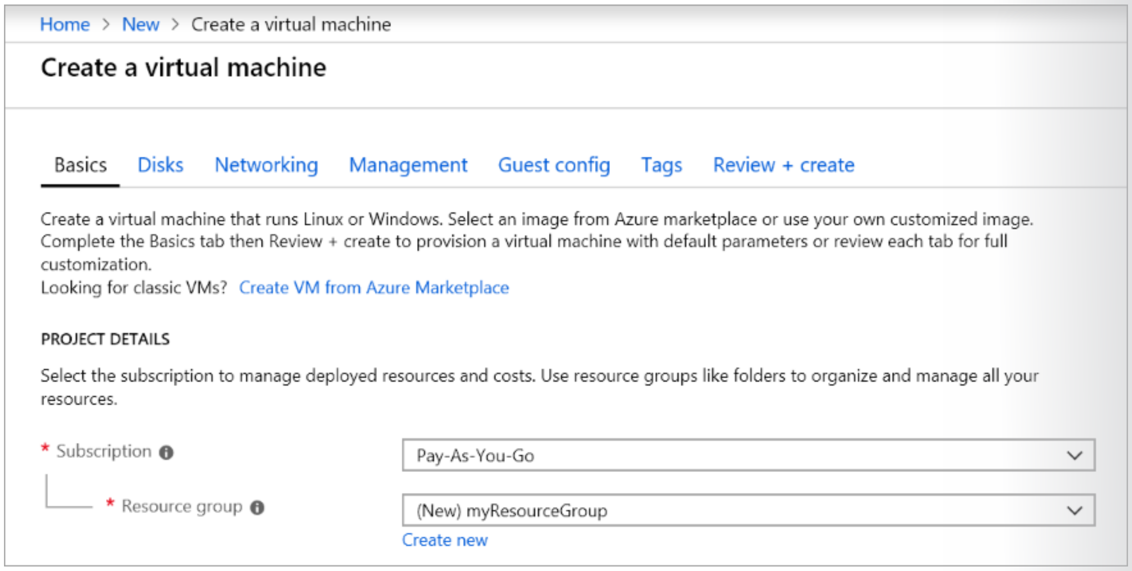
If you don't have an Azure subscription, create a **free account**[1] before you begin.

Sign in to Azure

Sign in to the Azure portal at https://portal.azure.com.

Create virtual machine

1. Choose **Create a resource** in the upper left-hand corner of the Azure portal.
2. In the search box above the list of Azure Marketplace resources, search for and select **Windows Server 2016 Datacenter**, then choose **Create**.
3. In the **Basics** tab, under **Project details**, make sure the correct subscription is selected and then choose to **Create new** resource group. Type the *myResourceGroup* for the name.



Create a virtual machine

Create a virtual machine that runs Linux or Windows. Select an image from Azure marketplace or use your own customized image. Complete the Basics tab then Review + create to provision a virtual machine with default parameters or review each tab for full customization.
Looking for classic VMs? Create VM from Azure Marketplace

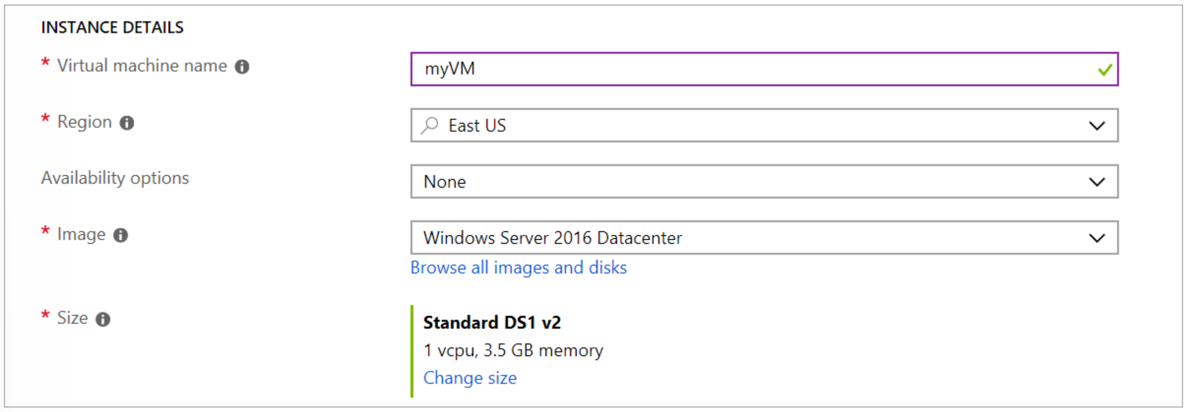
PROJECT DETAILS

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

* Subscription: Pay-As-You-Go
* Resource group: (New) myResourceGroup

4.
5. Under **Instance details**, type *myVM* for the **Virtual machine name** and choose *East US* for your **Location**. Leave the other defaults.

1 https://azure.microsoft.com/free/?WT.mc_id=A261C142F

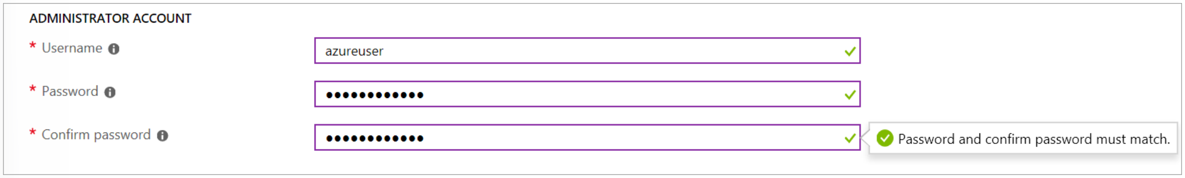


INSTANCE DETAILS

* Virtual machine name	myVM
* Region	East US
Availability options	None
* Image	Windows Server 2016 Datacenter Browse all images and disks
* Size	**Standard DS1 v2** 1 vcpu, 3.5 GB memory Change size

6.

7. Under **Administrator account**, provide a username, such as *azureuser* and a password. The password must be at least 12 characters long and meet the defined complexity requirements.



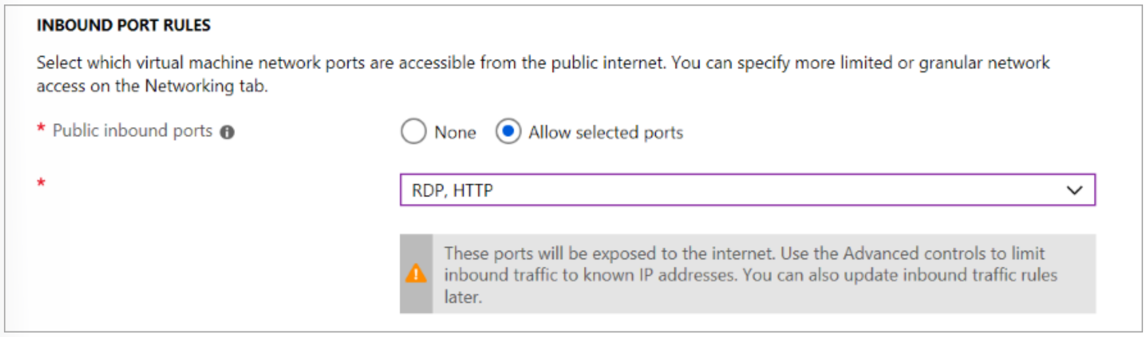
ADMINISTRATOR ACCOUNT

* Username	azureuser
* Password	••••••••••••
* Confirm password	••••••••••••

Password and confirm password must match.

8.

9. Under **Inbound port rules**, choose **Allow selected ports** and then select **RDP (3389)** and **HTTP** from the drop-down.



INBOUND PORT RULES

Select which virtual machine network ports are accessible from the public internet. You can specify more limited or granular network access on the Networking tab.

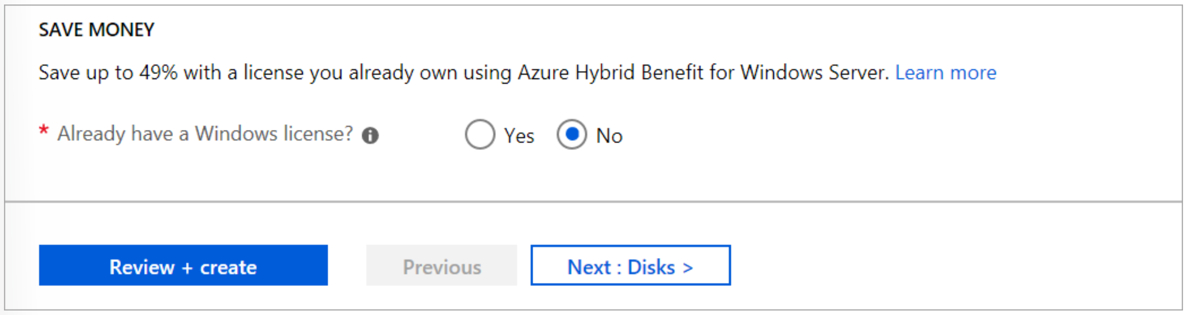
* Public inbound ports: None [✓ Allow selected ports]

* RDP, HTTP

These ports will be exposed to the internet. Use the Advanced controls to limit inbound traffic to known IP addresses. You can also update inbound traffic rules later.

10.

11. Leave the remaining defaults and then select the **Review + create** button at the bottom of the page.



SAVE MONEY

Save up to 49% with a license you already own using Azure Hybrid Benefit for Windows Server. Learn more

* Already have a Windows license? Yes [✓ No]

12.

Connect to virtual machine

Create a remote desktop connection to the virtual machine. These directions tell you how to connect to your VM from a Windows computer. On a Mac, you need to install an RDP client from the Mac App Store.

1. Select the **Connect** button on the virtual machine properties page.
2. In the **Connect to virtual machine** page, keep the default options to connect by DNS name over port 3389 and click **Download RDP file**.
3. Open the downloaded RDP file and select **Connect** when prompted.
4. In the **Windows Security** window, select **More choices** and then **Use a different account**. Type the username as localhost\username, enter password you created for the virtual machine, and then select **OK**.
5. You may receive a certificate warning during the sign-in process. Select **Yes** or **Continue** to create the connection.

Install web server

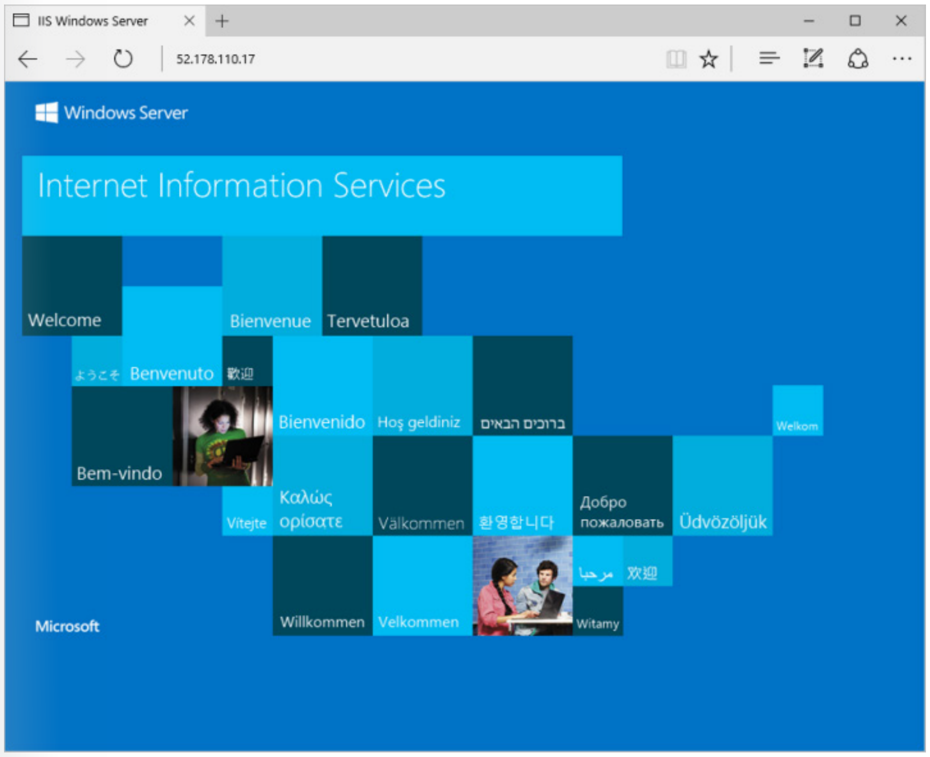
To see your VM in action, install the IIS web server. Open a PowerShell prompt on the VM and run the following command:

```
Install-WindowsFeature -name Web-Server -IncludeManagementTools
```

When done, close the RDP connection to the VM.

View the IIS welcome page

In the portal, select the VM and in the overview of the VM, use the **Click to copy** button to the right of the IP address to copy it and paste it into a browser tab. The default IIS welcome page will open, and should look like this:



Clean up resources

When no longer needed, you can delete the resource group, virtual machine, and all related resources. To do so, select the resource group for the virtual machine, select **Delete**, then confirm the name of the resource group to delete.

Create an Azure VM by using Powershell

The Azure PowerShell module is used to create and manage Azure resources from the PowerShell command line or in scripts. This tutorial shows you how to use the Azure PowerShell module to deploy a virtual machine (VM) in Azure that runs Windows Server 2016. To see your VM in action, you then RDP to the VM and install the IIS web server.

If you don't have an Azure subscription, create a **free account**[2] before you begin.

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this tutorial. It has common Azure tools preinstalled and configured to use with your account.

You can also launch Cloud Shell in a separate browser tab by going to https://shell.azure.com/powershell.

If you choose to install and use the PowerShell locally, this tutorial requires the Azure PowerShell module version 5.7.0 or later. Run `Get-Module -ListAvailable AzureRM` to find the version, you also need to run `Connect-AzureRmAccount` to create a connection with Azure.

2 https://azure.microsoft.com/free/?WT.mc_id=A261C142F

Create resource group

Create an Azure resource group with `New-AzureRmResourceGroup`. A resource group is a logical container into which Azure resources are deployed and managed.

```
New-AzureRmResourceGroup -Name myResourceGroup -Location EastUS
```

Create virtual machine

Create a VM with `New-AzureRmVM`. Provide names for each of the resources and the `New-AzureRmVM` cmdlet creates if they don't already exist.

When prompted, provide a username and password to be used as the logon credentials for the VM:

```
New-AzureRmVm `
    -ResourceGroupName "myResourceGroup" `
    -Name "myVM" `
    -Location "East US" `
    -VirtualNetworkName "myVnet" `
    -SubnetName "mySubnet" `
    -SecurityGroupName "myNetworkSecurityGroup" `
    -PublicIpAddressName "myPublicIpAddress" `
    -OpenPorts 80,3389
```

Connect to virtual machine

After the deployment has completed, RDP to the VM. To see your VM in action, the IIS web server is then installed.

To see the public IP address of the VM, use the `Get-AzureRmPublicIpAddress` cmdlet:

```
Get-AzureRmPublicIpAddress -ResourceGroupName "myResourceGroup" | Select
"IpAddress"
```

Use the following command to create a remote desktop session from your local computer. Replace the IP address with the public IP address of your VM.

```
mstsc /v:publicIpAddress
```

In the **Windows Security** window, select **More choices**, and then select **Use a different account**. Type the username as localhost\username, enter password you created for the virtual machine, and then select **OK**.

You may receive a certificate warning during the sign-in process. Select **Yes** or **Continue** to create the connection

Install web server

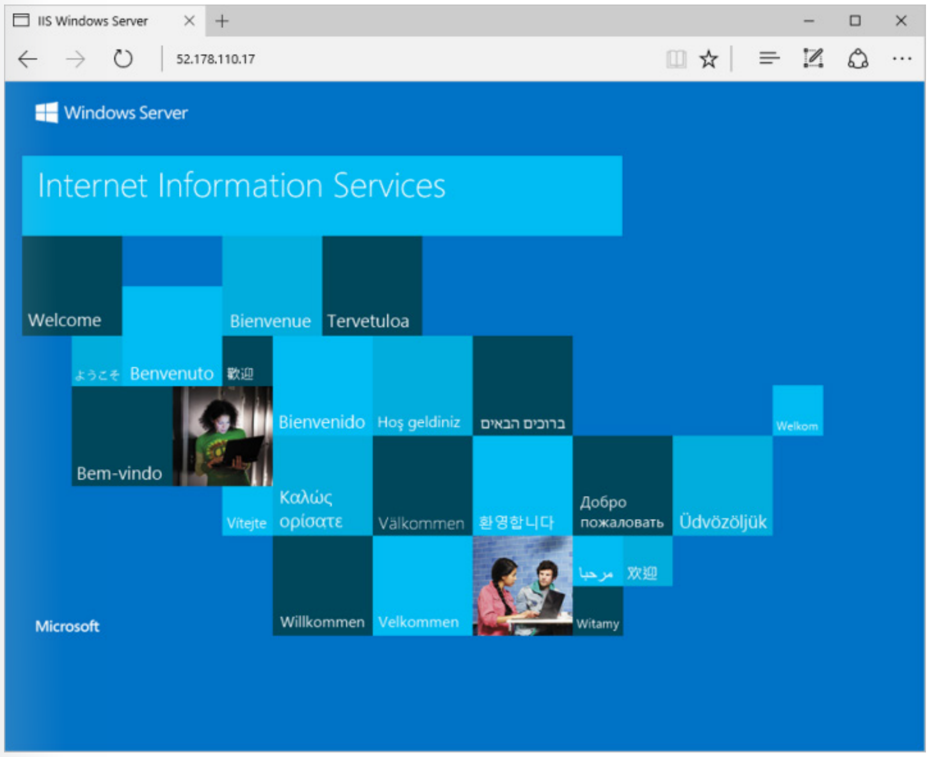
To see your VM in action, install the IIS web server. Open a PowerShell prompt on the VM and run the following command:

```
Install-WindowsFeature -name Web-Server -IncludeManagementTools
```

When done, close the RDP connection to the VM.

View the web server in action

With IIS installed and port 80 now open on your VM from the Internet, use a web browser of your choice to view the default IIS welcome page. Use the public IP address of your VM obtained in a previous step. The following example shows the default IIS web site:



Clean up resources

When no longer needed, you can use the `Remove-AzureRmResourceGroup` cmdlet to remove the resource group, VM, and all related resources:

```
Remove-AzureRmResourceGroup -Name myResourceGroup
```

Create and Manage Azure VMs by using C#

An Azure Virtual Machine (VM) needs several supporting Azure resources. This section covers creating, managing, and deleting VM resources using C# and Visual Studio. You learn how to:

- Create a Visual Studio project
- Install the package
- Create credentials
- Create resources
- Perform management tasks
- Delete resources

- Run the application

Create a Visual Studio project

1. If you haven't already, install **Visual Studio**[3]. Select **.NET desktop development** on the Workloads page, and then select **Install**. In the summary, you can see that **.NET Framework 4 - 4.6 development tools** is automatically selected for you. If you have already installed Visual Studio, you can add the .NET workload using the Visual Studio Launcher.
2. In Visual Studio, select **File > New > Project**.
3. In **Templates > Visual C#**, select **Console App (.NET Framework)**, enter *myDotnetProject* for the name of the project, select the location of the project, and then select **OK**.

Install the package

NuGet packages are the easiest way to install the libraries that you need to finish these steps. To get the libraries that you need in Visual Studio, follow these steps:

1. select **Tools > Nuget Package Manager**, and then select **Package Manager Console**.
2. Type this command in the console:

```
Install-Package Microsoft.Azure.Management.Fluent
```

Create credentials

Before you start this step, make sure that you have access to an Active Directory service principal, if you need to create one this **article**[4] will walk you through the steps. You should also record the application ID, the authentication key, and the tenant ID that you need in a later step.

Create the authorization file

1. In Solution Explorer, right-select **myDotnetProject > Add > New Item**, and then select **Text File** in *Visual C# Items*. Name the file *azureauth.properties*, and then select **Add**.
2. Add these authorization properties:

```
subscription=<subscription-id>
client=<application-id>
key=<authentication-key>
tenant=<tenant-id>
managementURI=https://management.core.windows.net/
baseURL=https://management.azure.com/
authURL=https://login.windows.net/
graphURL=https://graph.windows.net/
```

3. Replace `<subscription-id>` with your subscription identifier, `<application-id>` with the Active Directory application identifier, `<authentication-key>` with the application key, and `<tenant-id>` with the tenant identifier.

3 https://docs.microsoft.com/visualstudio/install/install-visual-studio
4 https://docs.microsoft.com/en-us/azure/active-directory/develop/howto-create-service-principal-portal

4. Save the azureauth.properties file.
5. Set an environment variable in Windows named AZURE_AUTH_LOCATION with the full path to authorization file that you created. For example, the following PowerShell command can be used:

```
[Environment]::SetEnvironmentVariable("AZURE_AUTH_LOCATION", "C:\Visual
Studio 2017\Projects\myDotnetProject\myDotnetProject\azureauth.properties",
"User")
```

Create the management client

1. Open the Program.cs file for the project that you created, and then add these using statements to the existing statements at top of the file:

```
using Microsoft.Azure.Management.Compute.Fluent;
using Microsoft.Azure.Management.Compute.Fluent.Models;
using Microsoft.Azure.Management.Fluent;
using Microsoft.Azure.Management.ResourceManager.Fluent;
using Microsoft.Azure.Management.ResourceManager.Fluent.Core;
```

2. To create the management client, add this code to the Main method:

```
var credentials = SdkContext.AzureCredentialsFactory
    .FromFile(Environment.GetEnvironmentVariable("AZURE_AUTH_LOCATION"));

var azure = Azure
    .Configure()
    .WithLogLevel(HttpLoggingDelegatingHandler.Level.Basic)
    .Authenticate(credentials)
    .WithDefaultSubscription();
```

Create resources

All resources the VM relies on must be contained in a Resource group.

Create the resource group

To specify values for the application and create the resource group, add this code to the Main method:

```
var groupName = "myResourceGroup";
var vmName = "myVM";
var location = Region.USWest;

Console.WriteLine("Creating resource group...");
var resourceGroup = azure.ResourceGroups.Define(groupName)
    .WithRegion(location)
    .Create();
```

Create the availability set

Availability sets make it easier for you to maintain the virtual machines used by your application. To create the availability set, add this code to the Main method:

```
Console.WriteLine("Creating availability set...");
var availabilitySet = azure.AvailabilitySets.Define("myAVSet")
    .WithRegion(location)
    .WithExistingResourceGroup(groupName)
    .WithSku(AvailabilitySetSkuTypes.Managed)
    .Create();
```

Create the public IP address

A Public IP address is needed to communicate with the virtual machine. Add this code to the Main method to create the public IP address for the virtual machine:

```
Console.WriteLine("Creating public IP address...");
var publicIPAddress = azure.PublicIPAddresses.Define("myPublicIP")
    .WithRegion(location)
    .WithExistingResourceGroup(groupName)
    .WithDynamicIP()
    .Create();
```

Create the virtual network

A virtual machine must be in a subnet of a Virtual network. To create a subnet and a virtual network, add this code to the Main method:

```
Console.WriteLine("Creating virtual network...");
var network = azure.Networks.Define("myVNet")
    .WithRegion(location)
    .WithExistingResourceGroup(groupName)
    .WithAddressSpace("10.0.0.0/16")
    .WithSubnet("mySubnet", "10.0.0.0/24")
    .Create();
```

Create the network interface

A virtual machine needs a network interface to communicate on the virtual network. To create a network interface, add this code to the Main method:

```
Console.WriteLine("Creating network interface...");
var networkInterface = azure.NetworkInterfaces.Define("myNIC")
    .WithRegion(location)
    .WithExistingResourceGroup(groupName)
    .WithExistingPrimaryNetwork(network)
    .WithSubnet("mySubnet")
    .WithPrimaryPrivateIPAddressDynamic()
    .WithExistingPrimaryPublicIPAddress(publicIPAddress)
```

```
        .Create();
```

Create the virtual machine

Now that you created all the supporting resources, you can create a virtual machine. To create the virtual machine, add this code to the Main method:

```
Console.WriteLine("Creating virtual machine...");
azure.VirtualMachines.Define(vmName)
    .WithRegion(location)
    .WithExistingResourceGroup(groupName)
    .WithExistingPrimaryNetworkInterface(networkInterface)
    .WithLatestWindowsImage("MicrosoftWindowsServer", "WindowsServer", 
"2012-R2-Datacenter")
    .WithAdminUsername("azureuser")
    .WithAdminPassword("Azure12345678")
    .WithComputerName(vmName)
    .WithExistingAvailabilitySet(availabilitySet)
    .WithSize(VirtualMachineSizeTypes.StandardDS1)
    .Create();
```

If you want to use an existing disk instead of a marketplace image, use this code:

```
var managedDisk = azure.Disks.Define("myosdisk")
    .WithRegion(location)
    .WithExistingResourceGroup(groupName)
    .WithWindowsFromVhd("https://mystorage.blob.core.windows.net/vhds/
myosdisk.vhd")
    .WithSizeInGB(128)
    .WithSku(DiskSkuTypes.PremiumLRS)
    .Create();

azure.VirtualMachines.Define("myVM")
    .WithRegion(location)
    .WithExistingResourceGroup(groupName)
    .WithExistingPrimaryNetworkInterface(networkInterface)
    .WithSpecializedOSDisk(managedDisk, OperatingSystemTypes.Windows)
    .WithExistingAvailabilitySet(availabilitySet)
    .WithSize(VirtualMachineSizeTypes.StandardDS1)
    .Create();
```

Perform management tasks

During the lifecycle of a virtual machine, you may want to run management tasks such as starting, stopping, or deleting a virtual machine. Additionally, you may want to create code to automate repetitive or complex tasks.

When you need to do anything with the VM, you need to get an instance of it:

```
var vm = azure.VirtualMachines.GetByResourceGroup(groupName, vmName);
```

Stop the VM

You can stop a virtual machine and keep all its settings, but continue to be charged for it, or you can stop a virtual machine and deallocate it. When a virtual machine is deallocated, all resources associated with it are also deallocated and billing ends for it.

To stop the virtual machine without deallocating it, add this code to the Main method:

```
Console.WriteLine("Stopping vm...");
vm.PowerOff();
Console.WriteLine("Press enter to continue...");
Console.ReadLine();
```

If you want to deallocate the virtual machine, change the PowerOff call to this code:

```
vm.Deallocate();
```

Start the VM

To start the virtual machine, add this code to the Main method:

```
Console.WriteLine("Starting vm...");
vm.Start();
Console.WriteLine("Press enter to continue...");
Console.ReadLine();
```

Resize the VM

Many aspects of deployment should be considered when deciding on a size for your virtual machine. To change size of the virtual machine, add this code to the Main method:

```
Console.WriteLine("Resizing vm...");
vm.Update()
    .WithSize(VirtualMachineSizeTypes.StandardDS2)
    .Apply();
Console.WriteLine("Press enter to continue...");
Console.ReadLine();
```

Add a data disk to the VM

To add a data disk to the virtual machine, add this code to the Main method to add a data disk that is 2 GB in size, han a LUN of 0 and a caching type of ReadWrite:

```
Console.WriteLine("Adding data disk to vm...");
vm.Update()
    .WithNewDataDisk(2, 0, CachingTypes.ReadWrite)
    .Apply();
Console.WriteLine("Press enter to delete resources...");
Console.ReadLine();
```

Delete resources

Because you are charged for resources used in Azure, it is always good practice to delete resources that are no longer needed. If you want to delete the virtual machines and all the supporting resources, all you have to do is delete the resource group.

To delete the resource group, add this code to the Main method:

```
azure.ResourceGroups.DeleteByName(groupName);
```

Run the application

It should take about five minutes for this console application to run completely from start to finish.

1. To run the console application, select **Start**.
2. Before you press **Enter** to start deleting resources, you could take a few minutes to verify the creation of the resources in the Azure portal. Click the deployment status to see information about the deployment.

Create ARM templates

Azure Resource Manager overview

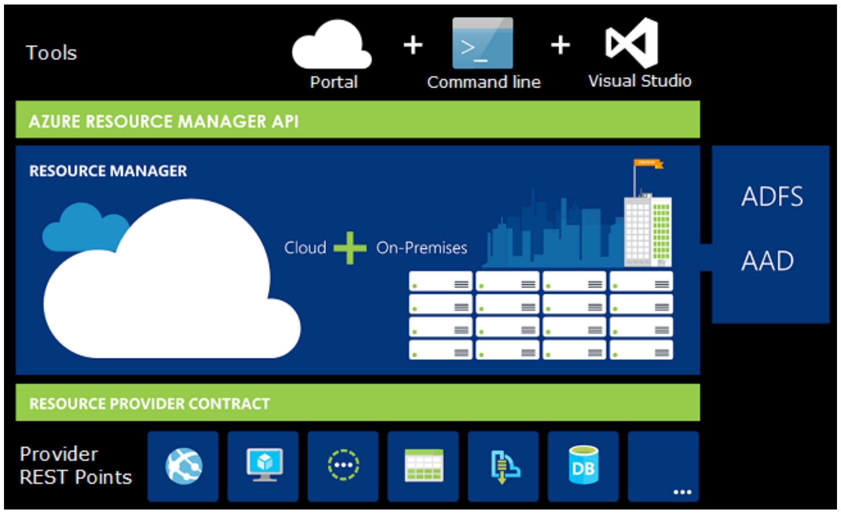
The infrastructure for your application is typically made up of many components – maybe a virtual machine, storage account, and virtual network, or a web app, database, database server, and third-party services. You may not see these components as separate entities, instead you see them as related and interdependent parts of a single entity. You want to deploy, manage, and monitor them as a group. Azure Resource Manager enables you to work with the resources in your solution as a group. You can deploy, update, or delete all the resources for your solution in a single, coordinated operation. You use a template for deployment and that template can work for different environments such as testing, staging, and production. Resource Manager provides security, auditing, and tagging features to help you manage your resources after deployment.

Consistent management layer

Resource Manager provides a consistent management layer to perform tasks through Azure PowerShell, Azure CLI, Azure portal, REST API, and client SDKs. All capabilities that are available in the Azure portal are also available through Azure PowerShell, Azure CLI, the Azure REST APIs, and client SDKs. Functionality initially released through APIs will be represented in the portal within 180 days of initial release.

Choose the tools and APIs that work best for you - they have the same capability and provide consistent results.

The following image shows how all the tools interact with the same Azure Resource Manager API. The API passes requests to the Resource Manager service, which authenticates and authorizes the requests. Resource Manager then routes the requests to the appropriate resource providers.



Terminology

If you're new to Azure Resource Manager, there are some terms you might not be familiar with.

- **resource** - A manageable item that is available through Azure. Some common resources are a virtual machine, storage account, web app, database, and virtual network, but there are many more.
- **resource group** - A container that holds related resources for an Azure solution. The resource group can include all the resources for the solution, or only those resources that you want to manage as a

group. You decide how you want to allocate resources to resource groups based on what makes the most sense for your organization.

- **resource provider** - A service that supplies the resources you can deploy and manage through Resource Manager. Each resource provider offers operations for working with the resources that are deployed. Some common resource providers are Microsoft.Compute, which supplies the virtual machine resource, Microsoft.Storage, which supplies the storage account resource, and Microsoft.Web, which supplies resources related to web apps.
- **Resource Manager template** - A JavaScript Object Notation (JSON) file that defines one or more resources to deploy to a resource group. It also defines the dependencies between the deployed resources. The template can be used to deploy the resources consistently and repeatedly.
- **declarative syntax** - Syntax that lets you state "Here is what I intend to create" without having to write the sequence of programming commands to create it. The Resource Manager template is an example of declarative syntax. In the file, you define the properties for the infrastructure to deploy to Azure.

The benefits of using Resource Manager

Resource Manager provides several benefits:

- You can deploy, manage, and monitor all the resources for your solution as a group, rather than handling these resources individually.
- You can repeatedly deploy your solution throughout the development lifecycle and have confidence your resources are deployed in a consistent state.
- You can manage your infrastructure through declarative templates rather than scripts.
- You can define the dependencies between resources so they're deployed in the correct order.
- You can apply access control to all services in your resource group because Role-Based Access Control (RBAC) is natively integrated into the management platform.
- You can apply tags to resources to logically organize all the resources in your subscription.
- You can clarify your organization's billing by viewing costs for a group of resources sharing the same tag.

Guidance

The following suggestions help you take full advantage of Resource Manager when working with your solutions.

- Define and deploy your infrastructure through the declarative syntax in Resource Manager templates, rather than through imperative commands.
- Define all deployment and configuration steps in the template. You should have no manual steps for setting up your solution.
- Run imperative commands to manage your resources, such as to start or stop an app or machine.
- Arrange resources with the same lifecycle in a resource group. Use tags for all other organizing of resources.

Resource groups

There are some important factors to consider when defining your resource group:

- All the resources in your group should share the same lifecycle. You deploy, update, and delete them together. If one resource, such as a database server, needs to exist on a different deployment cycle it should be in another resource group.
- Each resource can only exist in one resource group.
- You can add or remove a resource to a resource group at any time.
- You can move a resource from one resource group to another group.
- A resource group can contain resources that reside in different regions.
- A resource group can be used to scope access control for administrative actions.
- A resource can interact with resources in other resource groups. This interaction is common when the two resources are related but don't share the same lifecycle (for example, web apps connecting to a database).

When creating a resource group, you need to provide a location for that resource group. You may be wondering, "Why does a resource group need a location? And, if the resources can have different locations than the resource group, why does the resource group location matter at all?" The resource group stores metadata about the resources. Therefore, when you specify a location for the resource group, you're specifying where that metadata is stored. For compliance reasons, you may need to ensure that your data is stored in a particular region.

Resource providers

Each resource provider offers a set of resources and operations for working with an Azure service. For example, if you want to store keys and secrets, you work with the **Microsoft.KeyVault** resource provider. This resource provider offers a resource type called vaults for creating the key vault.

The name of a resource type is in the format: **{resource-provider}/{resource-type}**. For example, the key vault type is **Microsoft.KeyVault/vaults**.

Before getting started with deploying your resources, you should gain an understanding of the available resource providers. Knowing the names of resource providers and resources helps you define resources you want to deploy to Azure. Also, you need to know the valid locations and API versions for each resource type.

ARM template deployment

With Resource Manager, you can create a template (in JSON format) that defines the infrastructure and configuration of your Azure solution. By using a template, you can repeatedly deploy your solution throughout its lifecycle and have confidence your resources are deployed in a consistent state. When you create a solution from the portal, the solution automatically includes a deployment template. You don't have to create your template from scratch because you can start with the template for your solution and customize it to meet your specific needs. You can also retrieve a template for an existing resource group by either exporting the current state of the resource group, or viewing the template used for a particular deployment. Viewing the exported template is a helpful way to learn about the template syntax.

Note: To view the JSON syntax for resources types, see **Define resources in Azure Resource Manager templates**[5].

5 https://docs.microsoft.com/en-us/azure/templates/

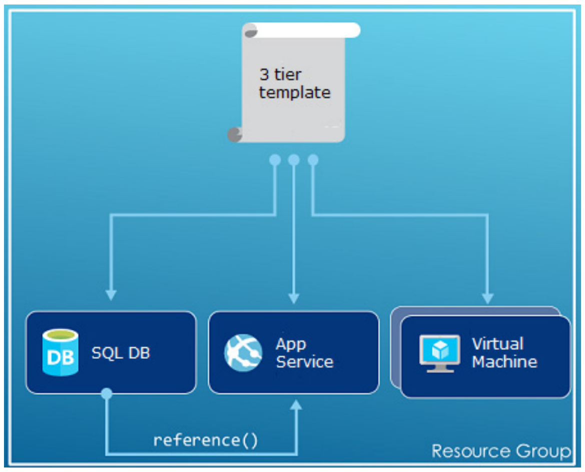
Resource Manager processes the template like any other request. It parses the template and converts its syntax into REST API operations for the appropriate resource providers. For example, when Resource Manager receives a template with the following resource definition:

```
"resources": [
  {
    "apiVersion": "2016-01-01",
    "type": "Microsoft.Storage/storageAccounts",
    "name": "mystorageaccount",
    "location": "westus",
    "sku": {
      "name": "Standard_LRS"
    },
    "kind": "Storage",
    "properties": {
    }
  }
]
```

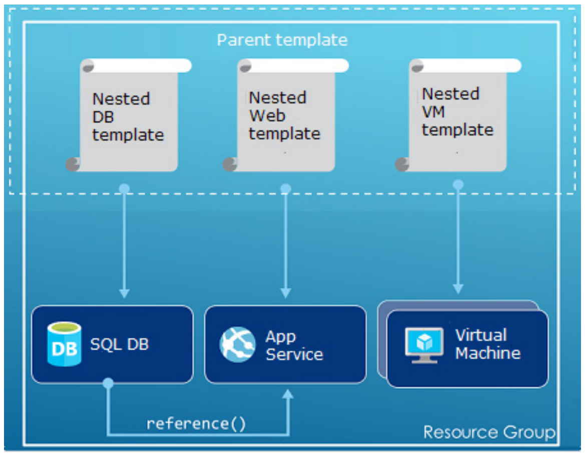
It converts the definition to the following REST API operation, which is sent to the Microsoft.Storage resource provider:

```
PUT
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/
{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/mystorage-
account?api-version=2016-01-01
REQUEST BODY
{
  "location": "westus",
  "properties": {
  }
  "sku": {
    "name": "Standard_LRS"
  },
  "kind": "Storage"
}
```

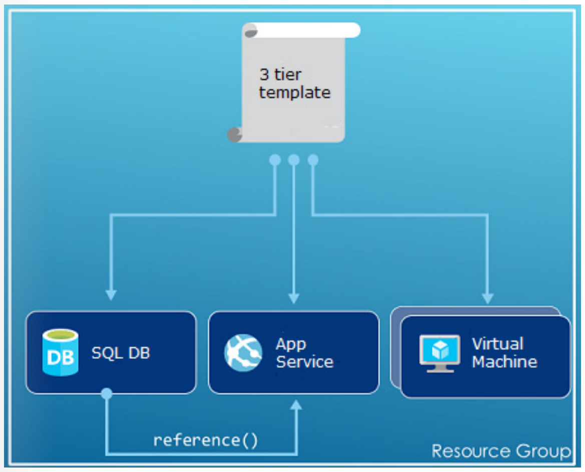
How you define templates and resource groups is entirely up to you and how you want to manage your solution. For example, you can deploy your three tier application through a single template to a single resource group.



But, you don't have to define your entire infrastructure in a single template. Often, it makes sense to divide your deployment requirements into a set of targeted, purpose-specific templates. You can easily reuse these templates for different solutions. To deploy a particular solution, you create a master template that links all the required templates. The following image shows how to deploy a three tier solution through a parent template that includes three nested templates.



If you envision your tiers having separate lifecycles, you can deploy your three tiers to separate resource groups. Notice the resources can still be linked to resources in other resource groups.



For information about nested templates, see **Using linked templates with Azure Resource Manager**[6].

Azure Resource Manager analyzes dependencies to ensure resources are created in the correct order. If one resource relies on a value from another resource (such as a virtual machine needing a storage account for disks), you set a dependency. For more information, see Defining dependencies in Azure Resource Manager templates.

You can also use the template for updates to the infrastructure. For example, you can add a resource to your solution and add configuration rules for the resources that are already deployed. If the template specifies creating a resource but that resource already exists, Azure Resource Manager performs an update instead of creating a new asset. Azure Resource Manager updates the existing asset to the same state as it would be as new.

Resource Manager provides extensions for scenarios when you need additional operations such as installing particular software that isn't included in the setup. If you're already using a configuration management service, like DSC, Chef or Puppet, you can continue working with that service by using extensions. For information about virtual machine extensions, see **About virtual machine extensions and features**[7].

Finally, the template becomes part of the source code for your app. You can check it in to your source code repository and update it as your app evolves. You can edit the template through Visual Studio.

Create ARM templates by using the Azure Portal

Learn how to create your first Azure Resource Manager template by generating one using the Azure portal, and the process of editing and deploying the template from the Azure portal. Resource Manager templates are JSON files that define the resources you need to deploy for your solution. The instructions in this tutorial create an Azure Storage account. You can use the same process to create other Azure resources.

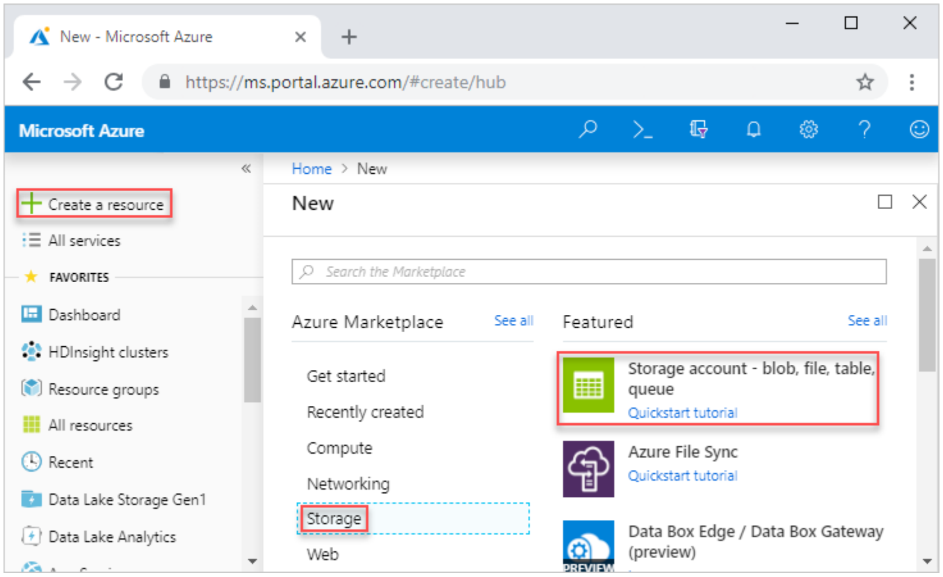
If you don't have an Azure subscription, create a **free account**[8] before you begin.

6 https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-linked-templates
7 https://docs.microsoft.com/en-us/azure/virtual-machines/windows/extensions-features?toc=%2fazure%2fvirtual-machines%2fwindows%2ftoc.json
8 https://azure.microsoft.com/free/

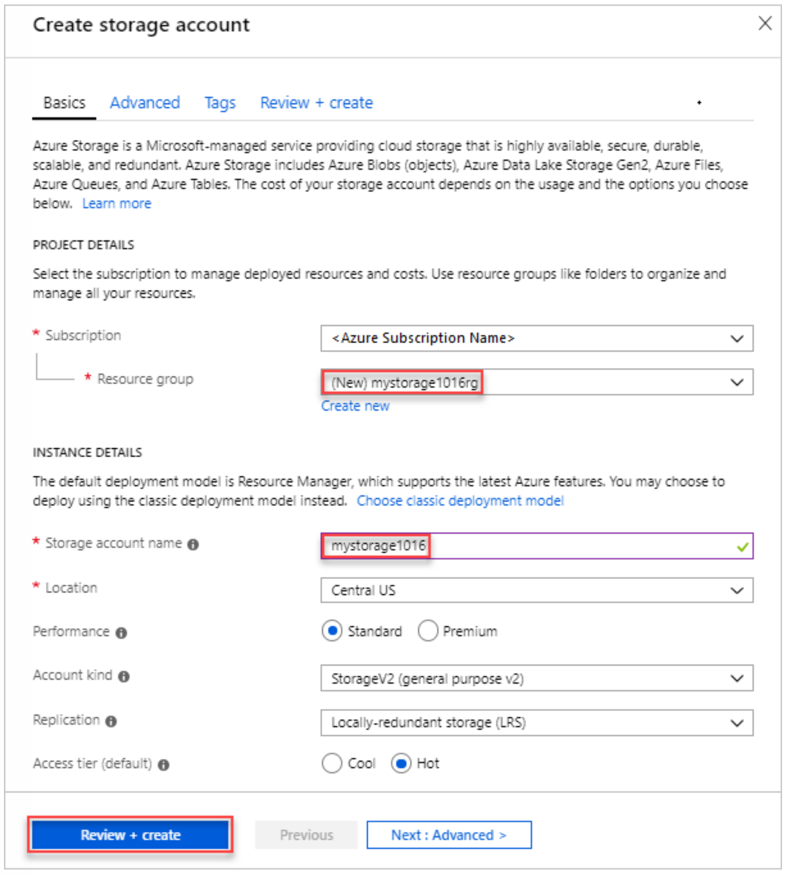
Generate a template using the portal

In this section, you create a storage account using the Azure portal. Before you deploy the storage account, you have the option to explore the template generated by the portal based on your configurations. You can save the template and reuse it in the future.

1. Sign in to the Azure portal: https://portal.azure.com/.
2. Select **Create a resource > Storage > Storage account - blob, file, table, queue**.



3.
4. Enter the following information.
   - **Resource group:** create a new Azure resource group with the name of your choice. On the screenshot, the resource group name is mystorage1016rg.
   - **Name:** give your storage account a unique name. On the screenshot, the name is mystorage1016.
   - You can use the default values for the rest of the properties.



Azure Storage is a Microsoft-managed service providing cloud storage that is highly available, secure, durable, scalable, and redundant. Azure Storage includes Azure Blobs (objects), Azure Data Lake Storage Gen2, Azure Files, Azure Queues, and Azure Tables. The cost of your storage account depends on the usage and the options you choose below. Learn more

PROJECT DETAILS

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

* Subscription	<Azure Subscription Name>
* Resource group	(New) mystorage1016rg
	Create new

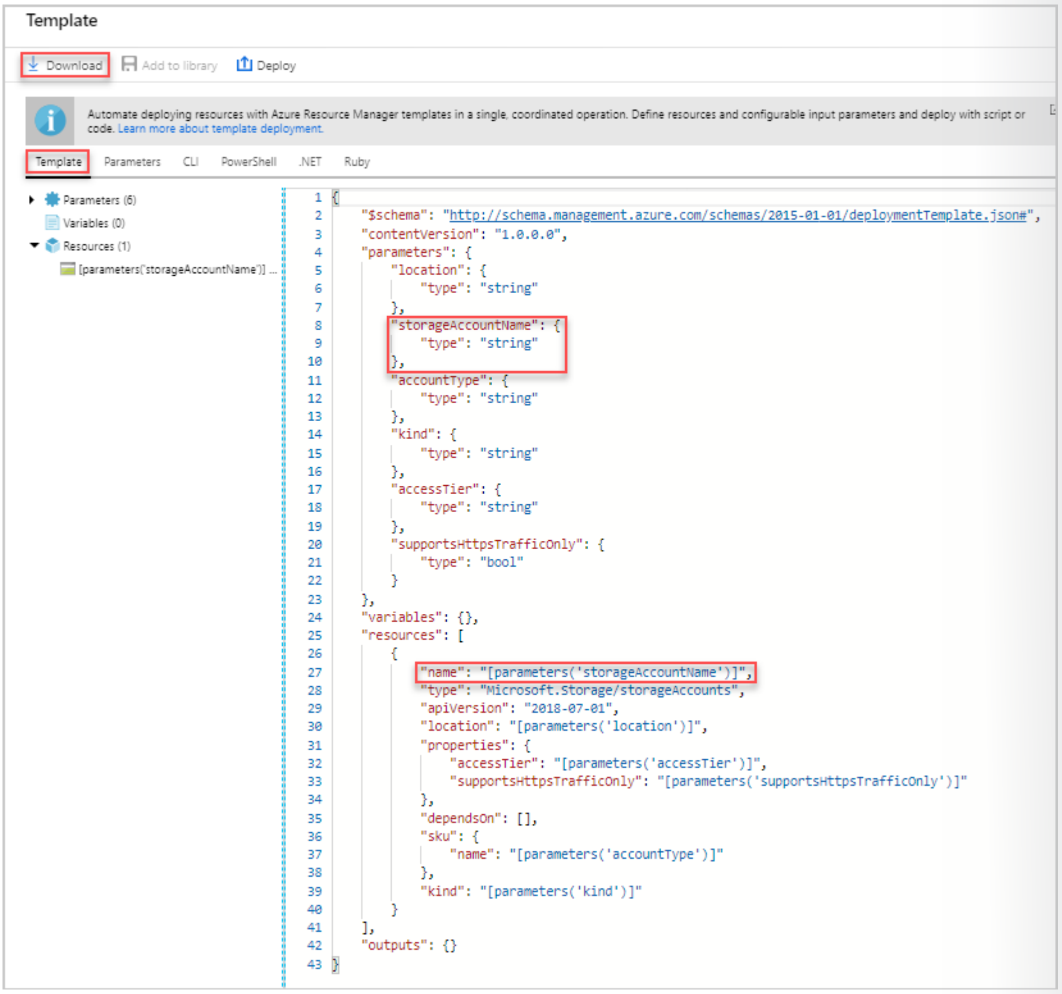
INSTANCE DETAILS

The default deployment model is Resource Manager, which supports the latest Azure features. You may choose to deploy using the classic deployment model instead. Choose classic deployment model

* Storage account name	mystorage1016
* Location	Central US
Performance	[✓ Standard] Premium
Account kind	StorageV2 (general purpose v2)
Replication	Locally-redundant storage (LRS)
Access tier (default)	Cool [✓ Hot]

-
- **Note:** Some of the exported templates require some edits before you can deploy them.

5. Select **Review + create** on the bottom of the screen.
6. Select **Download a template for automation** on the bottom of the screen. The portal shows the generated template:



Automate deploying resources with Azure Resource Manager templates in a single, coordinated operation. Define resources and configurable input parameters and deploy with script or code. Learn more about template deployment.

```
{
    "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "location": {
            "type": "string"
        },
        "storageAccountName": {
            "type": "string"
        },
        "accountType": {
            "type": "string"
        },
        "kind": {
            "type": "string"
        },
        "accessTier": {
            "type": "string"
        },
        "supportsHttpsTrafficOnly": {
            "type": "bool"
        }
    },
    "variables": {},
    "resources": [
        {
            "name": "[parameters('storageAccountName')]",
            "type": "Microsoft.Storage/storageAccounts",
            "apiVersion": "2018-07-01",
            "location": "[parameters('location')]",
            "properties": {
                "accessTier": "[parameters('accessTier')]",
                "supportsHttpsTrafficOnly": "[parameters('supportsHttpsTrafficOnly')]"
            },
            "dependsOn": [],
            "sku": {
                "name": "[parameters('accountType')]"
            },
            "kind": "[parameters('kind')]"
        }
    ],
    "outputs": {}
}
```

7.

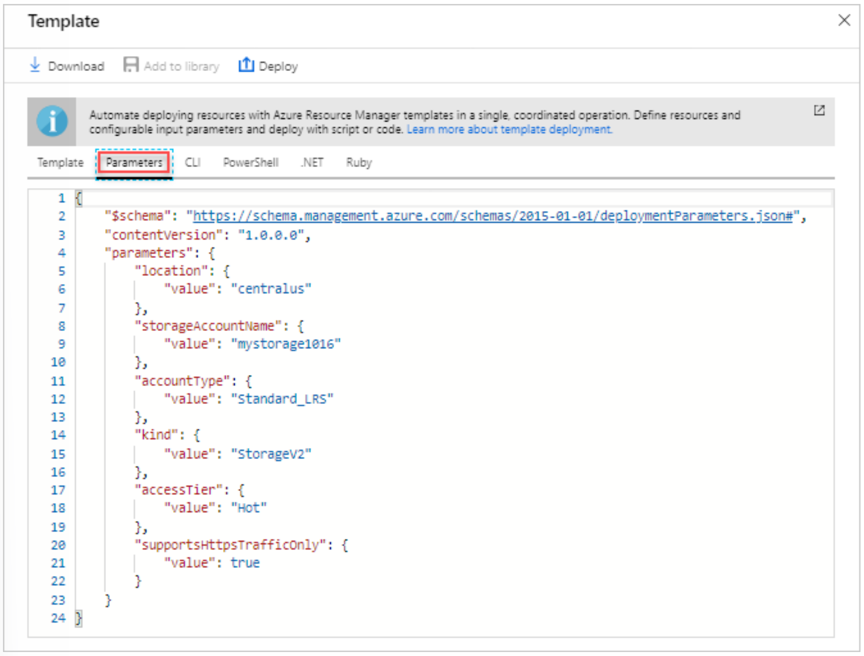
8. The main pane shows the template. It is a JSON file with four top-level elements - `schema`, `contentVersion`, `parameters` and `resources`.

9. There are six parameters defined. One of them is called **storageAccountName**. The second highlighted part shows how to use this parameter in the template. In the next section, you edit the template to use a generated name for the storage account.

10. In the template, one Azure resource is defined. The type is `[Microsoft.Storage/storageAccounts]`. See how the resource is defined, and the definition structure.

11. Select **Download**. Save **template.json** from the downloaded package to your computer. In the next section, you use a template deployment tool to edit the template.

12. Select the **Parameter** tab to see the values you provided for the parameters. Write down these values, you need them in the next section when you deploy the template.


Template
Download
Add to library
Deploy
Automate deploying resources with Azure Resource Manager templates in a single, coordinated operation. Define resources and configurable input parameters and deploy with script or code. Learn more about template deployment.
Template
Parameters
CLI
PowerShell
.NET
Ruby

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "location": {
            "value": "centralus"
        },
        "storageAccountName": {
            "value": "mystorage1016"
        },
        "accountType": {
            "value": "Standard_LRS"
        },
        "kind": {
            "value": "StorageV2"
        },
        "accessTier": {
            "value": "Hot"
        },
        "supportsHttpsTrafficOnly": {
            "value": true
        }
    }
}
```

13.

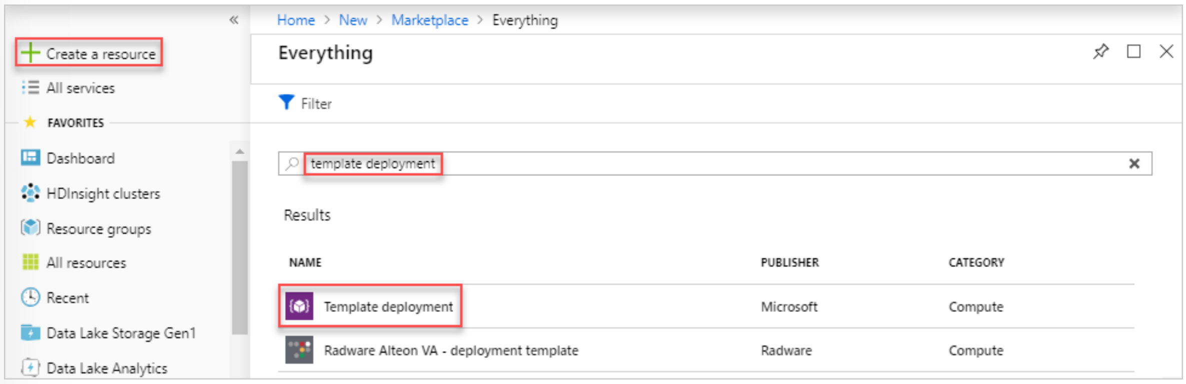
14. Using both the template and the parameters files, you can create an Azure storage account.

Edit and deploy the template

The Azure portal can be used to perform some basic template editing. In this tutorial, you use a portal tool called Template Deployment. To edit a more complex template, consider using Visual Studio Code which provides richer edit functionalities.

Azure requires that each Azure service has a unique name. The deployment fails if you enter a storage account name that already exists. To avoid this issue, you can use a template function call `uniques-tring()` to generate a unique storage account name.

1. In the Azure portal, select **Create a resource**.
2. In **Search the Marketplace**, type **template deployment**, and then press **ENTER**.
3. Select **Template deployment**.



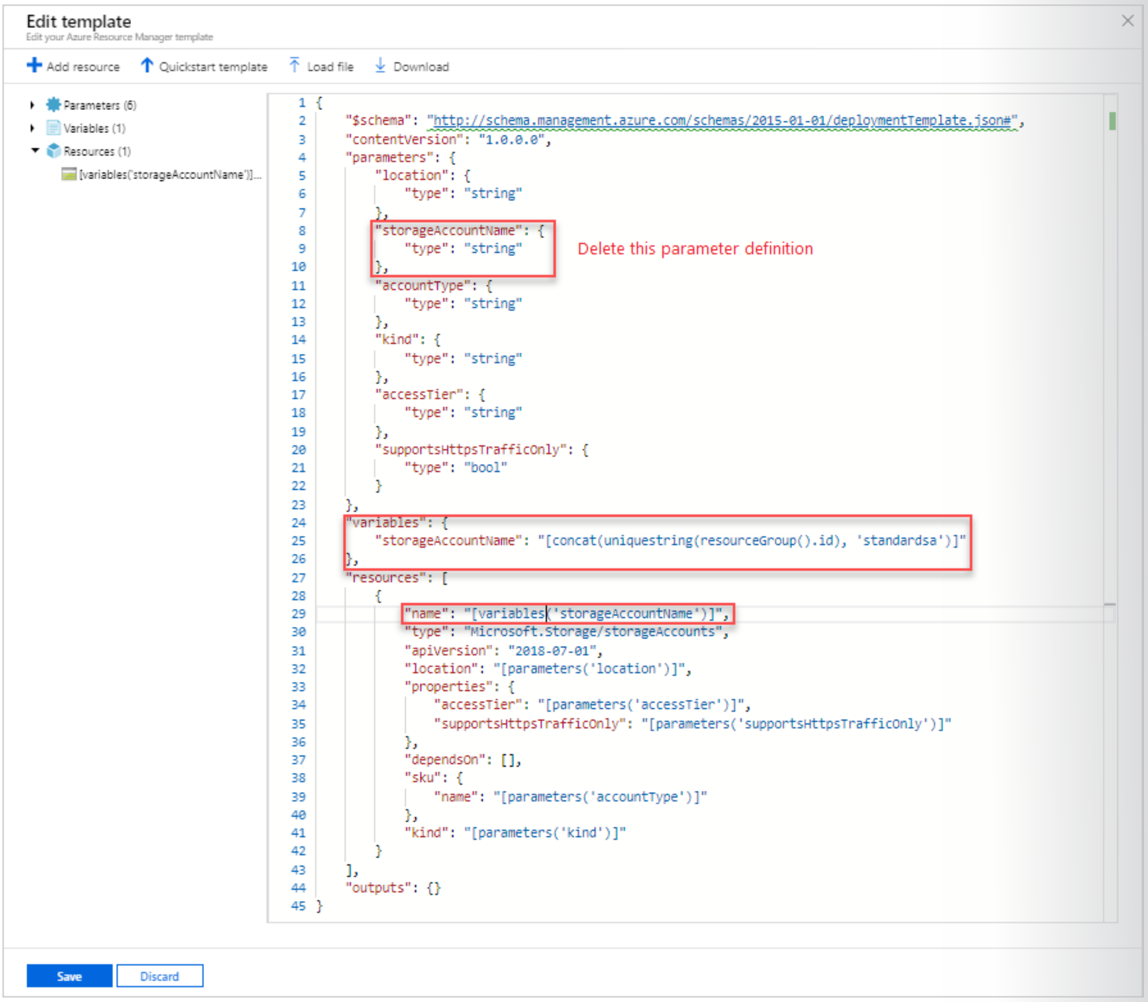
NAME	PUBLISHER	CATEGORY
Template deployment	Microsoft	Compute
Radware Alteon VA - deployment template	Radware	Compute

4.

5. Select **Create**.
6. Select **Build your own template** in the editor.

7. Select **Load file**, and then follow the instructions to load template.json you downloaded in the last section.
8. Add one variable as shown in the following screenshot:

```
"storageAccountName": "[concat(uniquestring(resourceGroup().id), 'standard-
sa')]"
```



```
{
    "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "location": {
            "type": "string"
        },
        "storageAccountName": {
            "type": "string"
        },
        "accountType": {
            "type": "string"
        },
        "kind": {
            "type": "string"
        },
        "accessTier": {
            "type": "string"
        },
        "supportsHttpsTrafficOnly": {
            "type": "bool"
        }
    },
    "variables": {
        "storageAccountName": "[concat(uniquestring(resourceGroup().id), 'standardsa')]"
    },
    "resources": [
        {
            "name": "[variables('storageAccountName')]",
            "type": "Microsoft.Storage/storageAccounts",
            "apiVersion": "2018-07-01",
            "location": "[parameters('location')]",
            "properties": {
                "accessTier": "[parameters('accessTier')]",
                "supportsHttpsTrafficOnly": "[parameters('supportsHttpsTrafficOnly')]"
            },
            "dependsOn": [],
            "sku": {
                "name": "[parameters('accountType')]"
            },
            "kind": "[parameters('kind')]"
        }
    ],
    "outputs": {}
}
```

9.
10. Two functions are used here: `concat()` and `uniqueString()`.
11. Remove the **storageAccountName** parameter highlighted in the previous screenshot.
12. Update the name element of the **Microsoft.Storage/storageAccounts** resource to use the newly defined variable instead of the parameter:

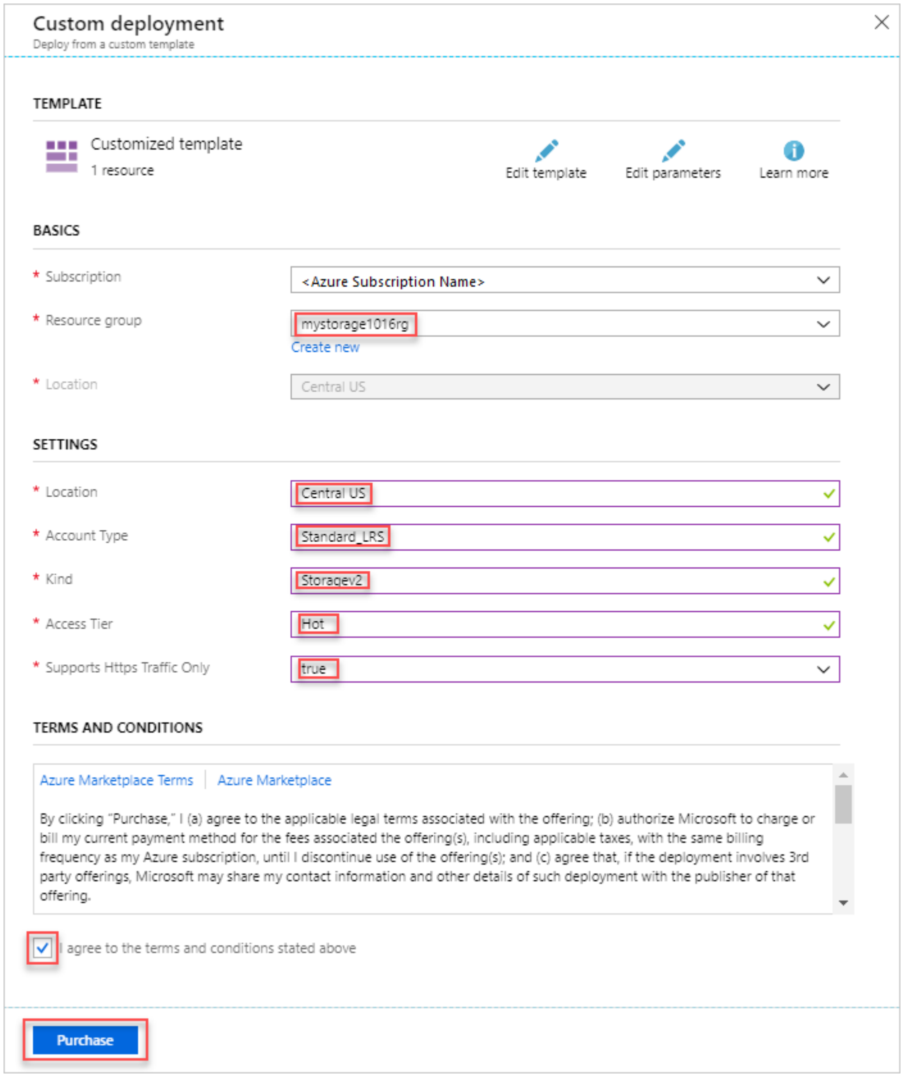
```
"name": "[variables('storageAccountName')]",
```

13. The final template shall look like:

```
{
    "$schema": "http://schema.management.azure.com/schemas/2015-01-01/
deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "location": {
            "type": "string"
        },
        "accountType": {
            "type": "string"
        },
        "kind": {
            "type": "string"
        },
        "accessTier": {
            "type": "string"
        },
        "supportsHttpsTrafficOnly": {
            "type": "bool"
        }
    },
    "variables": {
        "storageAccountName": "[concat(uniquestring(resourceGroup().id), 
'standardsa')]"
    },
    "resources": [
        {
            "name": "[variables('storageAccountName')]",
            "type": "Microsoft.Storage/storageAccounts",
            "apiVersion": "2018-07-01",
            "location": "[parameters('location')]",
            "properties": {
                "accessTier": "[parameters('accessTier')]",
                "supportsHttpsTrafficOnly": "[parameters('supportsHttpsTraf-
ficOnly')]"
            },
            "dependsOn": [],
            "sku": {
                "name": "[parameters('accountType')]"
            },
            "kind": "[parameters('kind')]"
        }
    ],
    "outputs": {}
}
```

14. Select **Save**.
15. Enter the following values:
    - **Resource group:** name your resource group with a unique name.
    - **Location:** select a location for the resource group.
    - **Location:** select a location for the storage account. You can use the same location as the resource group.
    - **Account Type:** Enter **Standard_LRS** for this tutorial.

- **Kind:** Enter **StorageV2** for this tutorial.
- **Access Tier:** Enter **Hot** for this tutorial.
- **Https Traffic Only Enabled:** Select true for this tutorial.
- **I agree to the terms and conditions stated above:** (select)
- Here is a screenshot of a sample deployment:



Custom deployment
Deploy from a custom template

TEMPLATE

Customized template
1 resource

BASICS

* Subscription	<Azure Subscription Name>
* Resource group	mystorage1016rg
	Create new
* Location	Central US

SETTINGS

* Location	Central US
* Account Type	Standard_LRS
* Kind	Storagev2
* Access Tier	Hot
* Supports Https Traffic Only	true

TERMS AND CONDITIONS

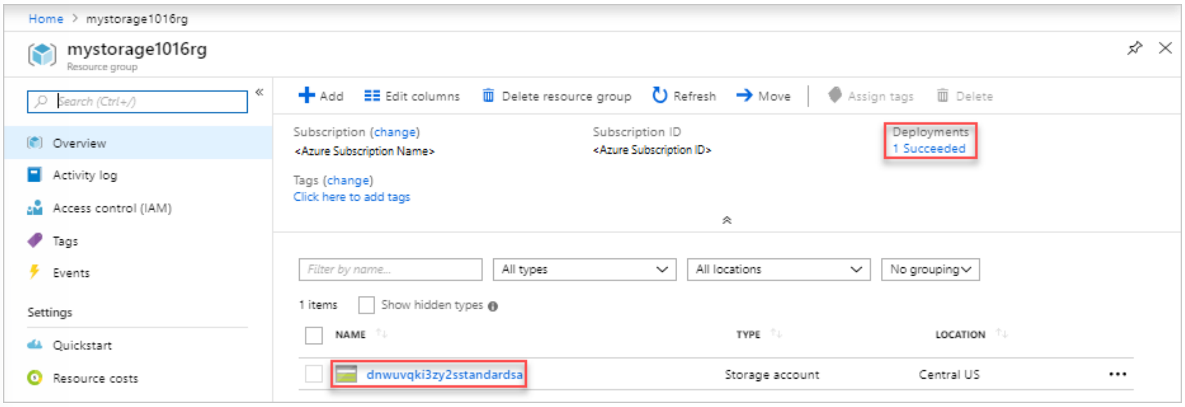
Azure Marketplace Terms | Azure Marketplace

By clicking "Purchase," I (a) agree to the applicable legal terms associated with the offering; (b) authorize Microsoft to charge or bill my current payment method for the fees associated the offering(s), including applicable taxes, with the same billing frequency as my Azure subscription, until I discontinue use of the offering(s); and (c) agree that, if the deployment involves 3rd party offerings, Microsoft may share my contact information and other details of such deployment with the publisher of that offering.

[✓] I agree to the terms and conditions stated above

-

16. Select **Purchase**.
17. Select the bell icon (notifications) from the top of the screen to see the deployment status. Wait until the deployment is completed.
18. Select **Go to resource group** from the notification pane. You shall see a screen similar to:



19.

20. You can see the deployment status was successful, and there is only one storage account in the resource group. The storage account name is a unique string generated by the template.

Clean up resources

When the Azure resources are no longer needed, clean up the resources you deployed by deleting the resource group.

1. In the Azure portal, select **Resource group** on the left menu.
2. Enter the resource group name in the **Filter by name** field.
3. Select the resource group name. You shall see the storage account in the resource group.
4. Select **Delete resource group** in the top menu.

Create ARM templates by using Visual Studio Code

Learn how to use Visual Studio code and the Azure Resource Manager Tools extension to create and edit Azure Resource Manager templates. You can create Resource Manager templates in Visual Studio Code without the extension, but the extension provides autocomplete options that simplify template development.

If you don't have an Azure subscription, create a free account before you begin. (https://azure.microsoft.com/free/)

Prerequisites

To complete this article, you need:

- Visual Studio Code. You can download a copy here: https://code.visualstudio.com/.
- Resource Manager Tools extension.

Follow these steps to install the Resource Manager Tools extension:

1. Open Visual Studio Code.
2. Press **CTRL+SHIFT+X** to open the Extensions pane.
3. Search for **Azure Resource Manager Tools**, and then select **Install**.
4. Select **Reload** to finish the extension installation.

Open a Quickstart template

Instead of creating a template from scratch, you open a template from **Azure Quickstart Templates**[9]. Azure QuickStart Templates is a repository for Resource Manager templates.

The template used in this tutorial is called **Create a standard storage account**[10]. The template defines an Azure Storage account resource.

1. From Visual Studio Code, select **File > Open File**.
2. In **File name**, paste the following URL:

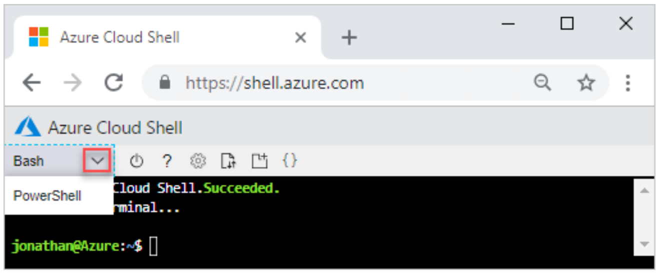
```
https://raw.githubusercontent.com/Azure/azure-quickstart-templates/mas-
ter/101-storage-account-create/azuredeploy.json
```

3. Select **Open** to open the file.
4. Select **File > Save As** to save the file as azuredeploy.json to your local computer.

Deploy the template

There are many methods for deploying templates. In this tutorial , you use the Azure Cloud shell. Cloud Shell supports both Azure CLI and Azure PowerShell, the steps below focus on Azure CLI.

1. Sign in to the **Azure Cloud shell**[11]

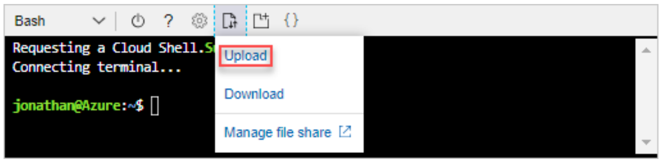


```
Cloud Shell.Succeeded.
rminal...

jonathan@Azure:~$
```

2.

3. On the upper left corner of the Cloud shell, it shows either PowerShell or Bash. To use CLI, you need to open a Bash session.
4. Select **Upload/download** files, and then select **Upload**.



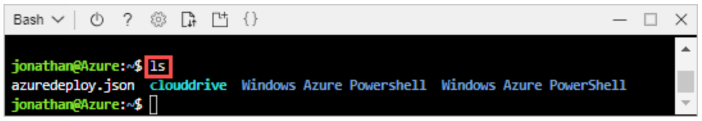
```
Requesting a Cloud Shell.S
Connecting terminal...

jonathan@Azure:~$
```

5.

6. You must upload the template file before you can deploy it from the shell.
7. Select the file you saved in the previous section. The default name is **azuredeploy.json**.
8. From the Cloud shell, run the **ls** command to verify the file is uploaded successfully. You can also use the **cat** command to verify the template content. The following image shows running the command from Bash. You use the same commands from a PowerShell session.

9 https://azure.microsoft.com/resources/templates/
10 https://azure.microsoft.com/resources/templates/101-storage-account-create/
11 https://shell.azure.com/



```
jonathan@Azure:~$ ls
azuredeploy.json  clouddrive  Windows Azure Powershell  Windows Azure PowerShell
jonathan@Azure:~$
```

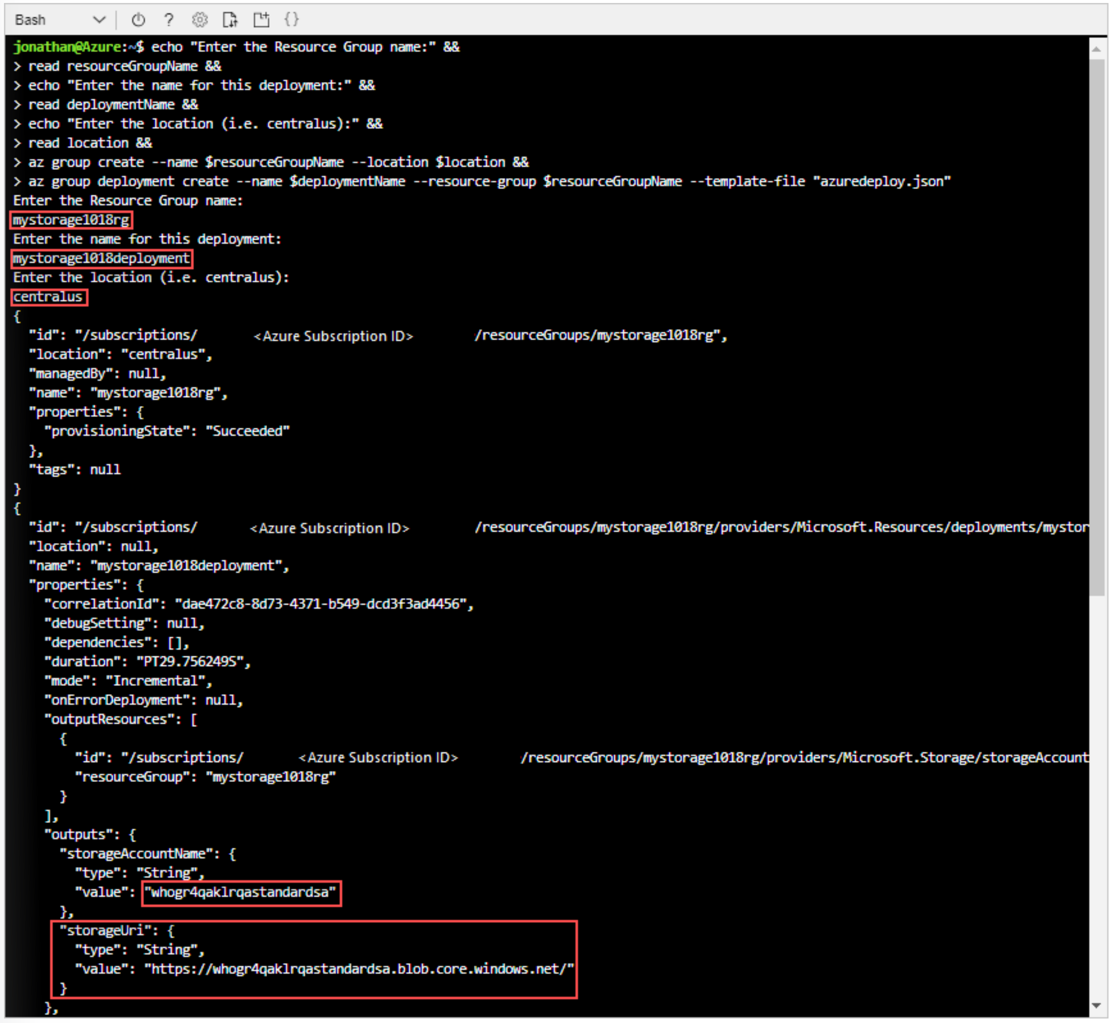
9.

10. From the Cloud shell, run the following commands. Select the tab to show the PowerShell code or the CLI code.

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
echo "Enter the name for this deployment:" &&
read deploymentName &&
echo "Enter the location (i.e. centralus):" &&
read location &&
az group create --name $resourceGroupName --location $location &&
az group deployment create --name $deploymentName --resource-group $re-
sourceGroupName --template-file "azuredeploy.json"
```

11. Update the template file name if you save the file to a name other than **azuredeploy.json**.

12. The following screenshot shows a sample deployment:



```
jonathan@Azure:~$ echo "Enter the Resource Group name:" &&
> read resourceGroupName &&
> echo "Enter the name for this deployment:" &&
> read deploymentName &&
> echo "Enter the location (i.e. centralus):" &&
> read location &&
> az group create --name $resourceGroupName --location $location &&
> az group deployment create --name $deploymentName --resource-group $resourceGroupName --template-file "azuredeploy.json"
Enter the Resource Group name:
mystorage1018rg
Enter the name for this deployment:
mystorage1018deployment
Enter the location (i.e. centralus):
centralus
{
  "id": "/subscriptions/      <Azure Subscription ID>       /resourceGroups/mystorage1018rg",
  "location": "centralus",
  "managedBy": null,
  "name": "mystorage1018rg",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null
}
{
  "id": "/subscriptions/      <Azure Subscription ID>       /resourceGroups/mystorage1018rg/providers/Microsoft.Resources/deployments/mystor
  "location": null,
  "name": "mystorage1018deployment",
  "properties": {
    "correlationId": "dae472c8-8d73-4371-b549-dcd3f3ad4456",
    "debugSetting": null,
    "dependencies": [],
    "duration": "PT29.756249S",
    "mode": "Incremental",
    "onErrorDeployment": null,
    "outputResources": [
      {
        "id": "/subscriptions/      <Azure Subscription ID>       /resourceGroups/mystorage1018rg/providers/Microsoft.Storage/storageAccount
        "resourceGroup": "mystorage1018rg"
      }
    ],
    "outputs": {
      "storageAccountName": {
        "type": "String",
        "value": "whogr4qaklrqastandardsa"
      },
      "storageUri": {
        "type": "String",
        "value": "https://whogr4qaklrqastandardsa.blob.core.windows.net/"
      }
    },
```

13.

14. The storage account name and the storage URL in the outputs section are highlighted on the screenshot. You need the storage account name in the next step.
15. Run the following CLI or PowerShell command to list the newly created storage account:

```
echo "Enter the Resource Group name:" &&
read resourceGroupName &&
echo "Enter the Storage Account name:" &&
read storageAccountName &&
az storage account show --resource-group $resourceGroupName --name $stor-
ageAccountName
```

Clean up resources

When the Azure resources are no longer needed, clean up the resources you deployed by deleting the resource group.

1. From the Azure portal, select **Resource group** from the left menu.
2. Enter the resource group name in the **Filter by name** field.
3. Select the resource group name. You shall see a total of six resources in the resource group.
4. Select **Delete resource group** from the top menu.

Configure Azure Disk Encryption for VMs

Encryption options for protecting VMs

Suppose your company's trading partners have security policies who require their trading data is protected with strong encryption. You use a B2B application that runs on your Windows servers, and stores data on the server data disk. Now that you're transitioning to the cloud, you need to demonstrate to your trading partners that data stored on your Azure VMs cannot be accessed by unauthorized users, devices, or applications. You need to decide on a strategy for implementing encryption of your B2B data.

Your audit requirements dictate that your encryption keys be managed in-house, and not by any third party. You're also concerned that the performance and manageability of your Azure-based servers is maintained. So before you implement encryption, you want to be sure that there won't be a performance hit.

What is encryption?

Encryption is about converting meaningful information into something that appears meaningless, such as a random sequence of letters and numbers. The process of encryption uses some form of **key** as part of the algorithm that creates the encrypted data. A key is also needed to perform the decryption. Keys may be ***symmetric***, where the same key is used for encryption and decryption, or ***asymmetric***, where different keys are used. An example of the latter is the **public-private** key pairs used in digital certificates.

Symmetric encryption

Algorithms that use symmetric keys, such as Advanced Encryption Standard (AES), are typically faster than public key algorithms, and are often used for protecting large data stores. Because there's only one key, procedures must be in place to prevent the key from becoming publicly known.

Asymmetric encryption

With asymmetric algorithms, only the private key member of the pair must be kept private and secure; as its name suggests, the public key can be made available to anyone without compromising the encrypted data. The downside of public key algorithms, however, is that they're much slower than symmetric algorithms, and cannot be used to encrypt large amounts of data.

Key management

In Azure, your encryption keys can be managed by Microsoft or the customer. Often the demand for customer-managed keys comes from organizations that need to demonstrate compliance with HIPAA, or other regulations. Such compliance may require that access to keys is logged, and that regular key changes are made and recorded.

Azure disk encryption technologies

The main encryption-based disk protection technologies for Azure VMs are:

- Storage Service Encryption (SSE)
- Azure Disk Encryption (ADE)

Storage Service Encryption

Azure Storage Service Encryption (SSE) is an encryption service built into Azure used to protect data at rest. The Azure storage platform automatically encrypts data before it's stored to several storage services, including Azure Managed Disks. Encryption is enabled by default using 256-bit AES encryption, and is managed by the storage account administrator.

Azure Disk Encryption

Azure Disk Encryption (ADE) is managed by the VM owner. It controls the encryption of Windows and Linux VM-controlled disks, using **BitLocker** on Windows VMs and **DM-Crypt** on Linux VMs. BitLocker Drive Encryption is a data protection feature that integrates with the operating system, and addresses the threats of data theft or exposure from lost, stolen, or inappropriately decommissioned computers. Similarly, DM-Crypt encrypts data at rest for Linux before writing to storage.

ADE ensures that all data on VM disks are encrypted at rest in Azure storage, and ADE is required for VMs backed up to the Recovery Vault.

With ADE, VMs boot under customer-controlled keys and policies. ADE is integrated with Azure Key Vault for the management of these disk-encryption keys and secrets.

Note: ADE does not support the encryption of Basic tier VMs, and you cannot use an on-premises Key Management Service (KMS) with ADE.

When to use encryption?

Computer data is at risk when it's in transit (transmitted across the Internet or other network), and when it's at rest (saved to a storage device). The at-rest scenario is the primary concern when protecting data on Azure VM disks. For example, someone might download the Virtual Hard Disk (VHD) file associated with an Azure VM, and save it on their laptop. If the VHD is not encrypted, the contents of the VHD are potentially accessible to anyone who can mount the VHD file on their computer.

For operating system (OS) disks, data such as passwords are encrypted automatically, so even if the VHD is not itself encrypted, it's not easy for such information to be accessed. Applications may also automatically encrypt their own data. However, even with such protections, if someone with malicious intent were to gain access to a data disk, and the disk itself was not encrypted, they might then be in a position to exploit any known weaknesses in that application's data protection. With disk encryption in place, such exploits are not possible.

Storage Service Encryption (SSE) is part of Azure itself, and there should be no noticeable performance impact on the VM disk IO when using SSE. Managed disks with SSE are now the default, and there should be no reason to change it. Azure Disk Encryption (ADE) makes use of VM operating system tools (BitLocker and DM-Crypt), so the VM itself has to do some work when encryption or decryption on VM disks is being performed. The impact of this additional VM CPU activity is typically negligible, except in certain situations. For instance, if you have a CPU-intensive application, there may be a case for leaving the OS disk unencrypted to maximize performance. In a situation such as this, you can store application data on a separate encrypted data disk, getting you the performance you need without compromising security.

Azure provides two complementary encryption technologies that are used to secure Azure VM disks. These technologies, SSE and ADE, encrypt at different layers, and serve different purposes. Both use AES 256-bit encryption. Using both technologies provides a defense-in-depth protection against unauthorized access to your Azure storage, and to specific VHDs.

Encrypt existing VM disks

Suppose your company has decided to implement Azure Disk Encryption (ADE) across all VMs. You need to evaluate how to roll out encryption to existing VM volumes. Here, we'll look at the requirements for ADE, and the steps involved in encrypting disks on existing Linux and Windows VMs.

Azure Disk Encryption prerequisites

Before you can encrypt your VM disks, you need to:

1. Create a key vault.
2. Set the key vault access policy to support disk encryption.
3. Use the key vault to store the encryption keys for ADE.

Azure Key Vault

The encryption keys used by ADE can be stored in Azure Key Vault. Azure Key Vault is a tool for securely storing and accessing secrets. A secret is anything that you want to tightly control access to, such as API keys, passwords, or certificates. This provides highly available and scalable secure storage, as defined in Federal Information Processing Standards (FIPS) 140-2 Level 2 validated Hardware Security Modules (HSMs). Using Key Vault, you keep full control of the keys used to encrypt your data, and you can manage and audit your key usage.

Note: Azure Disk Encryption requires that your key vault and your VMs are in the same Azure region; this ensures that encryption secrets do not cross regional boundaries.

You can configure and manage your key vault with:

Powershell

```
New-AzureRmKeyVault -Location <location> `
    -ResourceGroupName <resource-group> `
    -VaultName "myKeyVault" `
    -EnabledForDiskEncryption
```

Azure CLI

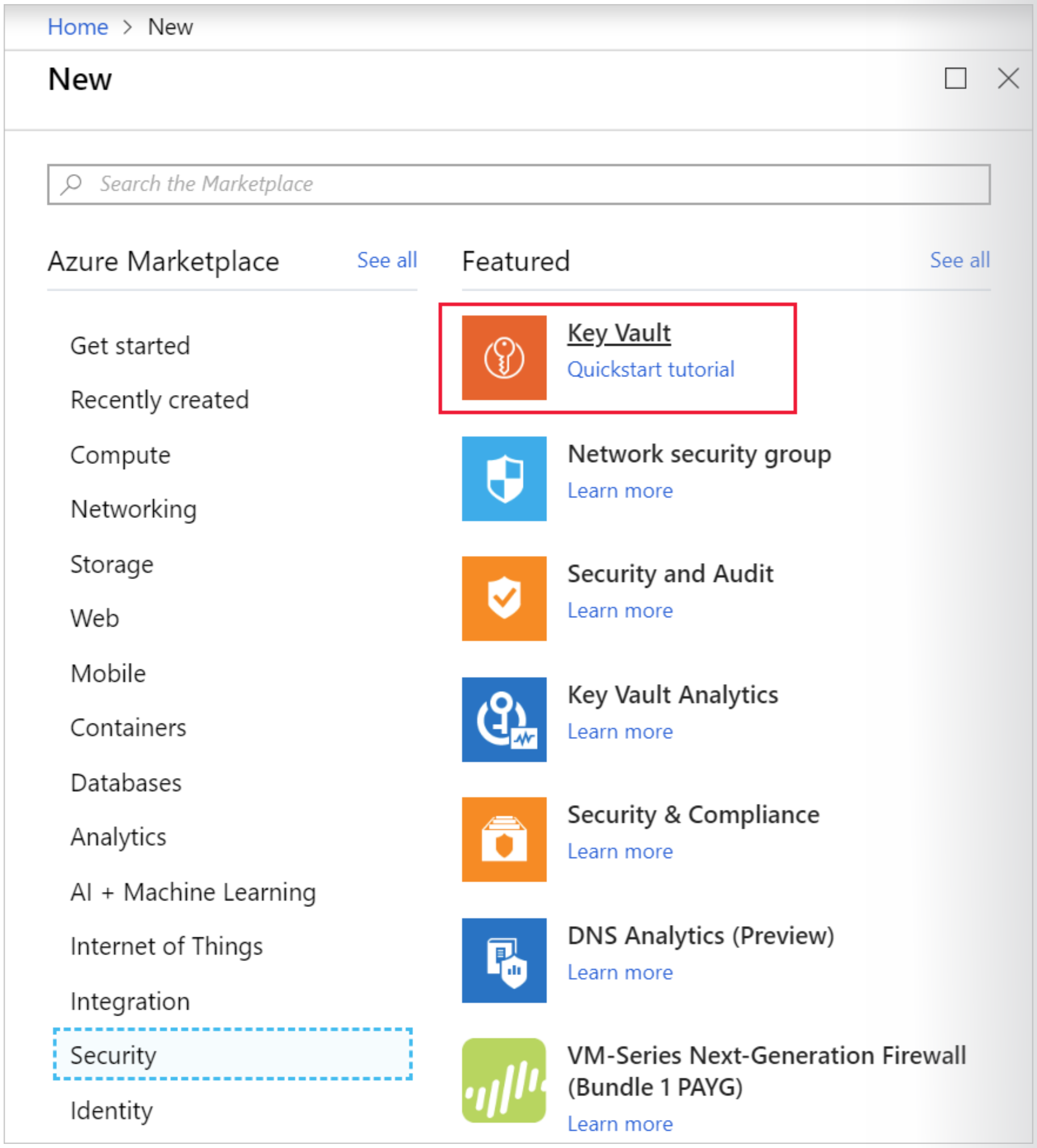
```
az keyvault create \
    --name "myKeyVault" \
    --resource-group <resource-group> \
    --location <location> \
    --enabled-for-disk-encryption True
```

Azure portal

An Azure Key Vault is a resource that can be created in the Azure portal using the normal resource creation process.

1. Select **Create a resource** in the sidebar on the left.

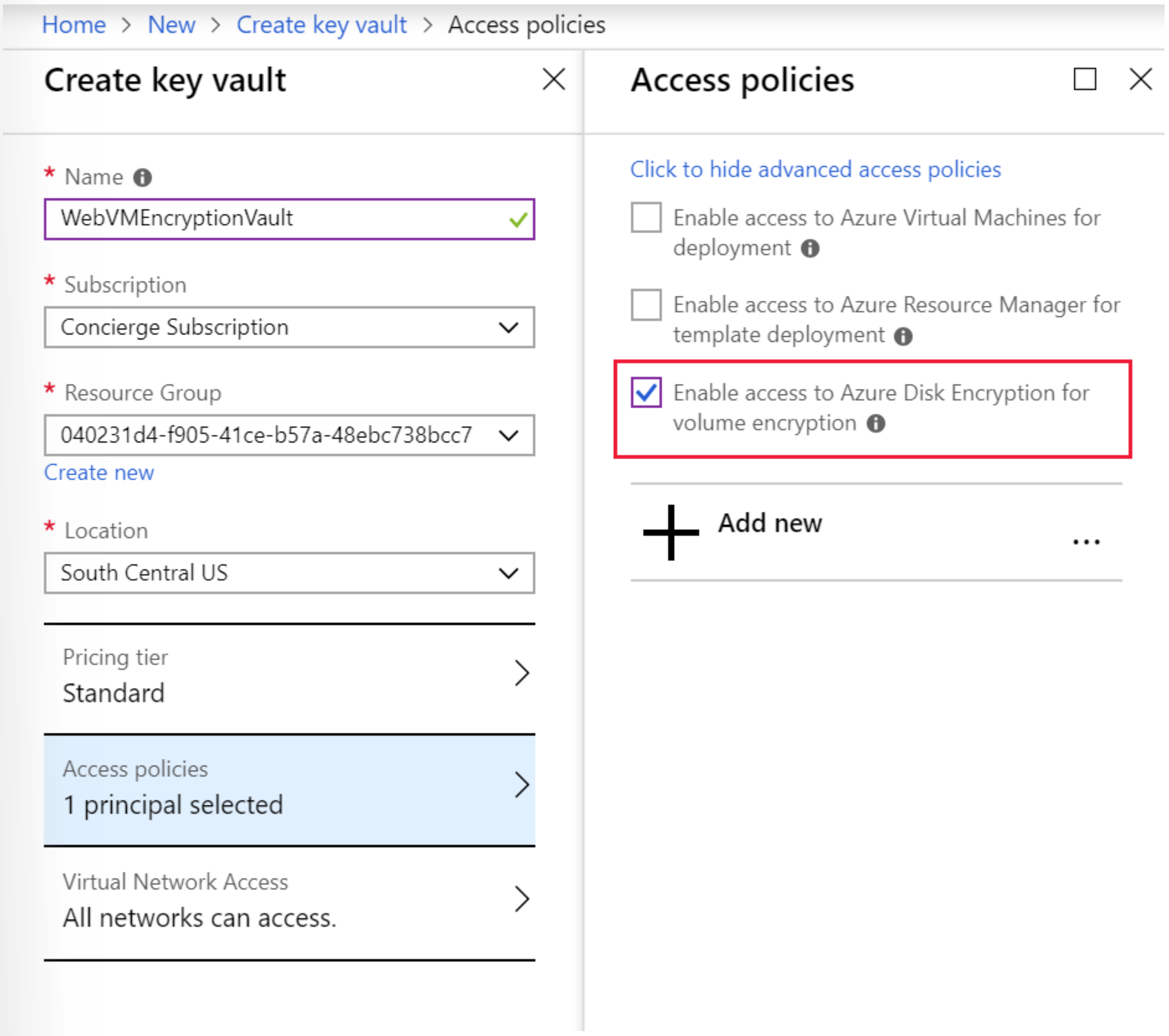
2. Search for “Key vault”. Select **Create** in the details window.



3.

4. Enter the details for the new Key Vault:
   - Enter a **Name** for the Key Vault
   - Select the subscription to place it in (defaults to your current subscription).
   - Select a **Resource Group**, or create a new resource group to own the Key Vault.
   - Select a **Location** for the Key Vault. Make sure to select the location the VM is in.
   - You can choose either Standard or Premium for the pricing tier. The main difference is that the premium tier allows for Hardware-encryption backed keys.

5. You must change the Access policies to support Disk Encryption. By default it adds your account to the policy.
   - Select **Access policies**.
   - Select **Advanced access policies**.
   - Check the **Enable access to Azure Disk Encryption for volume encryption**.
   - You can remove your account if you like, it's not necessary if you only intend to use the Key Vault for disk encryption.
   - Select **OK** to save the changes.



Home > New > Create key vault > Access policies

Create key vault

* Name: WebVMEncryptionVault

* Subscription: Concierge Subscription

* Resource Group: 040231d4-f905-41ce-b57a-48ebc738bcc7

Create new

* Location: South Central US

Pricing tier: Standard

Access policies: 1 principal selected

Virtual Network Access: All networks can access.

Access policies

Click to hide advanced access policies

- [ ] Enable access to Azure Virtual Machines for deployment
- [ ] Enable access to Azure Resource Manager for template deployment
- [✓] Enable access to Azure Disk Encryption for volume encryption

1. Select **Create** to create the new Key Vault.

Enabling access policies in the key vault

Azure needs access to the encryption keys or secrets in your key vault to make them available to the VM for booting and decrypting the volumes. We covered this for the portal when we changed the **Advanced access policies** above.

There are three policies you can enable.

- **Disk encryption** - Required for Azure Disk encryption.
- **Deployment** - (Optional) Enables the Microsoft.Compute resource provider to retrieve secrets from this key vault when this key vault is referenced in resource creation, for example when creating a virtual machine.
- **Template deployment** - (Optional) Enables Azure Resource Manager to get secrets from this key vault when this key vault is referenced in a template deployment.

Here's how to enable the disk encryption policy. The other two are similar but use different flags.

PowerShell

```
Set-AzureRmKeyVaultAccessPolicy -VaultName <keyvault-name> -ResourceGroupN-
ame <resource-group> -EnabledForDiskEncryption
```

Azure CLI

```
az keyvault update --name <keyvault-name> --resource-group <resource-group>
--enabled-for-disk-encryption "true"
```

Encrypting an existing VM disk

Once you have the Key Vault setup, you can encrypt the VM using either Azure CLI or Azure PowerShell. The first time you encrypt a Windows VM, you can choose to encrypt either all disks or the OS disk only. On some Linux distributions, only the data disks may be encrypted. To be eligible for encryption, your Windows disks must be formatted as NTFS volumes.

Warning: You must take a snapshot or a backup of managed disks before you can turn on encryption. The SkipVmBackup flag specified below tells the tool that the backup is complete on managed disks. Without the backup, you will be unable to recover the VM if the encryption fails for some reason.

With PowerShell, use the `Set-AzureRmVmDiskEncryptionExtension` cmdlet to enable encryption.

```
Set-AzureRmVmDiskEncryptionExtension `
    -ResourceGroupName <resource-group> `
    -VMName <vm-name> `
    -VolumeType [All | OS | Data]
    -DiskEncryptionKeyVaultId <keyVault.ResourceId> `
    -DiskEncryptionKeyVaultUrl <keyVault.VaultUri> `
     -SkipVmBackup
```

For the Azure CLI, use the `az vm encryption enable` command to enable encryption.

```
az vm encryption enable \
    --resource-group <resource-group> \
    --name <vm-name> \
    --disk-encryption-keyvault <keyvault-name> \
    --volume-type [all | os | data] \
    --skipvmbackup
```

Viewing the status of the disk

You can check whether specific disks are encrypted or not.

PowerShell

```
Get-AzureRmVmDiskEncryptionStatus  -ResourceGroupName <resource-group>
-VMName <vm-name>
```

Azure CLI

```
az vm encryption show --resource-group <resource-group> --name <vm-name>
```

Both of these commands will return the status of each disk attached to the specified VM.

Decrypting drives

You can reverse the encryption through PowerShell using `Disable-AzureRmVMDiskEncryption`.

```
Disable-AzureRmVMDiskEncryption -ResourceGroupName <resource-group> -VMName
<vm-name>
```

For the Azure CLI, use the `vm encryption disable` command.

```
az vm encryption disable --resource-group <resource-group> --name <vm-name>
```

These commands disable encryption for volumes of type all for the specified virtual machine. Just like the encrypt version, you can specify a `-VolumeType` parameter `[All | OS | Data]` to decide what disks to decrypt. It defaults to `All` if not supplied.

Warning: Disabling data disk encryption on Windows VM when both OS and data disks have been encrypted doesn't work as expected. You must disable encryption on all disks instead.

Automate secure VM deployments by using Azure Resource Manager templates

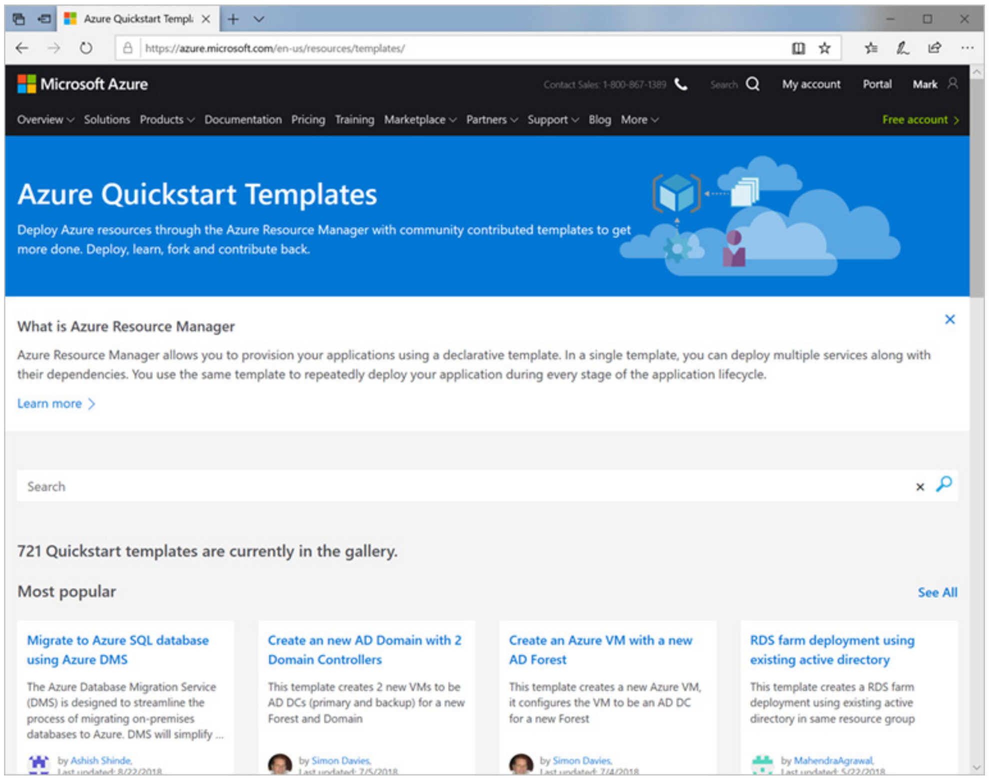
Suppose your company is deploying several servers as part of their cloud transition. VM disks must be encrypted during the deployment, so there's no time when the disks are vulnerable. You want to automate this process, and have to modify the Azure Resource Manager templates to automatically enable encryption.

Here, we'll look at how to use an Azure Resource Manager template to automatically enable encryption for new Windows VMs.

What are Azure Resource Manager templates?

Resource Manager templates are JSON files used to define a set of resources to deploy to Azure. You can write them from scratch, and for some Azure resources, including VMs, you can use the Azure portal to generate them. You'll need to complete the required information for a manual VM deployment, but instead of deploying the VM to Azure, you save the template. You can then reuse the template to create that specific VM configuration.

There are example templates available in docs to automate all sorts of administrative tasks. In fact, we could have used one of these templates to encrypt our VM that we just did manually!



Azure Quickstart Templates

Deploy Azure resources through the Azure Resource Manager with community contributed templates to get more done. Deploy, learn, fork and contribute back.

What is Azure Resource Manager

Azure Resource Manager allows you to provision your applications using a declarative template. In a single template, you can deploy multiple services along with their dependencies. You use the same template to repeatedly deploy your application during every stage of the application lifecycle.

Learn more

Search

721 Quickstart templates are currently in the gallery.

Most popular See All

Migrate to Azure SQL database using Azure DMS
The Azure Database Migration Service (DMS) is designed to streamline the process of migrating on-premises databases to Azure. DMS will simplify ...
by Ashish Shinde, Last updated: 8/22/2018

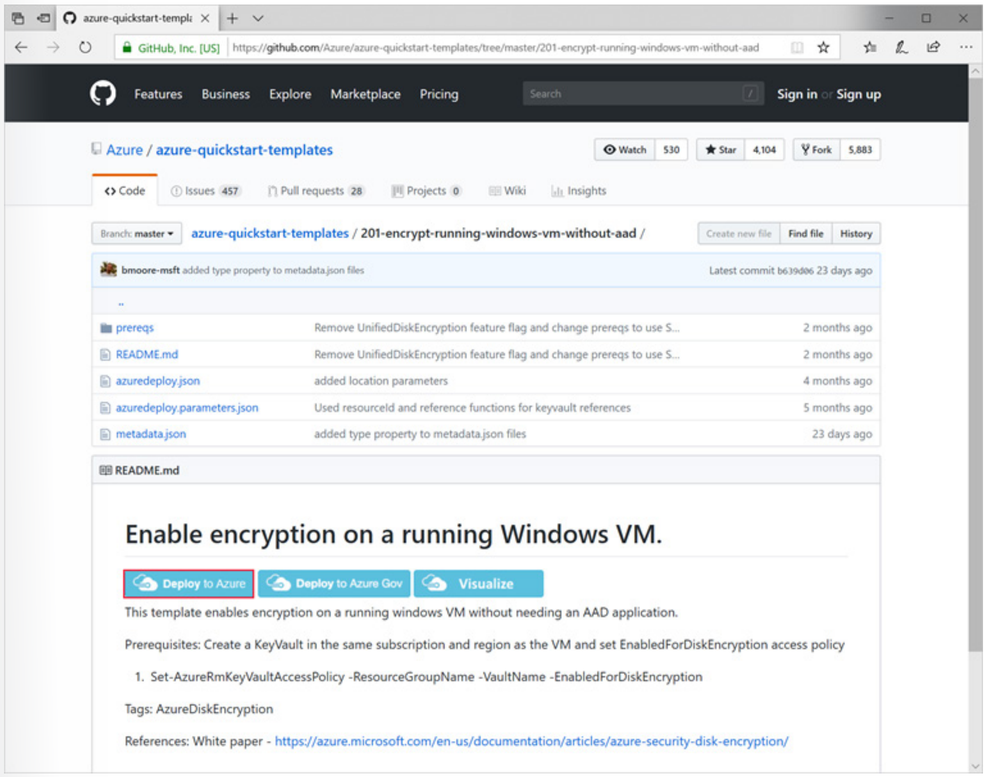
Create an new AD Domain with 2 Domain Controllers
This template creates 2 new VMs to be AD DCs (primary and backup) for a new Forest and Domain
by Simon Davies, Last updated: 7/5/2018

Create an Azure VM with a new AD Forest
This template creates a new Azure VM, it configures the VM to be an AD DC for a new Forest
by Simon Davies, Last updated: 7/4/2018

RDS farm deployment using existing active directory
This template creates a RDS farm deployment using existing active directory in same resource group
by MahendraAgrawal, Last updated: 5/22/2018

Using GitHub templates

The actual template source is stored in GitHub. You can browse to a template in GitHub and deploy right to Azure from the page.


https://github.com/Azure/azure-quickstart-templates/tree/master/201-encrypt-running-windows-vm-without-aad

Azure / azure-quickstart-templates

Branch: master · azure-quickstart-templates / 201-encrypt-running-windows-vm-without-aad /

bmoore-msft added type property to metadata.json files — Latest commit b639d06 23 days ago

..		
prereqs	Remove UnifiedDiskEncryption feature flag and change prereqs to use S…	2 months ago
README.md	Remove UnifiedDiskEncryption feature flag and change prereqs to use S…	2 months ago
azuredeploy.json	added location parameters	4 months ago
azuredeploy.parameters.json	Used resourceId and reference functions for keyvault references	5 months ago
metadata.json	added type property to metadata.json files	23 days ago

README.md

Enable encryption on a running Windows VM.

This template enables encryption on a running windows VM without needing an AAD application.

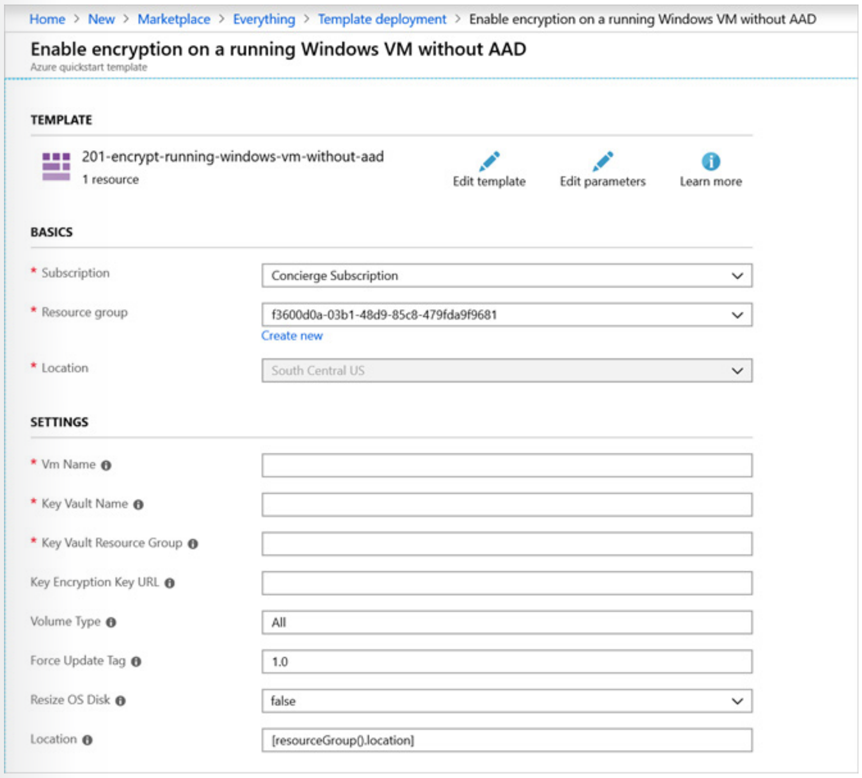
Prerequisites: Create a KeyVault in the same subscription and region as the VM and set EnabledForDiskEncryption access policy

1. Set-AzureRmKeyVaultAccessPolicy -ResourceGroupName -VaultName -EnabledForDiskEncryption

Tags: AzureDiskEncryption

References: White paper - https://azure.microsoft.com/en-us/documentation/articles/azure-security-disk-encryption/

When the template is deployed, Azure will display a list of required input fields.


Home > New > Marketplace > Everything > Template deployment > Enable encryption on a running Windows VM without AAD

Enable encryption on a running Windows VM without AAD

Azure quickstart template

TEMPLATE

201-encrypt-running-windows-vm-without-aad
1 resource

BASICS

* Subscription	Concierge Subscription
* Resource group	f3600d0a-03b1-48d9-85c8-479fda9f9681
* Location	South Central US

SETTINGS

* Vm Name	
* Key Vault Name	
* Key Vault Resource Group	
Key Encryption Key URL	
Volume Type	All
Force Update Tag	1.0
Resize OS Disk	false
Location	[resourceGroup().location]

You can then execute the template to create, modify, or remove resources.

Running templates in the Azure portal

If you already know the template you want to use, or you have saved templates in your Azure account, you can use the **Create a resource > Template Deployment** resource to locate and run defined templates in the portal. You can search through templates by name, edit a template to change the parameters or behavior, and execute the template right from the GUI.

Running templates from the command line

Given a URL to a template, you can execute it with Azure PowerShell. For example, we could run the disk encryption template with the following PowerShell command:

```
New-AzureRmResourceGroupDeployment `
    -Name encrypt-disk `
    -ResourceGroupName <resource-group-name> `
    -TemplateUri https://raw.githubusercontent.com/azure/azure-quick-
start-templates/master/201-encrypt-running-windows-vm-without-aad/azurede-
ploy.json
```

Or, if you prefer the Azure CLI, with the group deployment create command.

```
azure config mode arm
azure group deployment create <my-resource-group> <my-deployment-name> \
    --template-uri https://raw.githubusercontent.com/azure/azure-quick-
start-templates/master/201-encrypt-running-windo
```

Review questions

Module 1 review questions

Azure VMs

What is an availability set and what are some of their benefits?

> Click to see suggested answer

An availability set is a logical feature used to ensure that a group of related VMs are deployed so that they aren't all subject to a single point of failure and not all upgraded at the same time during a host operating system upgrade in the datacenter. VMs placed in an availability set should perform an identical set of functionalities and have the same software installed.

You can create availability sets through the Azure portal in the disaster recovery section. Also, you can build them using Resource Manager templates, or any of the scripting or API tools. When you place VMs into an availability set, Azure guarantees to spread them across Fault Domains and Update Domains.

Azure Disk Encryption

Can you name and describe the main encryption-based disk protection technologies?

> Click to see suggested answer

Storage Service Encryption: Azure Storage Service Encryption (SSE) is an encryption service built into Azure used to protect data at rest. The Azure storage platform automatically encrypts data before it's stored to several storage services, including Azure Managed Disks. Encryption is enabled by default using 256-bit AES encryption, and is managed by the storage account administrator.

Azure Disk Encryption: Azure Disk Encryption (ADE) is managed by the VM owner. It controls the encryption of Windows and Linux VM-controlled disks, using BitLocker on Windows VMs and DM-Crypt on Linux VMs. BitLocker Drive Encryption is a data protection feature that integrates with the operating system, and addresses the threats of data theft or exposure from lost, stolen, or inappropriately decommissioned computers. Similarly, DM-Crypt encrypts data at rest for Linux before writing to storage.

Implement batch jobs by using Azure Batch Services

Azure Batch overview

Introduction to Azure Batch

Use Azure Batch to run large-scale parallel and high-performance computing (HPC) batch jobs efficiently in Azure. Azure Batch creates and manages a pool of compute nodes (virtual machines), installs the applications you want to run, and schedules jobs to run on the nodes. There is no cluster or job scheduler software to install, manage, or scale. Instead, you use Batch APIs and tools, command-line scripts, or the Azure portal to configure, manage, and monitor your jobs.

Developers can use Batch as a platform service to build SaaS applications or client apps where large-scale execution is required. For example, build a service with Batch to run a Monte Carlo risk simulation for a financial services company, or a service to process many images.

There is no additional charge for using Batch. You only pay for the underlying resources consumed, such as the virtual machines, storage, and networking.

Run parallel workloads

Batch works well with intrinsically parallel (also known as "embarrassingly parallel") workloads. Intrinsically parallel workloads are those where the applications can run independently, and each instance completes part of the work. When the applications are executing, they might access some common data, but they do not communicate with other instances of the application. Intrinsically parallel workloads can therefore run at a large scale, determined by the amount of compute resources available to run applications simultaneously.

You can also use Batch to run tightly coupled workloads; these are workloads where the applications you run need to communicate with each other, as opposed to run independently. Tightly coupled applications normally use the Message Passing Interface (MPI) API. You can run your tightly coupled workloads with Batch using Microsoft MPI or Intel MPI. Improve application performance with specialized HPC and GPU-optimized VM sizes.

Many tightly coupled jobs can be run in parallel using Batch. For example, perform multiple simulations of a liquid flowing through a pipe with varying pipe widths.

Additional Batch capabilities

Higher-level, workload-specific capabilities are also available for Azure Batch:

- Batch supports large-scale rendering workloads with rendering tools including Autodesk Maya, 3ds Max, Arnold, and V-Ray.
- R users can install the doAzureParallel R package to easily scale out the execution of R algorithms on Batch pools.

You can also run Batch jobs as part of a larger Azure workflow to transform data, managed by tools such as Azure Data Factory.

Azure Batch Service REST API

The REST APIs for the Azure Batch service offer developers a means to schedule large-scale parallel and HPC applications in the cloud.

Azure Batch REST APIs can be accessed from within a service running in Azure, or directly over the Internet from any application that can send an HTTPS request and HTTPS response.

Batch account

All access to the Batch service requires a Batch account, and the account is the basis for authentication. The Base URL for Batch service is `https://{account-name}.{region-id}.batch.azure.com`.

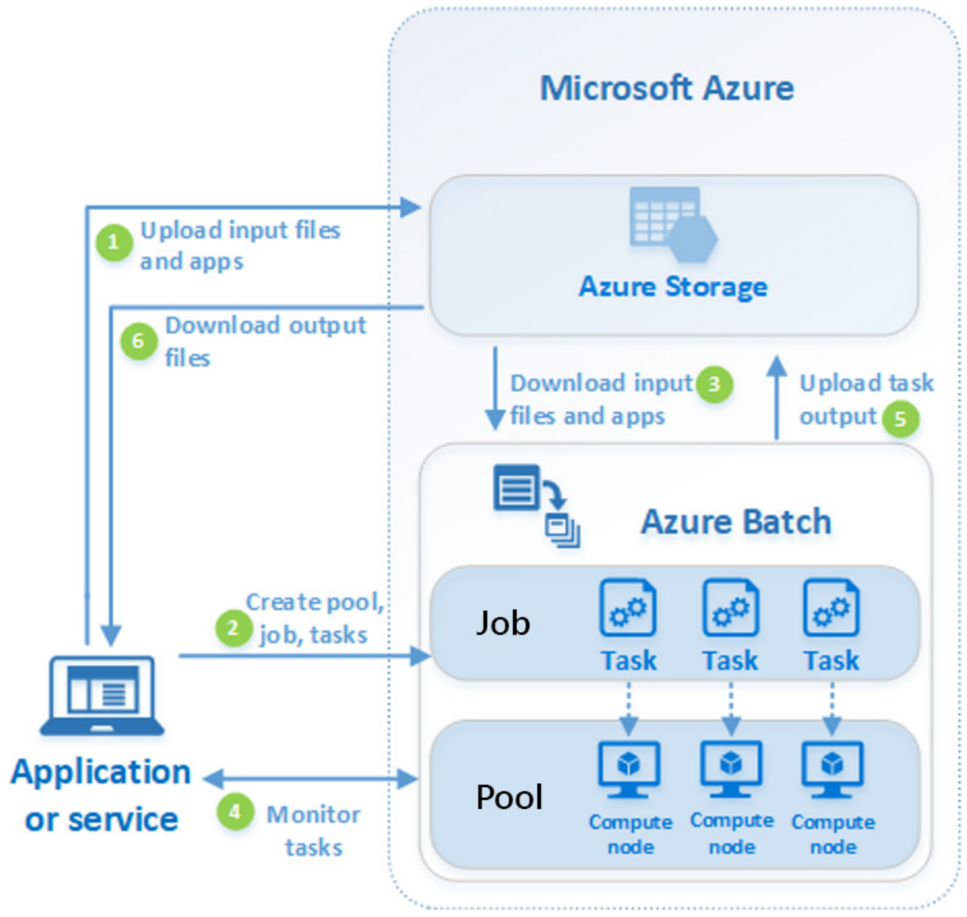
REST APIs

Use these APIs to schedule and run large scale computational workloads. All operations conform to the HTTP/1.1 protocol specification and each operation returns a request-id header that can be used to obtain information about the request. You must make sure that requests made to these resources are secure.

How Azure Batch works

A common scenario for Batch involves scaling out intrinsically parallel work, such as the rendering of images for 3D scenes, on a pool of compute nodes. This pool of compute nodes can be your “render farm” that provides tens, hundreds, or even thousands of cores to your rendering job.

The following diagram shows steps in a common Batch workflow, with a client application or hosted service using Batch to run a parallel workload.



Step	Description
1. Upload input files and the applications to process those files to your Azure Storage account.	The input files can be any data that your application processes, such as financial modeling data, or video files to be transcoded. The application files can include scripts or applications that process the data, such as a media transcoder.
2. Create a Batch pool of compute nodes in your Batch account, a job to run the workload on the pool, and tasks in the job.	Pool nodes are the VMs that execute your tasks. Specify properties such as the number and size of the nodes, a Windows or Linux VM image, and an application to install when the nodes join the pool. Manage the cost and size of the pool by using low-priority VMs or automatically scaling the number of nodes as the workload changes. When you add tasks to a job, the Batch service automatically schedules the tasks for execution on the compute nodes in the pool. Each task uses the application that you uploaded to process the input files.

3. Download input files and the applications to Batch	Before each task executes, it can download the input data that it is to process to the assigned compute node. If the application isn't already installed on the pool nodes, it can be downloaded here instead. When the downloads from Azure Storage complete, the task executes on the assigned node.
4. Monitor task execution	As the tasks run, query Batch to monitor the progress of the job and its tasks. Your client application or service communicates with the Batch service over HTTPS. Because you may be monitoring thousands of tasks running on thousands of compute nodes, be sure to query the Batch service efficiently.
5. Upload task output	As the tasks complete, they can upload their result data to Azure Storage. You can also retrieve files directly from the file system on a compute node.
6. Download output files	When your monitoring detects that the tasks in your job have completed, your client application or service can download the output data for further processing.

Keep in mind this is just one way to use Batch, and this scenario describes just some of its features. For example, you can execute multiple tasks in parallel on each compute node. Or, use job preparation and completion tasks to prepare the nodes for your jobs, then clean up afterward.

Azure Batch service resources

Some of the following resources–accounts, compute nodes, pools, jobs, and tasks–are required by all solutions that use the Batch service. Others, like application packages, are helpful, but optional, features.

Account

A Batch account is a uniquely identified entity within the Batch service. All processing is associated with a Batch account.

You can create an Azure Batch account using the Azure portal or programmatically, such as with the Batch Management .NET library. When creating the account, you can associate an Azure storage account for storing job-related input and output data or applications.

You can run multiple Batch workloads in a single Batch account, or distribute your workloads among Batch accounts that are in the same subscription, but in different Azure regions.

When creating a Batch account, you can choose between two pool allocation modes: user subscription and Batch service. For most cases, you should use the default Batch service mode, in which pools are allocated behind the scenes in Azure-managed subscriptions. In the alternative user subscription mode, Batch VMs and other resources are created directly in your subscription when a pool is created. User subscription mode is required if you want to create Batch pools using Azure Reserved VM Instances. To create a Batch account in user subscription mode, you must also register your subscription with Azure Batch, and associate the account with an Azure Key Vault.

Azure Storage account

Most Batch solutions use Azure Storage for storing resource files and output files. For example, your Batch tasks (including standard tasks, start tasks, job preparation tasks, and job release tasks) typically specify resource files that reside in a storage account.

Batch supports the following Azure Storage account options:

- General-purpose v2 (GPv2) accounts
- General-purpose v1 (GPv1) accounts
- Blob storage accounts (currently supported for pools in the Virtual Machine configuration)

You can associate a storage account with your Batch account when you create the Batch account, or later. Consider your cost and performance requirements when choosing a storage account. For example, the GPv2 and blob storage account options support greater capacity and scalability limits compared with GPv1. These account options can improve the performance of Batch solutions that contain a large number of parallel tasks that read from or write to the storage account.

Compute node

A compute node is an Azure virtual machine (VM) or cloud service VM that is dedicated to processing a portion of your application's workload. The size of a node determines the number of CPU cores, memory capacity, and local file system size that is allocated to the node. You can create pools of Windows or Linux nodes by using Azure Cloud Services, images from the Azure Virtual Machines Marketplace, or custom images that you prepare. See the following Pool section for more information on these options.

Nodes can run any executable or script that is supported by the operating system environment of the node. This includes *.exe, *.cmd, *.bat and PowerShell scripts for Windows–and binaries, shell, and Python scripts for Linux.

Pool

A pool is a collection of nodes that your application runs on. The pool can be created manually by you, or automatically by the Batch service when you specify the work to be done. You can create and manage a pool that meets the resource requirements of your application. A pool can be used only by the Batch account in which it was created. A Batch account can have more than one pool.

Azure Batch pools build on top of the core Azure compute platform. They provide large-scale allocation, application installation, data distribution, health monitoring, and flexible adjustment of the number of compute nodes within a pool (scaling).

Every node that is added to a pool is assigned a unique name and IP address. When a node is removed from a pool, any changes that are made to the operating system or files are lost, and its name and IP address are released for future use. When a node leaves a pool, its lifetime is over.

Compute node type and target number of nodes

When you create a pool, you can specify which types of compute nodes you want and the target number for each. The two types of compute nodes are:

- **Dedicated compute nodes.** Dedicated compute nodes are reserved for your workloads. They are more expensive than low-priority nodes, but they are guaranteed to never be preempted.
- **Low-priority compute nodes.** Low-priority nodes take advantage of surplus capacity in Azure to run your Batch workloads. Low-priority nodes are less expensive per hour than dedicated nodes, and

enable workloads requiring a lot of compute power. For more information, see Use low-priority VMs with Batch.

- Low-priority compute nodes may be preempted when Azure has insufficient surplus capacity. If a node is preempted while running tasks, the tasks are requeued and run again once a compute node becomes available again. Low-priority nodes are a good option for workloads where the job completion time is flexible and the work is distributed across many nodes. Before you decide to use low-priority nodes for your scenario, make sure that any work lost due to preemption will be minimal and easy to recreate.

You can have both low-priority and dedicated compute nodes in the same pool. Each type of node — low-priority and dedicated — has its own target setting, for which you can specify the desired number of nodes.

The number of compute nodes is referred to as a target because, in some situations, your pool might not reach the desired number of nodes. For example, a pool might not achieve the target if it reaches the core quota for your Batch account first. Or, the pool might not achieve the target if you have applied an auto-scaling formula to the pool that limits the maximum number of nodes.

Job

A job is a collection of tasks. It manages how computation is performed by its tasks on the compute nodes in a pool.

The job specifies the pool in which the work is to be run. You can create a new pool for each job, or use one pool for many jobs. You can create a pool for each job that is associated with a job schedule, or for all jobs that are associated with a job schedule.

You can specify an optional job priority. When a job is submitted with a higher priority than jobs that are currently in progress, the tasks for the higher-priority job are inserted into the queue ahead of tasks for the lower-priority jobs. Tasks in lower-priority jobs that are already running are not preempted.

Batch can detect and then retry failed tasks. You can specify the maximum number of task retries as a constraint, including whether a task is always or never retried. Retrying a task means that the task is requeued to be run again.
Your client application can add tasks to a job, or you can specify a job manager task. A job manager task contains the information that is necessary to create the required tasks for a job, with the job manager task being run on one of the compute nodes in the pool. The job manager task is handled specifically by Batch–it is queued as soon as the job is created, and is restarted if it fails. A job manager task is required for jobs that are created by a job schedule because it is the only way to define the tasks before the job is instantiated.

By default, jobs remain in the active state when all tasks within the job are complete. You can change this behavior so that the job is automatically terminated when all tasks in the job are complete. Set the job's onAllTasksComplete property (OnAllTasksComplete in Batch .NET) to terminatejob to automatically terminate the job when all of its tasks are in the completed state.

Application packages

The application packages feature provides easy management and deployment of applications to the compute nodes in your pools. You can upload and manage multiple versions of the applications run by your tasks, including their binaries and support files. Then you can automatically deploy one or more of these applications to the compute nodes in your pool.

You can specify application packages at the pool and task level. When you specify pool application packages, the application is deployed to every node in the pool. When you specify task application packages, the application is deployed only to nodes that are scheduled to run at least one of the job's tasks, just before the task's command line is run.

Batch handles the details of working with Azure Storage to store your application packages and deploy them to compute nodes, so both your code and management overhead can be simplified.

Azure Batch service quotas and limits

As with other Azure services, there are limits on certain resources associated with the Batch service. Many of these limits are default quotas applied by Azure at the subscription or account level. This article discusses those defaults, and how you can request quota increases.

Keep these quotas in mind as you design and scale up your Batch workloads. For example, if your pool doesn't reach the target number of compute nodes you specified, you might have reached the core quota limit for your Batch account.

You can run multiple Batch workloads in a single Batch account, or distribute your workloads among Batch accounts that are in the same subscription, but in different Azure regions.

Resource quotas

Resource	Default Limit	Maximum Limit
Batch accounts per region per subscription	1 - 3	50
Dedicated cores per Batch account	10 - 100	N/A
Low-priority cores per Batch account	10 - 100	N/A
Active jobs and job schedules per Batch account	100 - 300	1000
Pools per Batch account	20 - 100	500

Cores quotas in user subscription mode

If you created a Batch account with pool allocation mode set to user subscription, quotas are applied differently. In this mode, Batch VMs and other resources are created directly in your subscription when a pool is created. The Azure Batch cores quotas do not apply to an account created in this mode. Instead, the quotas in your subscription for regional compute cores and other resources are applied.

Run a batch job by using Azure CLI and Azure portal

Running Batch jobs with Azure CLI

The Azure CLI is used to create and manage Azure resources from the command line or in scripts. This section of the course shows how to use the Azure CLI to create a Batch account, a *pool* of compute nodes (virtual machines), and a *job* that runs tasks on the pool. Each sample task runs a basic command on one of the pool nodes. After completing this, you will understand the key concepts of the Batch service and be ready to try Batch with more realistic workloads at larger scale.

If you don't have an Azure subscription, create a free account before you begin.

Create a resource group

Create a resource group with the `az group create` command. An Azure resource group is a logical container into which Azure resources are deployed and managed.

The following example creates a resource group named myResourceGroup in the eastus2 location.

```
az group create \
    --name myResourceGroup \
    --location eastus2
```

Create a storage account

You can link an Azure Storage account with your Batch account. Although not required for this quickstart, the storage account is useful to deploy applications and store input and output data for most real-world workloads. Create a storage account in your resource group with the `az storage account` create command.

```
az storage account create \
    --resource-group myResourceGroup \
    --name mystorageaccount \
    --location eastus2 \
    --sku Standard_LRS
```

Create a Batch account

Create a Batch account with the `az batch account create` command. You need an account to create compute resources (pools of compute nodes) and Batch jobs.

The following example creates a Batch account named *mybatchaccount* in *myResourceGroup*, and links the storage account you created.

```
az batch account create \
    --name mybatchaccount \
    --storage-account mystorageaccount \
    --resource-group myResourceGroup \
    --location eastus2
```

To create and manage compute pools and jobs, you need to authenticate with Batch. Log in to the account with the `az batch account login` command. After you log in, your `az batch` commands use this account context.

```
az batch account login \
    --name mybatchaccount \
    --resource-group myResourceGroup \
    --shared-key-auth
```

Create a pool of compute nodes

Now that you have a Batch account, create a sample pool of Linux compute nodes using the az batch pool create command. The following example creates a pool named *mypool* of 2 size `Standard_A1_v2` nodes running Ubuntu 16.04 LTS. The suggested node size offers a good balance of performance versus cost for this example.

```
az batch pool create \
    --id mypool --vm-size Standard_A1_v2 \
    --target-dedicated-nodes 2 \
    --image canonical:ubuntuserver:16.04-LTS \
    --node-agent-sku-id "batch.node.ubuntu 16.04"
```

Batch creates the pool immediately, but it takes a few minutes to allocate and start the compute nodes. During this time, the pool is in the `resizing` state. To see the status of the pool, run the `az batch pool show` command. This command shows all the properties of the pool, and you can query for specific properties. The following command gets the allocation state of the pool:

```
az batch pool show --pool-id mypool \
    --query "allocationState"
```

Continue the following steps to create a job and tasks while the pool state is changing. The pool is ready to run tasks when the allocation state is `steady` and all the nodes are running.

Create a job

Now that you have a pool, create a job to run on it. A Batch job is a logical group for one or more tasks. A job includes settings common to the tasks, such as priority and the pool to run tasks on. Create a Batch job by using the `az batch job create` command. The following example creates a job *myjob* on the pool *mypool*. Initially the job has no tasks.

```
az batch job create \
    --id myjob \
    --pool-id mypool
```

Create tasks

Now use the `az batch task create` command to create some tasks to run in the job. In this example, you create four identical tasks. Each task runs a `command-line` to display the Batch environment variables on a compute node, and then waits 90 seconds. When you use Batch, this command line is where you specify your app or script. Batch provides a number of ways to deploy apps and scripts to compute nodes.

The following Bash script creates 4 parallel tasks (*mytask1* to *mytask4*).

```
for i in {1..4}
do
   az batch task create \
    --task-id mytask$i \
    --job-id myjob \
    --command-line "/bin/bash -c 'printenv | grep AZ_BATCH; sleep 90s'"
done
```

The command output shows settings for each of the tasks. Batch distributes the tasks to the compute nodes.

View task status

After you create a task, Batch queues it to run on the pool. Once a node is available to run it, the task runs.

Use the `az batch task show` command to view the status of the Batch tasks. The following example shows details about mytask1 running on one of the pool nodes.

```
az batch task show \
    --job-id myjob \
    --task-id mytask1
```

The command output includes many details, but take note of the `exitCode` of the task command line and the `nodeId`. An `exitCode` of 0 indicates that the task command line completed successfully. The `nodeId` indicates the ID of the pool node on which the task ran.

View task output

To list the files created by a task on a compute node, use the `az batch task file list` command. The following command lists the files created by *mytask1*:

```
az batch task file list \
    --job-id myjob \
    --task-id mytask1 \
    --output table
```

To download one of the output files to a local directory, use the `az batch task file download` command. In this example, task output is in *stdout.txt*.

```
az batch task file download \
    --job-id myjob \
    --task-id mytask1 \
    --file-path stdout.txt \
    --destination ./stdout.txt
```

You can view the contents of `stdout.txt` in a text editor. The contents show the Azure Batch environment variables that are set on the node. When you create your own Batch jobs, you can reference these environment variables in task command lines, and in the apps and scripts run by the command lines.

Running Batch jobs with Azure Portal

This section of the course shows you how to use the Azure Portal to create and run a Batch job. It follows the same pattern as that shown in the previous Azure CLI example.

Create a Batch account

Follow these steps to create a sample Batch account for test purposes. You need a Batch account to create pools and jobs. As shown here, you can link an Azure storage account with the Batch account. Although not required for this quickstart, the storage account is useful to deploy applications and store input and output data for most real-world workloads.

1. In the Azure Portal select **Create a resource > Compute > Batch Service**.
2. Enter values for **Account** name and **Resource** group. The account name must be unique within the Azure **Location** selected, use only lowercase characters or numbers, and contain 3-24 characters.
3. In **Storage account**, select an existing storage account or create a new one.
4. Keep the defaults for remaining settings, and select **Create** to create the account.

When the Deployment succeeded message appears, go to the Batch account in the portal.

Create a pool of compute nodes

Now that you have a Batch account, create a sample pool of Windows compute nodes for test purposes. The pool for this quick example consists of 2 nodes running a Windows Server 2012 R2 image from the Azure Marketplace.

1. In the Batch account, select **Pools > Add**.
2. Enter a **Pool ID** called *mypool*.
3. In **Operating System**, select the following settings (you can explore other options).

Setting	Value
Image Type	Marketplace (Linux/Windows)
Publisher	MicrosoftWindowsServer
Offer	WindowsServer
SKU	2012-R2-Datacenter-smalldisk

4. Scroll down to enter **Node Size** and **Scale** settings. The suggested node size offers a good balance of performance versus cost for this quick example.

Setting	Value
Node pricing tier	Standard_A1
Target dedicated nodes	2

5. Keep the defaults for remaining settings, and select **OK** to create the pool.

Batch creates the pool immediately, but it takes a few minutes to allocate and start the compute nodes. During this time, the pool's **Allocation state** is **Resizing**. You can go ahead and create a job and tasks while the pool is resizing.

After a few minutes, the state of the pool is **Steady**, and the nodes start. Select **Nodes** to check the state of the nodes. When a node's state is **Idle**, it is ready to run tasks.

Create a job

Now that you have a pool, create a job to run on it. A Batch job is a logical group for one or more tasks. A job includes settings common to the tasks, such as priority and the pool to run tasks on. Initially the job has no tasks.

1. In the Batch account view, select **Jobs > Add**.
2. Enter a **Job ID** called *myjob*. In **Pool**, select *mypool*. Keep the defaults for the remaining settings, and select **OK**.

After the job is created, the **Tasks** page opens.

Create tasks

Now create sample tasks to run in the job. Typically you create multiple tasks that Batch queues and distributes to run on the compute nodes. In this example, you create two identical tasks. Each task runs a command line to display the Batch environment variables on a compute node, and then waits 90 seconds.

When you use Batch, the command line is where you specify your app or script. Batch provides a number of ways to deploy apps and scripts to compute nodes.

To create the first task:

1. Select **Add**.
2. Enter a **Task ID** called *mytask*.
3. In **Command line**, enter `cmd /c "set AZ_BATCH & timeout /t 90 > NUL"`. Keep the defaults for the remaining settings, and select **OK**.

After you create a task, Batch queues it to run on the pool. When a node is available to run it, the task runs.

To create a second task, go back to step 1. Enter a different **Task ID**, but specify an identical command line. If the first task is still running, Batch starts the second task on the other node in the pool.

View task output

The preceding task examples complete in a couple of minutes. To view the output of a completed task, select **Files on node**, and then select the file `stdout.txt`. This file shows the standard output of the task.

The contents show the Azure Batch environment variables that are set on the node. When you create your own Batch jobs and tasks, you can reference these environment variables in task command lines, and in the apps and scripts run by the command lines.

Running Batch jobs by using code

Introduction to the Batch Management client library

You can lower maintenance overhead in your Azure Batch applications by using the Batch Management .NET library to automate Batch account creation, deletion, key management, and quota discovery.

- **Create and delete Batch accounts** within any region. If, as an independent software vendor (ISV) for example, you provide a service for your clients in which each is assigned a separate Batch account for billing purposes, you can add account creation and deletion capabilities to your customer portal.
- **Retrieve and regenerate account keys** programmatically for any of your Batch accounts. This can help you comply with security policies that enforce periodic rollover or expiry of account keys. When you have several Batch accounts in various Azure regions, automation of this rollover process increases your solution's efficiency.
- **Check account quotas** and take the trial-and-error guesswork out of determining which Batch accounts have what limits. By checking your account quotas before starting jobs, creating pools, or adding compute nodes, you can proactively adjust where or when these compute resources are created. You can determine which accounts require quota increases before allocating additional resources in those accounts.
- **Combine features of other Azure services** for a full-featured management experience–by using Batch Management .NET, Azure Active Directory, and the Azure Resource Manager together in the same application. By using these features and their APIs, you can provide a frictionless authentication experience, the ability to create and delete resource groups, and the capabilities that are described above for an end-to-end management solution.

Create and delete Batch accounts

As mentioned, one of the primary features of the Batch Management API is to create and delete Batch accounts in an Azure region. To do so, use BatchManagementClient.Account.CreateAsync and DeleteAsync, or their synchronous counterparts.

The following code snippet creates an account, obtains the newly created account from the Batch service, and then deletes it. In this snippet and the others in this section of the course, `batchManagementClient` is a fully initialized instance of BatchManagementClient.

```
// Create a new Batch account
await batchManagementClient.Account.CreateAsync("MyResourceGroup",
    "mynewaccount",
    new BatchAccountCreateParameters() { Location = "West US" });

// Get the new account from the Batch service
AccountResource account = await batchManagementClient.Account.GetAsync(
    "MyResourceGroup",
    "mynewaccount");

// Delete the account
await batchManagementClient.Account.DeleteAsync("MyResourceGroup", account.
Name);
```

Applications that use the Batch Management .NET library and its BatchManagementClient class require **service administrator** or **coadministrator** access to the subscription that owns the Batch account to be managed.

Retrieve and regenerate account keys

Obtain primary and secondary account keys from any Batch account within your subscription by using `ListKeysAsync`. You can regenerate those keys by using `RegenerateKeyAsync`.

```
// Get and print the primary and secondary keys
BatchAccountListKeyResult accountKeys =
    await batchManagementClient.Account.ListKeysAsync(
        "MyResourceGroup",
        "mybatchaccount");
Console.WriteLine("Primary key:   {0}", accountKeys.Primary);
Console.WriteLine("Secondary key: {0}", accountKeys.Secondary);

// Regenerate the primary key
BatchAccountRegenerateKeyResponse newKeys =
    await batchManagementClient.Account.RegenerateKeyAsync(
        "MyResourceGroup",
        "mybatchaccount",
        new BatchAccountRegenerateKeyParameters() {
            KeyName = AccountKeyType.Primary
            });
```

You can create a streamlined connection workflow for your management applications. First, obtain an account key for the Batch account you wish to manage with `ListKeysAsync`. Then, use this key when initializing the Batch .NET library's `BatchSharedKeyCredentials` class, which is used when initializing BatchClient.

Checking Azure and Batch account quotas

Azure subscriptions and the individual Azure services like Batch all have default quotas that limit the number of certain entities within them. For the default quotas for Azure subscriptions, see **Azure subscription and service limits**[1], quotas, and constraints. For the default quotas of the Batch service, see **Quotas and limits for the Azure Batch service**[2]. By using the Batch Management .NET library, you can check these quotas in your applications. This enables you to make allocation decisions before you add accounts or compute resources like pools and compute nodes.

Check an Azure subscription for Batch account quotas

Before creating a Batch account in a region, you can check your Azure subscription to see whether you are able to add an account in that region.

In the code snippet below, we first use `BatchManagementClient.Account.ListAsync` to get a collection of all Batch accounts that are within a subscription. Once we've obtained this collection, we determine how many accounts are in the target region. Then we use `BatchManagementClient.`

1 https://docs.microsoft.com/en-us/azure/azure-subscription-service-limits
2 https://docs.microsoft.com/en-us/azure/batch/batch-quota-limit

`Subscriptions` to obtain the Batch account quota and determine how many accounts (if any) can be created in that region.

```
// Get a collection of all Batch accounts within the subscription
BatchAccountListResponse listResponse =
        await batchManagementClient.Account.ListAsync(new AccountListParam-
eters());
IList<AccountResource> accounts = listResponse.Accounts;
Console.WriteLine("Total number of Batch accounts under subscription id
{0}:  {1}",
    creds.SubscriptionId,
    accounts.Count);

// Get a count of all accounts within the target region
string region = "westus";
int accountsInRegion = accounts.Count(o => o.Location == region);

// Get the account quota for the specified region
SubscriptionQuotasGetResponse quotaResponse = await batchManagementClient.
Subscriptions.GetSubscriptionQuotasAsync(region);
Console.WriteLine("Account quota for {0} region: {1}", region, quotaRe-
sponse.AccountQuota);

// Determine how many accounts can be created in the target region
Console.WriteLine("Accounts in {0}: {1}", region, accountsInRegion);
Console.WriteLine("You can create {0} accounts in the {1} region.", quota-
Response.AccountQuota - accountsInRegion, region);
```

In the snippet above, `creds` is an instance of `TokenCloudCredentials`.

Check a Batch account for compute resource quotas

Before increasing compute resources in your Batch solution, you can check to ensure the resources you want to allocate won't exceed the account's quotas. In the code snippet below, we print the quota information for the Batch account named `mybatchaccount`. In your own application, you could use such information to determine whether the account can handle the additional resources to be created.

```
// First obtain the Batch account
BatchAccountGetResponse getResponse =
    await batchManagementClient.Account.GetAsync("MyResourceGroup", "my-
batchaccount");
AccountResource account = getResponse.Resource;

// Now print the compute resource quotas for the account
Console.WriteLine("Core quota: {0}", account.Properties.CoreQuota);
Console.WriteLine("Pool quota: {0}", account.Properties.PoolQuota);
Console.WriteLine("Active job and job schedule quota: {0}", account.Proper-
ties.ActiveJobAndJobScheduleQuota);
```

While there are default quotas for Azure subscriptions and services, many of these limits can be raised by issuing a request in the Azure portal.

Manage batch jobs by using Batch Service API

Authenticating requests

Every request made against the Batch service must be authenticated. The Batch service supports authentication either via Shared Key or Azure Active Directory (Azure AD).

Authentication via Shared Key

An authenticated request requires two headers: the *Date_ or _ocp-date* header and the *Authorization* header. The following sections describe how to construct these headers.

Specify the date header

All authenticated requests must include the Coordinated Universal Time (UTC) timestamp for the request. You can specify the timestamp either in the *ocp-date* header, or in the standard HTTP/HTTPS *Date* header. If both headers are specified for the request, the value of ocp-date is used as the creation time of the request.

The Batch service must receive a request within 15 minutes of when it is created. By doing this, the service is guarded against security attacks, such as replay attacks. The *ocp-date* header is provided because some HTTP client libraries and proxies automatically set the *Date* header, and do not give you an opportunity to read its value in order to include it in the authenticated request. If you set *ocp-date*, construct the signature with an empty value for the *Date* header.

Specify the authorization header

An authenticated request must include the *Authorization* header. To authenticate a request, you must sign the request with the key for the account that is making the request and pass that signature as part of the request.

The format for the *Authorization* header is as follows:

```
Authorization="SharedKey <AccountName>:<Signature>"
```

`SharedKey` is the name of the authorization scheme, `AccountName` is the name of the account requesting the resource, and `Signature` is a Hash-based Message Authentication Code (HMAC) constructed from the request, computed by using the SHA256 algorithm, and then encoded by using Base64 encoding.

The following sections describe how to construct the *Authorization* header.
Construct the signature string

When you construct the signature string, keep in mind the following:

- The VERB portion of the string is the HTTP verb, such as GET or POST, and must be uppercase.
- Each header included in the signature string may appear only once.
- The values of all standard HTTP headers must be included in the string in the order shown in the signature format, without the header names. These headers may be empty if they are not being specified as part of the request; in that case, only the new line character is required.

- When the verb is POST, the Content-Type and Content-Length values are required as request headers and as values in the signature string. Content-Type must be set to **application/json;odata=minimal-metadata**.
- If the *ocp-date* header is specified, the *Date* header is not required, simply specify an empty line for the *Date* portion of the signature string.
- All new line characters (\n) shown are required within the signature string.

To encode the signature string for a request against the Batch service, use the following format:

```
StringToSign = VERB + "\n" +
  Content-Encoding + "\n"
  Content-Language + "\n"
  Content-Length + "\n"
  Content-MD5 + "\n"
  Content-Type + "\n" +
  Date + "\n" +
  If-Modified-Since + "\n"
  If-Match + "\n"
  If-None-Match + "\n"
  If-Unmodified-Since + "\n"
  Range + "\n"
  CanonicalizedHeaders +
  CanonicalizedResource;
```

The following example shows a signature string for a request to list the jobs in an account with a timeout of 20 seconds. When a header value does not exist, only the new line character is specified.

```
GET\n\n\n\n\n\n\n\n\n\n\n\nocp-date:Tue, 29 Jul 2014 21:49:13 GMT\n /
myaccount/jobs\napi-version:2014-01-01.1.0\ntimeout:20
```

Breaking this down line-by-line shows each portion of the same string:

```
GET\n /*HTTP Verb*/
\n    /*Content-Encoding*/
\n    /*Content-Language*/
\n    /*Content-Length*/
\n    /*Content-MD5*/
\n    /*Content-Type*/
\n    /*Date*/
\n    /*If-Modified-Since */
\n    /*If-Match */
\n    /*If-None-Match */
\n    /*If-Unmodified-Since*/
\n    /* Range */
ocp-date:Tue, 29 Jul 2014 21:49:13 GMT\n    /*CanonicalizedHeaders*/
/myaccount/jobs\napi-version:2014-04-01.1.0\ntimeout:20    /*Canonicalize-
dResource*/
```

Next, encode this string by using the HMAC-SHA256 algorithm over the UTF-8-encoded signature string, construct the *Authorization* header, and add the header to the request. The following example shows the *Authorization* header for the same operation:

```
Authorization: SharedKey myaccount:ctzMq410TV3wS7upTBcunJTDLEJwMAZuFPfr0mr-
rA08=
```

Construct the canonicalized headers string

To construct the `CanonicalizedHeaders` portion of the signature string, follow these steps:

1. Retrieve all headers for the resource that begin with ocp-, including the ocp-date header.
2. Convert each HTTP header name to lowercase.
3. Sort the headers lexicographically by header name, in ascending order. Each header may appear only once in the string.
4. Replace any breaking white space with a single space.
5. Trim any white space around the colon in the header.
6. Append a new line character to each canonicalized header in the resulting list. Construct the `CanonicalizedHeaders` string by concatenating all headers in this list into a single string.

Construct the canonicalized resource string

The `CanonicalizedResource` part of the signature string represents the resource of the Batch service that is targeted by the request. Any portion of the `CanonicalizedResource` string that is derived from the URI of the resource should be encoded exactly as it is in the URI.

Keep in mind the following rules for constructing the canonicalized resource string:

- Avoid using the new line character (\n) in values for query parameters. If it must be used, ensure that it does not affect the format of the canonicalized resource string.
- Avoid using commas in query parameter values.

You can construct the `CanonicalizedResource` string as follows:

1. Beginning with a slash ("/"), followed by the name of the account that owns the resource being accessed.
2. Append the resource's encoded URI path, without any query parameters.
3. Retrieve all query parameters on the resource URI, including the api-version parameter.
4. Convert all parameter names to lowercase.
5. Sort the query parameters lexicographically by parameter name, in ascending order.
6. URL-decode each query parameter name and value.
7. Append each query parameter name and value to the string in the following format, making sure to include the colon (:) between the name and the value:

   ```
   parameter-name:parameter-value
   ```

8. If a query parameter has more than one value, sort all values lexicographically, then include them in a comma-separated list:

   ```
   parameter-name:parameter-value-1,parameter-value-2,parameter-value-n
   ```

9. Append a new line character (\n) after each name-value pair.

Encode the Signature

To encode the signature, call the HMAC-SHA256 algorithm on the UTF-8-encoded signature string and encode the result as Base64. Use the following format (shown as pseudocode):

```
Signature=Base64(HMAC-SHA256(UTF8(StringToSign)))
```

Authentication via Azure AD

Azure AD is Microsoft's multi-tenant cloud based directory and identity management service. The Batch service supports authentication with Azure AD.

Note: Authentication with Azure AD is required only if your Batch account is set up to allocate pools in a user subscription. The pool allocation option is available when you create a new Batch account. If your account is set up to allocate pools in a subscription managed by Batch, then using Azure AD for authentication is optional. For more information, see Batch – VNet and Custom Image Support for Virtual Machine Pools.

To use Azure AD with the Batch service, you'll need the following endpoints.

The Azure AD endpoint "common" endpoint is:

`https://login.microsoftonline.com/common`

The resource endpoint for the Batch service is:

`https://batch.core.windows.net/`

Batch Status and Error Codes

REST API operations for the Batch service return standard HTTP status codes, as defined in the HTTP/1.1 Status Code Definitions.

API operations may also return additional error information to provide the developer with more information about the error. For example, the following error response indicates that a query parameter specified on the request URI was invalid, and provides additional information about the invalid parameter's name and value and the reason for the error.

```
{
  "code": "InvalidQueryParameterValue",
  "message": {
      "lang": "en-us",
      "value": "Value for one of the query parameters specified in the
request URI is invalid"
  },
  "values": [{
      "key": "QueryParameterName",
      "value": "state"
  }, {
      "key": "QueryParameterValue",
      "value": "deleted"
  }, {
      "key": "Reason",
      "value": "invalid state"
  }]
}
```

Job/Task Scheduling Error Codes

If the Batch service encounters an error when starting a task on a node, it marks the task as completed. The error information is returned within a `failureInfo` element in the response body of List the files associated with a task and Get information about a task APIs.

Similarly, if the Batch service encounters an error while starting the job, it marks the job as completed. This scheduling error information is returned within a `schedulingError` element in the response body of Get information about a job in Batch APIs.

The following table provides the list of categories for task scheduling errors.

Category	Description
UserError	Errors in the task specification provided by the user.
ServerError	Errors encountered by the Batch service that prevent it from scheduling the task.

Below is a sample pre-processing error returned by the Batch service.

```
{
  "preProcessingError": {
    "category": "UserError",
    "code": "BlobNotFound",
    "message": "The specified blob does not exist.",
    "values": {
      "name": "FilePath",
      "value": "myfile.txt"
    }
  }
}
```

Common parameters and headers

The following information is common to all operations that you might do related to Batch resources:

- Specify the optional *$select* value as a list of properties to return for a resource.
- Specify the optional *$skiptoken* value in the URI if a partial result is returned in a previous operation call. If the response contains an *odata.nextLink* element, the value of the *odata.nextLink* element includes a *$skiptoken* parameter with a value that specifies the starting point in the collection of entities for the next GET operation. The *$skiptoken* parameter must only be used on the URI contained in the value of the *odata.nextLink* element.
- Specify the optional *maxresults* value in the URI to define the number of items to return in a response. The maximum number of items varies based on the resource:
  - A maximum of 100 certificates can be returned.
  - A maximum of 1000 pools can be returned.
  - A maximum of 1000 nodes can be returned.

- A maximum of 1000 job schedules can be returned.
- A maximum of 1000 jobs can be returned.
- A maximum of 1000 tasks can be returned.

- Set the *Authorization* header to a string that contains the authentication scheme, the account name, and the authentication signature.

Representation of Date/Time Values

Specifying Date/Time values in HTTP headers

The Batch service follows RFC 1123 for representation of date/time values in headers. This is the preferred format for HTTP/1.1 operations, as described in section 3.3 of the **HTTP/1.1 Protocol Parameters**[3] specification. An example of this format is:

```
Sun, 06 Nov 1994 08:49:37 GMT
```

The following format is also supported, as described in the HTTP/1.1 protocol specification:

```
Sunday, 06-Nov-94 08:49:37 GMT
```

Both are represented in the Coordinated Universal Time (UTC), also known as Greenwich Mean Time.

Specifying Date/Time values in URI Parameters and Request/ Response Body

Date/time values in query parameters and request/response body are expressed as UTC times and must adhere to a valid ISO 8601 format. Supported ISO 8601 formats include the following:

- `YYYY-MM-DD`
- `YYYY-MM-DDThh:mmTZD`
- `YYYY-MM-DDThh:mm:ssTZD`

For the date portion of these formats, `YYYY` is a four-digit year representation, `MM` is a two-digit month representation, and `DD` is a two-digit day representation. For the time portion, `hh` is the hour representation in 24-hour notation, `mm` is the two-digit minute representation, and `ss` is the two-digit second representation. A time designator `T` separates the date and time portions of the string, while a time zone designator `TZD` specifies a time zone.

Specifying conditional headers

The Batch service follows the HTTP/1.1 protocol specification for conditional headers, and uses the below rules to process requests specifying conditional headers:

- If a request specifies both the `If-None-Match` and `If-Modified-Since` headers, the request is evaluated based on the criteria specified in `If-None-Match`.
- If a request specifies both the `If-Match` and `If-Unmodified-Since` headers, the request is evaluated based on the criteria specified in `If-Match`.

3 http://go.microsoft.com/fwlink/?linkid=133333

- With the exception of the two combinations of conditional headers listed above, a request may specify only a single conditional header. Specifying more than one conditional header results in status code 400 (Bad Request).

To see which conditional headers are supported by an operation, see the documentation for that operation (for example, **Job Enable**[4]). Not all operations support all headers.

HTTP Response Codes for Operations Supporting Conditional Headers

Batch service returns an HTTP response code in accordance with the HTTP/1.1 protocol specification (RFC 2616).

Read Operations

The following table indicates the response codes returned for an unmet condition for each conditional header when the operation is a read operation. Read operations use the verbs GET or HEAD.

Conditional header	Response code if condition has not been met
`If-Match`	Precondition Failed (412)
`If-None-Match`	Not Modified (304)
`If-Modified-Since`	Not Modified (304)
`If-Unmodified-Since`	Precondition Failed (412)

Write Operations

The following table indicates the response codes returned for an unmet condition for each conditional header when the operation is a write operation. Write operations use the verbs POST, PUT, PATCH or DELETE.

Conditional header	Response code if condition has not been met
`If-Match`	Precondition Failed (412)
`If-None-Match`	Precondition Failed (412)
`If-Modified-Since`	Precondition Failed (412)
`If-Unmodified-Since`	Precondition Failed (412)

Batch Service REST API Versioning

Operations provided by the Batch service REST API may have multiple versions for backwards compatibility as the API evolves over time. You must specify which version of an operation you wish to use when it is called by providing the version with your REST call. If your application calls an older version of an operation, you can choose to continue calling the older version, or modify your code to call a newer version. If the version is not specified or an incorrect version is specified, then an error is returned.

To specify which version of an operation to use, set the *api-version* query parameter. The version is of the format Group.Major.Minor where Group is in the format 'YYYY-MM-DD' and Major is an integer and Minor is an integer.

4 https://docs.microsoft.com/en-us/rest/api/batchservice/job/enable

Review questions

Module 2 review questions

Azure Batch service resources

Most Batch solutions use Azure Storage for storing resource files and output files. For example, your Batch tasks (including standard tasks, start tasks, job preparation tasks, and job release tasks) typically specify resource files that reside in a storage account.

What types of Azure Storage accounts will Azure Batch support?

> Click to see suggested answer

Batch supports the following Azure Storage account options:

- General-purpose v2 (GPv2) accounts
- General-purpose v1 (GPv1) accounts
- Blob storage accounts (currently supported for pools in the Virtual Machine configuration)

You can associate a storage account with your Batch account when you create the Batch account, or later. Consider your cost and performance requirements when choosing a storage account. For example, the GPv2 and blob storage account options support greater capacity and scalability limits compared with GPv1. These account options can improve the performance of Batch solutions that contain a large number of parallel tasks that read from or write to the storage account.

Coding for Batch accounts

Azure subscriptions and the individual Azure services like Batch all have default quotas that limit the number of certain entities within them. You want to create a new Batch account in a region and you want to check your Azure subscription to see whether you are able to add an account in that region.

How would you code for that?

> Click to see suggested answer

In the code snippet below, we first use BatchManagementClient.Account.ListAsync to get a collection of all Batch accounts that are within a subscription. Once we've obtained this collection, we determine how many accounts are in the target region. Then we use BatchManagementClient.Subscriptions to obtain the Batch account quota and determine how many accounts (if any) can be created in that region.

```
// Get a collection of all Batch accounts within the subscription
BatchAccountListResponse listResponse =
        await batchManagementClient.Account.ListAsync(new AccountListParam-
eters());
IList<AccountResource> accounts = listResponse.Accounts;
Console.WriteLine("Total number of Batch accounts under subscription id
{0}:  {1}",
    creds.SubscriptionId,
    accounts.Count);
```

```
// Get a count of all accounts within the target region
string region = "westus";
int accountsInRegion = accounts.Count(o => o.Location == region);

// Get the account quota for the specified region
SubscriptionQuotasGetResponse quotaResponse = await batchManagementClient.
Subscriptions.GetSubscriptionQuotasAsync(region);
Console.WriteLine("Account quota for {0} region: {1}", region, quotaRe-
sponse.AccountQuota);

// Determine how many accounts can be created in the target region
Console.WriteLine("Accounts in {0}: {1}", region, accountsInRegion);
Console.WriteLine("You can create {0} accounts in the {1} region.", quota-
Response.AccountQuota - a
```

Create containerized solutions

Azure Kubernetes Service (AKS) core concepts

What is Kubernetes?

Kubernetes is a rapidly evolving platform that manages container-based applications and their associated networking and storage components. The focus is on the application workloads, not the underlying infrastructure components. Kubernetes provides a declarative approach to deployments, backed by a robust set of APIs for management operations.

You can build and run modern, portable, microservices-based applications that benefit from Kubernetes orchestrating and managing the availability of those application components. Kubernetes supports both stateless and stateful applications as teams progress through the adoption of microservices-based applications.

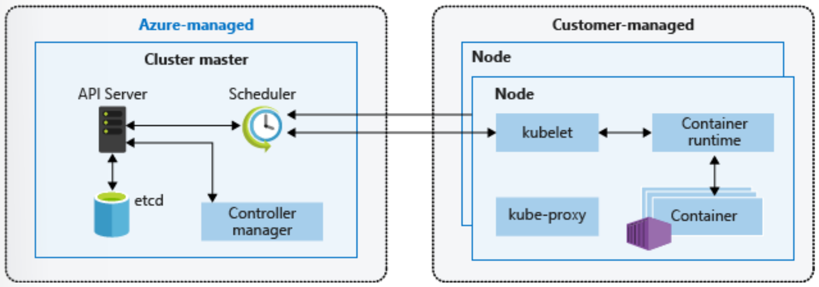
As an open platform, Kubernetes allows you to build your applications with your preferred programming language, OS, libraries, or messaging bus. Existing continuous integration and continuous delivery (CI/CD) tools can integrate with Kubernetes to schedule and deploy releases.

Azure Kubernetes Service (AKS) provides a managed Kubernetes service that reduces the complexity for deployment and core management tasks, including coordinating upgrades. The AKS cluster masters are managed by the Azure platform, and you only pay for the AKS nodes that run your applications. AKS is built on top of the open-source Azure Container Service Engine (acs-engine).

Kubernetes cluster architecture

A Kubernetes cluster is divided into two components:

- *Cluster master* nodes provide the core Kubernetes services and orchestration of application workloads.
- *Nodes* run your application workloads.



Cluster master

When you create an AKS cluster, a cluster master is automatically created and configured. This cluster master is provided as a managed Azure resource abstracted from the user. There is no cost for the cluster master, only the nodes that are part of the AKS cluster.

The cluster master includes the following core Kubernetes components:

- **kube-apiserver** - The API server is how the underlying Kubernetes APIs are exposed. This component provides the interaction for management tools, such as `kubectl` or the Kubernetes dashboard.
- **etcd** - To maintain the state of your Kubernetes cluster and configuration, the highly available *etcd* is a key value store within Kubernetes.
- **kube-scheduler** - When you create or scale applications, the Scheduler determines what nodes can run the workload and starts them.
- **kube-controller-manager** - The Controller Manager oversees a number of smaller Controllers that perform actions such as replicating pods and handling node operations.

AKS provides a single-tenant cluster master, with a dedicated API server, Scheduler, etc. You define the number and size of the nodes, and the Azure platform configures the secure communication between the cluster master and nodes. Interaction with the cluster master occurs through Kubernetes APIs, such as `kubectl` or the Kubernetes dashboard.

This managed cluster master means that you do not need to configure components like a highly available etcd store, but it also means that you cannot access the cluster master directly. Upgrades to Kubernetes are orchestrated through the Azure CLI or Azure portal, which upgrades the cluster master and then the nodes. To troubleshoot possible issues, you can review the cluster master logs through Azure Log Analytics.

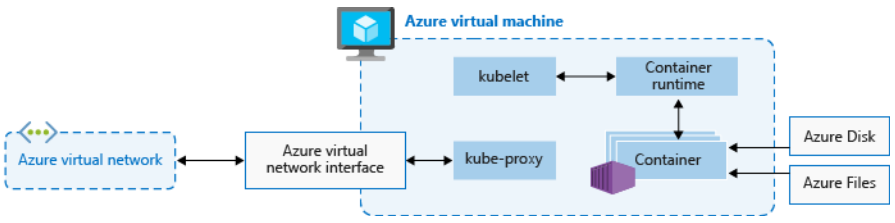
If you need to configure the cluster master in a particular way or need direct access to them, you can deploy your own Kubernetes cluster using `aks-engine`.

Nodes and node pools

To run your applications and supporting services, you need a Kubernetes node. An AKS cluster has one or more nodes, which is an Azure virtual machine (VM) that runs the Kubernetes node components and container runtime:

- The `kubelet` is the Kubernetes agent that processes the orchestration requests from the cluster master and scheduling of running the requested containers.
- Virtual networking is handled by the *kube-proxy* on each node. The proxy routes network traffic and manages IP addressing for services and pods.

- The *container runtime* is the component that allows containerized applications to run and interact with additional resources such as the virtual network and storage. In AKS, Docker is used as the container runtime.



The Azure VM size for your nodes defines how many CPUs, how much memory, and the size and type of storage available (such as high-performance SSD or regular HDD). If you anticipate a need for applications that require large amounts of CPU and memory or high-performance storage, plan the node size accordingly. You can also scale up the number of nodes in your AKS cluster to meet demand.

In AKS, the VM image for the nodes in your cluster is currently based on Ubuntu Linux. When you create an AKS cluster or scale up the number of nodes, the Azure platform creates the requested number of VMs and configures them. There is no manual configuration for you to perform.

If you need to use a different host OS, container runtime, or include custom packages, you can deploy your own Kubernetes cluster using `aks-engine`. The upstream `aks-engine` releases features and provide configuration options before they are officially supported in AKS clusters. For example, if you wish to use Windows containers or a container runtime other than Docker, you can use `aks-engine` to configure and deploy a Kubernetes cluster that meets your current needs.

Resource reservations

You don't need to manage the core Kubernetes components on each node, such as the *kubelet*, *kube-proxy*, and *kube-dns*, but they do consume some of the available compute resources. To maintain node performance and functionality, the following compute resources are reserved on each node:

- **CPU** - 60ms
- **Memory** - 20% up to 4 GiB

These reservations mean that the amount of available CPU and memory for your applications may appear less than the node itself contains. If there are resource constraints due to the number of applications that you run, these reservations ensure CPU and memory remains available for the core Kubernetes components. The resource reservations cannot be changed.

For example:

- **Standard DS2 v2** node size contains 2 vCPU and 7 GiB memory
  - 20% of 7 GiB memory = 1.4 GiB
  - A total of (7 - 1.4) = 5.6 GiB memory is available for the node
- **Standard E4s v3** node size contains 4 vCPU and 32 GiB memory
  - 20% of 32 GiB memory = 6.4 GiB, but AKS only reserves a maximum of 4 GiB
  - A total of (32 - 4) = 28 GiB is available for the node

The underlying node OS also requires some amount of CPU and memory resources to complete its own core functions.

Node pools

Nodes of the same configuration are grouped together into *node pools*. A Kubernetes cluster contains one or more node pools. The initial number of nodes and size are defined when you create an AKS cluster, which creates a *default node pool*. This default node pool in AKS contains the underlying VMs that run your agent nodes.

When you scale or upgrade an AKS cluster, the action is performed against the default node pool. For upgrade operations, running containers are scheduled on other nodes in the node pool until all the nodes are successfully upgraded.

Pods

Kubernetes uses *pods* to run an instance of your application. A pod represents a single instance of your application. Pods typically have a 1:1 mapping with a container, although there are advanced scenarios where a pod may contain multiple containers. These multi-container pods are scheduled together on the same node, and allow containers to share related resources.

When you create a pod, you can define *resource limits* to request a certain amount of CPU or memory resources. The Kubernetes Scheduler tries to schedule the pods to run on a node with available resources to meet the request. You can also specify maximum resource limits that prevent a given pod from consuming too much compute resource from the underlying node. A best practice is to include resource limits for all pods to help the Kubernetes Scheduler understand what resources are needed and permitted.

A pod is a logical resource, but the container(s) are where the application workloads run. Pods are typically ephemeral, disposable resources, and individually scheduled pods miss some of the high availability and redundancy features Kubernetes provides. Instead, pods are usually deployed and managed by Kubernetes *Controllers*, such as the Deployment Controller.

Deployments and YAML manifests

A deployment represents one or more identical pods, managed by the Kubernetes Deployment Controller. A deployment defines the number of replicas (pods) to create, and the Kubernetes Scheduler ensures that if pods or nodes encounter problems, additional pods are scheduled on healthy nodes.

You can update deployments to change the configuration of pods, container image used, or attached storage. The Deployment Controller drains and terminates a given number of replicas, creates replicas from the new deployment definition, and continues the process until all replicas in the deployment are updated.

Most stateless applications in AKS should use the deployment model rather than scheduling individual pods. Kubernetes can monitor the health and status of deployments to ensure that the required number of replicas run within the cluster. When you only schedule individual pods, the pods are not restarted if they encounter a problem, and are not rescheduled on healthy nodes if their current node encounters a problem.

If an application requires a quorum of instances to always be available for management decisions to be made, you don't want an update process to disrupt that ability. *Pod Disruption Budgets* can be used to define how many replicas in a deployment can be taken down during an update or node upgrade. For example, if you have 5 replicas in your deployment, you can define a pod disruption of 4 to only permit one replica from being deleted/rescheduled at a time. As with pod resource limits, a best practice is to define pod disruption budgets on applications that require a minimum number of replicas to always be present.

Deployments are typically created and managed with `kubectl create` or `kubectl apply`. To create a deployment, you define a manifest file in the YAML (YAML Ain't Markup Language) format. The following example creates a basic deployment of the NGINX web server. The deployment specifies 3 replicas to be created, and that port 80 be open on the container. Resource requests and limits are also defined for CPU and memory.

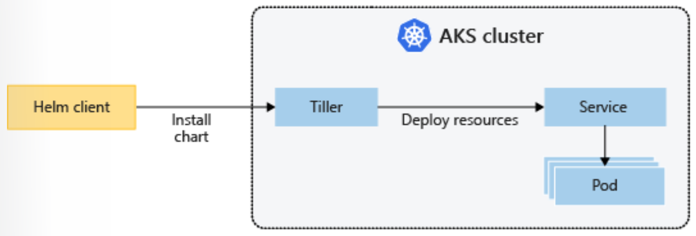
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.15.2
        ports:
        - containerPort: 80
        resources:
          requests:
            cpu: 250m
            memory: 64Mi
          limits:
            cpu: 500m
            memory: 256Mi
```

More complex applications can be created by also including services such as load balancers within the YAML manifest.

Package management with Helm

A common approach to managing applications in Kubernetes is with Helm. You can build and use existing public Helm charts that contain a packaged version of application code and Kubernetes YAML manifests to deploy resources. These Helm charts can be stored locally, or often in a remote repository, such as an Azure Container Registry Helm chart repo.

To use Helm, a server component called *Tiller* is installed in your Kubernetes cluster. The Tiller manages the installation of charts within the cluster. The Helm client itself is installed locally on your computer, or can be used within the Azure Cloud Shell. You can search for or create Helm charts with the client, and then install them to your Kubernetes cluster.



For more information, see **Install applications with Helm in Azure Kubernetes Service (AKS)**[1].

StatefulSets and DaemonSets

The Deployment Controller uses the Kubernetes Scheduler to run a given number of replicas on any available node with available resources. This approach of using deployments may be sufficient for stateless applications, but not for applications that require a persistent naming convention or storage. For applications that require a replica to exist on each node, or selected nodes, within a cluster, the Deployment Controller doesn't look at how replicas are distributed across the nodes.

There are two Kubernetes resources that let you manage these types of applications:

- *StatefulSets* - Maintain the state of applications beyond an individual pod lifecycle, such as storage.
- *DaemonSets* - Ensure a running instance on each node, early in the Kubernetes bootstrap process.

StatefulSets

Modern application development often aims for stateless applications, but *StatefulSets* can be used for stateful applications, such as applications that include database components. A StatefulSet is similar to a deployment in that one or more identical pods are created and managed. Replicas in a StatefulSet follow a graceful, sequential approach to deployment, scale, upgrades, and terminations. With a StatefulSet, the naming convention, network names, and storage persist as replicas are rescheduled.

You define the application in YAML format using `kind: StatefulSet`, and the StatefulSet Controller then handles the deployment and management of the required replicas. Data is written to persistent storage, provided by Azure Managed Disks or Azure Files. With StatefulSets, the underlying persistent storage remains even when the StatefulSet is deleted.

Replicas in a StatefulSet are scheduled and run across any available node in an AKS cluster. If you need to ensure that at least one pod in your Set runs on a node, you can instead use a DaemonSet.

DaemonSets

For specific log collection or monitoring needs, you may need to run a given pod on all, or selected, nodes. A DaemonSet is again used to deploy one or more identical pods, but the DaemonSet Controller ensures that each node specified runs an instance of the pod.

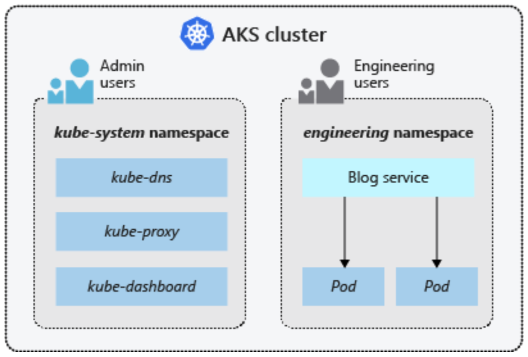
The DaemonSet Controller can schedule pods on nodes early in the cluster boot process, before the default Kubernetes scheduler has started. This ability ensures that the pods in a DaemonSet are started before traditional pods in a Deployment or StatefulSet are scheduled.

Like StatefulSets, a DaemonSet is defined as part of a YAML definition using `kind: DaemonSet`.

1 https://docs.microsoft.com/en-us/azure/aks/kubernetes-helm

Namespaces

Kubernetes resources, such as pods and Deployments, are logically grouped into a *namespace*. These groupings provide a way to logically divide an AKS cluster and restrict access to create, view, or manage resources. You can create namespaces to separate business groups, for example. Users can only interact with resources within their assigned namespaces.



When you create an AKS cluster, the following namespaces are available:

- *default* - This namespace is where pods and deployments are created by default when none is provided. In smaller environments, you can deploy applications directly into the default namespace without creating additional logical separations. When you interact with the Kubernetes API, such as with kubectl get pods, the default namespace is used when none is specified.
- *kube-system* - This namespace is where core resources exist, such as network features like DNS and proxy, or the Kubernetes dashboard. You typically don't deploy your own applications into this namespace.
- *kube-public* - This namespace is typically not used, but can be used for resources to be visible across the whole cluster, and can viewed by any users.

AKS access and identity

There are different ways to authenticate with and secure Kubernetes clusters. Using role-based access controls (RBAC), you can grant users or groups access to only the resources they need. With Azure Kubernetes Service (AKS), you can further enhance the security and permissions structure by using Azure Active Directory. These approaches help you secure your application workloads and customer data.

This section introduces the core concepts that help you authenticate and assign permissions in AKS:

- Kubernetes service accounts
- Azure Active Directory integration
- Role-based access controls (RBAC)
- Roles and ClusterRoles
- RoleBindings and ClusterRoleBindings

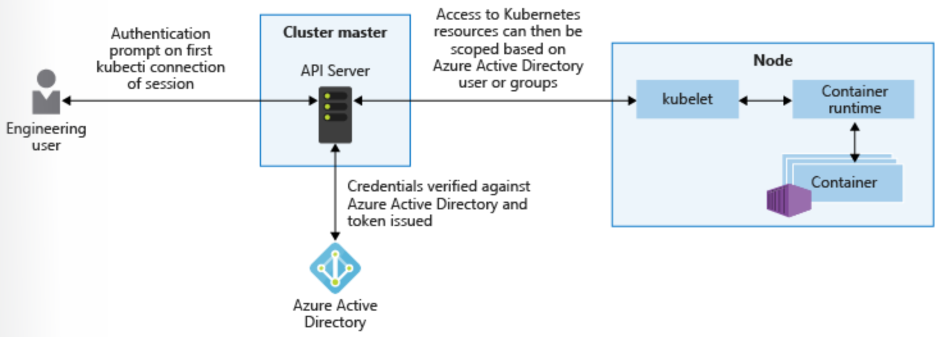
Kubernetes service accounts

One of the primary user types in Kubernetes is a service account. A service account exists in, and is managed by, the Kubernetes API. The credentials for service accounts are stored as Kubernetes secrets, which allows them to be used by authorized pods to communicate with the API Server. Most API requests provide an authentication token for a service account or a normal user account.

Normal user accounts allow more traditional access for human administrators or developers, not just services and processes. Kubernetes itself does not provide an identity management solution where regular user accounts and passwords are stored. Instead, external identity solutions can be integrated into Kubernetes. For AKS clusters, this integrated identity solution is Azure Active Directory.

Azure Active Directory integration

The security of AKS clusters can be enhanced with the integration of Azure Active Directory (AD). Built on decades of enterprise identity management, Azure AD is a multi-tenant, cloud-based directory, and identity management service that combines core directory services, application access management, and identity protection. With Azure AD, you can integrate on-premises identities into AKS clusters to provide a single source for account management and security.


Authentication prompt on first kubecti connection of session
Cluster master
API Server
Access to Kubernetes resources can then be scoped based on Azure Active Directory user or groups
Node
kubelet
Container runtime
Container
Engineering user
Credentials verified against Azure Active Directory and token issued
Azure Active Directory

With Azure AD-integrated AKS clusters, you can grant users or groups access to Kubernetes resources within a namespace or across the cluster. To obtain a `kubectl` configuration context, a user can run the `az aks get-credentials` command. When a user then interacts with the AKS cluster with `kubectl`, they are prompted to sign in with their Azure AD credentials. This approach provides a single source for user account management and password credentials. The user can only access the resources as defined by the cluster administrator.

Azure AD authentication in AKS clusters uses OpenID Connect, an identity layer built on top of the OAuth 2.0 protocol. OAuth 2.0 defines mechanisms to obtain and use access tokens to access protected resources, and OpenID Connect implements authentication as an extension to the OAuth 2.0 authorization process. For more information on OpenID Connect, see the **Open ID Connect documentation**[2]. To verify the authentication tokens obtained from Azure AD through OpenID Connect, AKS clusters use Kubernetes Webhook Token Authentication. For more information, see the **Webhook Token Authentication documentation**[3].

Role-based access controls (RBAC)

To provide granular filtering of the actions that users can perform, Kubernetes uses role-based access controls (RBAC). This control mechanism lets you assign users, or groups of users, permission to do things like create or modify resources, or view logs from running application workloads. These permissions can be scoped to a single namespace, or granted across the entire AKS cluster. With Kubernetes RBAC, you create roles to define permissions, and then assign those *roles* to users with *role bindings*.

2 https://docs.microsoft.com/en-us/azure/active-directory/develop/v1-protocols-openid-connect-code
3 https://kubernetes.io/docs/reference/access-authn-authz/authentication/#webhook-token-authentication

Azure role-based access controls (RBAC)

One additional mechanism for controlling access to resources is Azure role-based access controls (RBAC). Kubernetes RBAC is designed to work on resources within your AKS cluster, and Azure RBAC is designed to work on resources within your Azure subscription. With Azure RBAC, you create a *role definition* that outlines the permissions to be applied. A user or group is then assigned this role definition for a particular *scope*, which could be an individual resource, a resource group, or across the subscription.

Roles and ClusterRoles

Before you assign permissions to users with Kubernetes RBAC, you first define those permissions as a *Role*. Kubernetes roles grant permissions. There is no concept of a *deny* permission.

Roles are used to grant permissions within a namespace. If you need to grant permissions across the entire cluster, or to cluster resources outside a given namespace, you can instead use *ClusterRoles*.

A ClusterRole works in the same way to grant permissions to resources, but can be applied to resources across the entire cluster, not a specific namespace.

RoleBindings and ClusterRoleBindings

Once roles are defined to grant permissions to resources, you assign those Kubernetes RBAC permissions with a *RoleBinding*. If your AKS cluster integrates with Azure Active Directory, bindings are how those Azure AD users are granted permissions to perform actions within the cluster.

Role bindings are used to assign roles for a given namespace. This approach lets you logically segregate a single AKS cluster, with users only able to access the application resources in their assigned namespace. If you need to bind roles across the entire cluster, or to cluster resources outside a given namespace, you can instead use *ClusterRoleBindings*.

A ClusterRoleBinding works in the same way to bind roles to users, but can be applied to resources across the entire cluster, not a specific namespace. This approach lets you grant administrators or support engineers access to all resources in the AKS cluster.

AKS security concepts for apps and clusters

To protect your customer data as you run application workloads in Azure Kubernetes Service (AKS), the security of your cluster is a key consideration. Kubernetes includes security components such as network policies and Secrets. Azure then adds in components such as network security groups and orchestrated cluster upgrades. These security components are combined to keep your AKS cluster running the latest OS security updates and Kubernetes releases, and with secure pod traffic and access to sensitive credentials.

This section introduces the core concepts that secure your applications in AKS:

- Master components security
- Node security
- Cluster upgrades
- Network security
- Kubernetes Secrets

Master security

In AKS, the Kubernetes master components are part of the managed service provided my Microsoft. Each AKS cluster has their own single-tenanted, dedicated Kubernetes master to provide the API Server, Scheduler, etc. This master is managed and maintained by Microsoft

By default, the Kubernetes API server uses a public IP address and with fully qualified domain name (FQDN). You can control access to the API server using Kubernetes role-based access controls and Azure Active Directory.

Node security

AKS nodes are Azure virtual machines that you manage and maintain. The nodes run an optimized Ubuntu Linux distribution with the Docker container runtime. When an AKS cluster is created or scaled up, the nodes are automatically deployed with the latest OS security updates and configurations.

The Azure platform automatically applies OS security patches to the nodes on a nightly basis. If an OS security update requires a host reboot, that reboot is not automatically performed. You can manually reboot the nodes, or a common approach is to use **Kured**[4], an open-source reboot daemon for Kubernetes. Kured runs as a [DaemonSet][aks-daemonset] and monitors each node for the presence of a file indicating that a reboot is required. Reboots are managed across the cluster using the same cordon and drain process as a cluster upgrade.

Nodes are deployed into a private virtual network subnet, with no public IP addresses assigned. For troubleshooting and management purposes, SSH is enabled by default. This SSH access is only available using the internal IP address. Azure network security group rules can be used to further restrict IP range access to the AKS nodes. Deleting the default network security group SSH rule and disabling the SSH service on the nodes prevents the Azure platform from performing maintenance tasks.

To provide storage, the nodes use Azure Managed Disks. For most VM node sizes, these are Premium disks backed by high-performance SSDs. The data stored on managed disks is automatically encrypted at rest within the Azure platform. To improve redundancy, these disks are also securely replicated within the Azure datacenter.

Cluster upgrades

For security and compliance, or to use the latest features, Azure provides tools to orchestrate the upgrade of an AKS cluster and components. This upgrade orchestration includes both the Kubernetes master and agent components. You can view a list of available Kubernetes versions for your AKS cluster. To start the upgrade process, you specify one of these available versions. Azure then safely cordons and drains each AKS node and performs the upgrade.

Cordon and drain

During the upgrade process, AKS nodes are individually cordoned from the cluster so new pods are not scheduled on them. The nodes are then drained and upgraded as follows:

- Existing pods are gracefully terminated and scheduled on remaining nodes.
- The node is rebooted, the upgrade process completed, and then joins back into the AKS cluster.
- Pods are scheduled to run on them again.

4 https://github.com/weaveworks/kured

- The next node in the cluster is cordoned and drained using the same process until all nodes are successfully upgraded.

Network security

For connectivity and security with on-premises networks, you can deploy your AKS cluster into existing Azure virtual network subnets. These virtual networks may have an Azure Site-to-Site VPN or Express Route connection back to your on-premises network. Kubernetes ingress controllers can be defined with private, internal IP addresses so services are only accessible over this internal network connection.

Azure network security groups

To filter the flow of traffic in virtual networks, Azure uses network security group rules. These rules define the source and destination IP ranges, ports, and protocols that are allowed or denied access to resources. Default rules are created to allow TLS traffic to the Kubernetes API server and for SSH access to the nodes. As you create services with load balancers, port mappings, or ingress routes, AKS automatically modifies the network security group for traffic to flow appropriately.

Kubernetes Secrets

A Kubernetes *Secret* is used to inject sensitive data into pods, such as access credentials or keys. You first create a Secret using the Kubernetes API. When you define your pod or deployment, a specific Secret can be requested. Secrets are only provided to nodes that have a scheduled pod that requires it, and the Secret is stored in *tmpfs*, not written to disk. When the last pod on a node that requires a Secret is deleted, the Secret is deleted from the node's tmpfs. Secrets are stored within a given namespace and can only be accessed by pods within the same namespace.

The use of Secrets reduces the sensitive information that is defined in the pod or service YAML manifest. Instead, you request the Secret stored in Kubernetes API Server as part of your YAML manifest. This approach only provides the specific pod access to the Secret.

Network concepts for apps in AKS

In a container-based microservices approach to application development, application components must work together to process their tasks. Kubernetes provides various resources that enable this application communication. You can connect to and expose applications internally or externally. To build highly available applications, you can load balance your applications. More complex applications may require configuration of ingress traffic for SSL/TLS termination or routing of multiple components. For security reasons, you may also need to restrict the flow of network traffic into or between pods and nodes.

This section introduces the core concepts that provide networking to your applications in AKS:

- Services
- Azure virtual networks
- Ingress controllers
- Network policies

Kubernetes basics

To allow access to your applications, or for application components to communicate with each other, Kubernetes provides an abstraction layer to virtual networking. Kubernetes nodes are connected to a virtual network, and can provide inbound and outbound connectivity for pods. The *kube-proxy* component runs on each node to provide these network features.

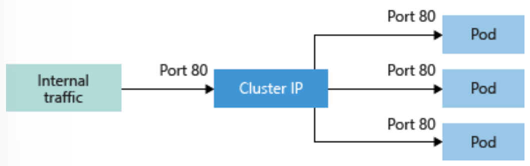
In Kubernetes, Services logically group pods to allow for direct access via an IP address or DNS name and on a specific port. You can also distribute traffic using a load balancer. More complex routing of application traffic can also be achieved with Ingress Controllers. Security and filtering of the network traffic for pods is possible with Kubernetes network policies.

The Azure platform also helps to simplify virtual networking for AKS clusters. When you create a Kubernetes load balancer, the underlying Azure load balancer resource is created and configured. As you open network ports to pods, the corresponding Azure network security group rules are configured. For HTTP application routing, Azure can also configure external DNS as new ingress routes are configured.

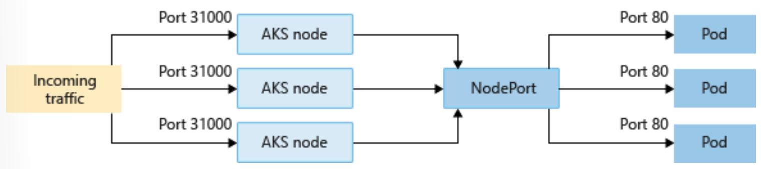
Services

To simplify the network configuration for application workloads, Kubernetes uses Services to logically group a set of pods together and provide network connectivity. The following Service types are available:

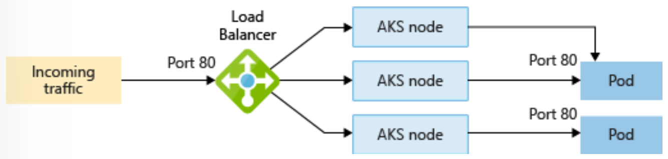
- **Cluster IP** - Creates an internal IP address for use within the AKS cluster. Good for internal-only applications that support other workloads within the cluster.



-
- **NodePort** - Creates a port mapping on the underlying node that allows the application to be accessed directly with the node IP address and port.



-
- **LoadBalancer** - Creates an Azure load balancer resource, configures an external IP address, and connects the requested pods to the load balancer backend pool. To allow customers traffic to reach the application, load balancing rules are created on the desired ports.



-
- For additional control and routing of the inbound traffic, you may instead use an Ingress controller.
- **ExternalName** - Creates a specific DNS entry for easier application access.

The IP address for load balancers and services can be dynamically assigned, or you can specify an existing static IP address to use. Both internal and external static IP addresses can be assigned. This existing static IP address is often tied to a DNS entry.

Both *internal* and *external* load balancers can be created. Internal load balancers are only assigned a private IP address, so can't be accessed from the Internet.

Azure virtual networks

In AKS, you can deploy a cluster that uses one of the following two network models:

- *Basic* networking - The network resources are created and configured as the AKS cluster is deployed.
- *Advanced* networking - The AKS cluster is connected to existing virtual network resources and configurations.

Basic networking

The *basic* networking option is the default configuration for AKS cluster creation. The Azure platform manages the network configuration of the cluster and pods. Basic networking is appropriate for deployments that do not require custom virtual network configuration. With basic networking, you can't define network configuration such as subnet names or the IP address ranges assigned to the AKS cluster.

Nodes in an AKS cluster configured for basic networking use the kubenet Kubernetes plugin.

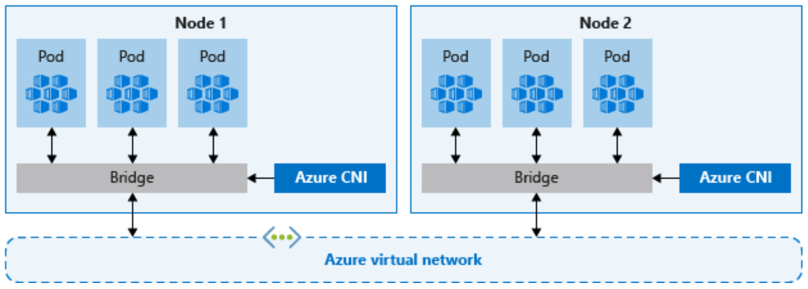
Basic networking provides the following features:

- Expose a Kubernetes service externally or internally through the Azure Load Balancer.
- Pods can access resources on the public Internet.

Advanced networking

Advanced networking places your pods in an Azure virtual network that you configure. This virtual network provides automatic connectivity to other Azure resources and integration with a rich set of capabilities. Advanced networking is appropriate for deployments that require specific virtual network configurations, such as to use an existing subnet and connectivity. With advanced networking, you can define these subnet names and IP address ranges.

Nodes in an AKS cluster configured for advanced networking use the Azure Container Networking Interface (CNI) Kubernetes plugin.



Advanced networking provides the following features over basic networking:

- Deploy your AKS cluster into an existing Azure virtual network, or create a new virtual network and subnet for your cluster.
- Every pod in the cluster is assigned an IP address in the virtual network. The pods can directly communicate with other pods in the cluster, and other nodes in the virtual network.
- A pod can connect to other services in a peered virtual network, including to on-premises networks over ExpressRoute and site-to-site (S2S) VPN connections. Pods are also reachable from on-premises.
- Pods in a subnet that have service endpoints enabled can securely connect to Azure services, such as Azure Storage and SQL DB.

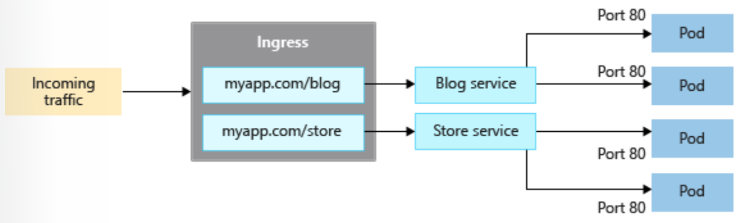
- You can create user-defined routes (UDR) to route traffic from pods to a Network Virtual Appliance.

For more information, see **Configure advanced network for an AKS cluster**[5].

Ingress controllers

When you create a LoadBalancer type Service, an underlying Azure load balancer resource is created. The load balancer is configured to distribute traffic to the pods in your Service on a given port. The LoadBalancer only works at layer 4 - the Service is unaware of the actual applications, and can't make any additional routing considerations.

Ingress controllers work at layer 7, and can use more intelligent rules to distribute application traffic. A common use of an Ingress controller is to route HTTP traffic to different applications based on the inbound URL.



In AKS, you can create an Ingress resource using something like NGINX, or use the AKS HTTP application routing feature. When you enable HTTP application routing for an AKS cluster, the Azure platform creates the Ingress controller and an External-DNS controller. As new Ingress resources are created in Kubernetes, the required DNS A records are created in a cluster-specific DNS zone.

Another common feature of Ingress is SSL/TLS termination. On large web applications accessed via HTTPS, the TLS termination can be handled by the Ingress resource rather than within the application itself. To provide automatic TLS certification generation and configuration, you can configure the Ingress resource to use providers such as Let's Encrypt.

Network security groups

A network security group filter traffic for VMs, such as the AKS nodes. As you create Services, such as a LoadBalancer, the Azure platform automatically configures any network security group rules that are needed. Don't manually configure network security group rules to filter traffic for pods in an AKS cluster. Define any required ports and forwarding as part of your Kubernetes Service manifests, and let the Azure platform create or update the appropriate rules.

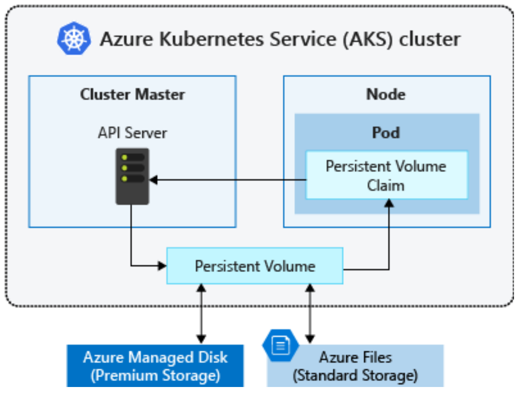
Default network security group rules exist for traffic such as SSH. These default rules are for cluster management and troubleshooting access. Deleting these default rules can cause problems with AKS management, and breaks the service level objective (SLO).

Storage options for apps in AKS

Applications that run in Azure Kubernetes Service (AKS) may need to store and retrieve data. For some application workloads, this data storage can use local, fast storage on the node that is no longer needed when the pods are deleted. Other application workloads may require storage that persists on more regular data volumes within the Azure platform. Multiple pods may need to share the same data volumes,

5 https://docs.microsoft.com/en-us/azure/aks/configure-advanced-networking

or reattach data volumes if the pod is rescheduled on a different node. Finally, you may need to inject sensitive data or application configuration information into pods.


Azure Kubernetes Service (AKS) cluster
Cluster Master
API Server
Node
Pod
Persistent Volume Claim
Persistent Volume
Azure Managed Disk (Premium Storage)
Azure Files (Standard Storage)

This section introduces the core concepts that provide storage to your applications in AKS:

- Volumes
- Persistent volumes
- Storage classes
- Persistent volume claims

Volumes

Applications often need to be able to store and retrieve data. As Kubernetes typically treats individual pods as ephemeral, disposable resources, different approaches are available for applications use and persist data as necessary. A *volume* represents a way to store, retrieve, and persist data across pods and through the application lifecycle.

Traditional volumes to store and retrieve data are created as Kubernetes resources backed by Azure Storage. You can manually create these data volumes to be assigned to pods directly, or have Kubernetes automatically create them. These data volumes can use Azure Disks or Azure Files:

- *Azure Disks* can be used to create a Kubernetes *DataDisk* resource. Disks can use Azure Premium storage, backed by high-performance SSDs, or Azure Standard storage, backed by regular HDDs. For most production and development workloads, use Premium storage. Azure Disks are mounted as *ReadWriteOnce*, so are only available to a single node. For storage volumes that can be accessed by multiple nodes simultaneously, use Azure Files.
- *Azure Files* can be used to mount an SMB 3.0 share backed by an Azure Storage account to pods. Files let you share data across multiple nodes and pods. Currently, Files can only use Azure Standard storage backed by regular HDDs.

In Kubernetes, volumes can represent more than just a traditional disk where information can be stored and retrieved. Kubernetes volumes can also be used as a way to inject data into a pod for use by the containers. Common additional volume types in Kubernetes include:

- *emptyDir* - This volume is commonly used as temporary space for a pod. All containers within a pod can access the data on the volume. Data written to this volume type persists only for the lifespan of the pod - when the pod is deleted, the volume is deleted. This volume typically uses the underlying local node disk storage, though can also exist only in the node's memory.
- *secret* - This volume is used to inject sensitive data into pods, such as passwords. You first create a Secret using the Kubernetes API. When you define your pod or deployment, a specific Secret can be

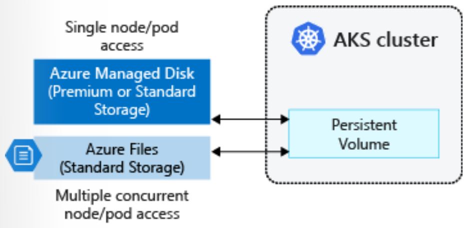
requested. Secrets are only provided to nodes that have a scheduled pod that requires it, and the Secret is stored in tmpfs, not written to disk. When the last pod on a node that requires a Secret is deleted, the Secret is deleted from the node's tmpfs. Secrets are stored within a given namespace and can only be accessed by pods within the same namespace.

- *configMap* - This volume type is used to inject key-value pair properties into pods, such as application configuration information. Rather than defining application configuration information within a container image, you can define it as a Kubernetes resource that can be easily updated and applied to new instances of pods as they are deployed. Like using a Secret, you first create a ConfigMap using the Kubernetes API. This ConfigMap can then be requested when you define a pod or deployment. ConfigMaps are stored within a given namespace and can only be accessed by pods within the same namespace.

Persistent volumes

Volumes are defined and created as part of the pod lifecycle only exist until the pod is deleted. Pods often expect their storage to remain if a pod is rescheduled on a different host during a maintenance event, especially in StatefulSets. A *persistent volume* (PV) is a storage resource created and managed by the Kubernetes API that can exist beyond the lifetime of an individual pod.

Azure Disks or Files are used to provide the PersistentVolume. As noted in the previous section on Volumes, the choice of Disks or Files is often determined by the need for concurrent access to the data or the performance tier.


Single node/pod access
Azure Managed Disk (Premium or Standard Storage)
Azure Files (Standard Storage)
Multiple concurrent node/pod access
AKS cluster
Persistent Volume

A PersistentVolume can be *statically* created by a cluster administrator, or dynamically created by the Kubernetes API server. If a pod is scheduled and requests storage that is not currently available, Kubernetes can create the underlying Azure Disk or Files storage and attach it to the pod. Dynamic provisioning uses a *StorageClass* to identify what type of Azure storage needs to be created.

Storage classes

To define different tiers of storage, such as Premium and Standard, you can create a *StorageClass*. The StorageClass also defines the *reclaimPolicy*. This reclaimPolicy controls the behavior of the underlying Azure storage resource when the pod is deleted and the persistent volume may no longer be required. The underlying storage resource can be deleted, or retained for use with a future pod.

In AKS, two initial StorageClasses are created:

- *default* - Uses Azure Standard storage to create a Managed Disk. The reclaim policy indicates that the underlying Azure Disk is deleted when the pod that used it is deleted.
- *managed-premium* - Uses Azure Premium storage to create Managed Disk. The reclaim policy again indicates that the underlying Azure Disk is deleted when the pod that used it is deleted.

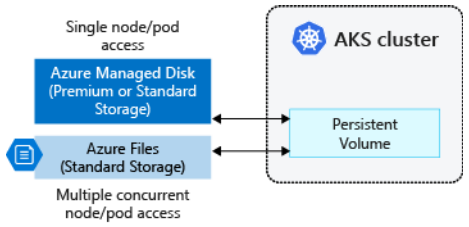
If no StorageClass is specified for a persistent volume, the default StorageClass is used. Take care when requesting persistent volumes so that they use the appropriate storage you need. You can create a

StorageClass for additional needs using `kubectl`. The following example uses Premium Managed Disks and specifies that the underlying Azure Disk should be retained when the pod is deleted:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: managed-premium-retain
provisioner: kubernetes.io/azure-disk
reclaimPolicy: Retain
parameters:
  storageaccounttype: Premium_LRS
  kind: Managed
```

Persistent volume claims

A PersistentVolumeClaim requests either Disk or File storage of a particular StorageClass, access mode, and size. The Kubernetes API server can dynamically provision the underlying storage resource in Azure if there is no existing resource to fulfill the claim based on the defined StorageClass. The pod definition includes the volume mount once the volume has been connected to the pod.


Single node/pod access
Azure Managed Disk (Premium or Standard Storage)
Azure Files (Standard Storage)
Multiple concurrent node/pod access
AKS cluster
Persistent Volume

A PersistentVolume is *bound* to a PersistentVolumeClaim once an available storage resource has been assigned to the pod requesting it. There is a 1:1 mapping of persistent volumes to claims.

The following example YAML manifest shows a persistent volume claim that uses the *managed-premium* StorageClass and requests a Disk *5Gi* in size:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: azure-managed-disk
spec:
  accessModes:
  - ReadWriteOnce
  storageClassName: managed-premium
  resources:
    requests:
      storage: 5Gi
```

When you create a pod definition, the persistent volume claim is specified to request the desired storage. You also then specify the *volumeMount* for your applications to read and write data. The following example YAML manifest shows how the previous persistent volume claim can be used to mount a volume at */mnt/azure*:

```
kind: Pod
apiVersion: v1
```

```
metadata:
  name: nginx
spec:
  containers:
    - name: myfrontend
      image: nginx
      volumeMounts:
      - mountPath: "/mnt/azure"
        name: volume
  volumes:
    - name: volume
      persistentVolumeClaim:
        claimName: azure-managed-disk
```

Scaling options for apps in AKS

As you run applications in Azure Kubernetes Service (AKS), you may need to increase or decrease the amount of compute resources. As the number of application instances you need change, the number of underlying Kubernetes nodes may also need to change. You may also need to quickly provision a large number of additional application instances.

This section introduces the core concepts that help you scale applications in AKS:

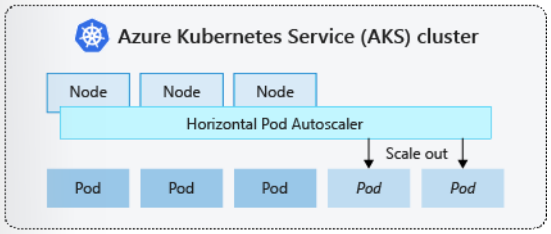
- Manually scale
- Horizontal pod autoscaler (HPA)
- Cluster autoscaler
- Azure Container Instance (ACI) integration with AKS

Manually scale pods or nodes

You can manually scale replicas (pods) and nodes to test how your application responds to a change in available resources and state. Manually scaling resources also lets you define a set amount of resources to use to maintain a fixed cost, such as the number of nodes. To manually scale, you define the replica or node count, and the Kubernetes API schedules creating additional pods or draining nodes.

Horizontal pod autoscaler

Kubernetes uses the horizontal pod autoscaler (HPA) to monitor the resource demand and automatically scale the number of replicas. By default, the horizontal pod autoscaler checks the Metrics API every 30 seconds for any required changes in replica count. When changes are required, the number of replicas is increased or decreased accordingly. Horizontal pod autoscaler works with AKS clusters that have deployed the Metrics Server for Kubernetes 1.8+.


Azure Kubernetes Service (AKS) cluster
Node
Node
Node
Horizontal Pod Autoscaler
Scale out
Pod
Pod
Pod
Pod
Pod

When you configure the horizontal pod autoscaler for a given deployment, you define the minimum and maximum number of replicas that can run. You also define the metric to monitor and base any scaling decisions on, such as CPU usage.

Cooldown of scaling events

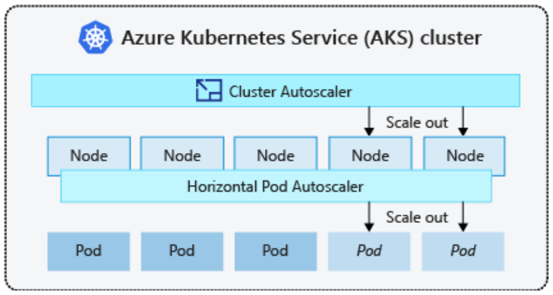
As the horizontal pod autoscaler checks the Metrics API every 30 seconds, previous scale events may not have successfully completed before another check is made. This behavior could cause the horizontal pod autoscaler to change the number of replicas before the previous scale event has been able to receive application workload and the resource demands to adjust accordingly.

To minimize these race events, cooldown or delay values can be set. These values define how long the horizontal pod autoscaler must wait after a scale event before another scale event can be triggered. This behavior allows the new replica count to take effect and the Metrics API reflect the distributed workload. By default, the delay on scale up events is 3 minutes, and the delay on scale down events is 5 minutes

You may need to tune these cooldown values. The default cooldown values may give the impression that the horizontal pod autoscaler isn't scaling the replica count quickly enough. For example, to more quickly increase the number of replicas in use, reduce the `--horizontal-pod-autoscaler-upscale-delay` when you create your horizontal pod autoscaler definitions using `kubectl`.

Cluster autoscaler

To respond to changing pod demands, Kubernetes has a cluster autoscaler that adjusts the number of nodes based on the requested compute resources in the node pool. By default, the cluster autoscaler checks the API server every 10 seconds for any required changes in node count. If the cluster autoscale determines that a change is required, the number of nodes in your AKS cluster is increased or decreased accordingly. The cluster autoscaler works with RBAC-enabled AKS clusters that run Kubernetes 1.10.x or higher.



Cluster autoscaler is typically used alongside the horizontal pod autoscaler. When combined, the horizontal pod autoscaler increases or decreases the number of pods based on application demand, and the cluster autoscaler adjusts the number of nodes as needed to run those additional pods accordingly.

Scale up events

If a node does not have sufficient compute resources to run a requested pod, that pod cannot progress through the scheduling process. The pod cannot start unless additional compute resources are available within the node pool.

When the cluster autoscaler notices pods that cannot be scheduled due to node pool resource constraints, the number of nodes within the node pool is increased to provide the additional compute resources. When those additional nodes are successfully deployed and available for use within the node pool, the pods are then scheduled to run on them.

If your application needs to scale rapidly, some pods may remain in a state waiting to be scheduled until the additional nodes deployed by the cluster autoscaler can accept the scheduled pods. For applications that have high burst demands, you can scale with virtual nodes and Azure Container Instances.

Scale down events

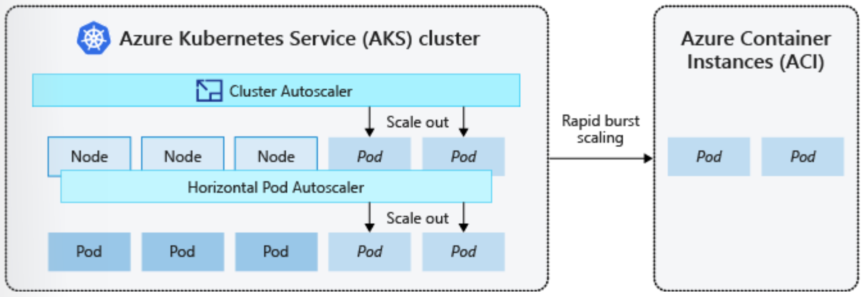
The cluster autoscaler also monitors the pod scheduling status for nodes that have not recently received new scheduling requests. This scenario indicates that the node pool has more compute resources than are required, and that the number of nodes can be decreased.

A node that passes a threshold for no longer being needed for 10 minutes by default is scheduled for deletion. When this situation occurs, pods are scheduled to run on other nodes within the node pool, and the cluster autoscaler decreases the number of nodes.

Your applications may experience some disruption as pods are scheduled on different nodes when the cluster autoscaler decreases the number of nodes. To minimize disruption, avoid applications that use a single pod instance.

Burst to Azure Container Instances

To rapidly scale your AKS cluster, you can integrate with Azure Container Instances (ACI). Kubernetes has built-in components to scale the replica and node count. However, if your application needs to rapidly scale, the horizontal pod autoscaler may schedule more pods than can be provided by the existing compute resources in the node pool. If configured, this scenario would then trigger the cluster autoscaler to deploy additional nodes in the node pool, but it may take a few minutes for those nodes to successfully provision and allow the Kubernetes scheduler to run pods on them.


Azure Kubernetes Service (AKS) cluster
Cluster Autoscaler
Scale out
Node
Node
Node
Pod
Pod
Horizontal Pod Autoscaler
Scale out
Pod
Pod
Pod
Pod
Pod
Rapid burst scaling
Azure Container Instances (ACI)
Pod
Pod

ACI lets you quickly deploy container instances without additional infrastructure overhead. When you connect with AKS, ACI becomes a secured, logical extension of your AKS cluster. The Virtual Kubelet component is installed in your AKS cluster that presents ACI as a virtual Kubernetes node. Kubernetes can then schedule pods that run as ACI instances through virtual nodes, not as pods on VM nodes directly in your AKS cluster.

Your application requires no modification to use virtual nodes. Deployments can scale across AKS and ACI and with no delay as cluster autoscaler deploys new nodes in your AKS cluster.

Virtual nodes are deployed to an additional subnet in the same virtual network as your AKS cluster. This virtual network configuration allows the traffic between ACI and AKS to be secured. Like an AKS cluster, an ACI instance is a secure, logical compute resource that is isolated from other users.

Developer best practices for managing resourc-es in AKS

As you develop and run applications in Azure Kubernetes Service (AKS), there are a few key areas to consider. How you manage your application deployments can negatively impact the end-user experience of services that you provide. To help you succeed, keep in mind some best practices you can follow as you develop and run applications in AKS.

In this section, you learn:

- What are pod resource requests and limits
- Ways to develop and deploy applications with Dev Spaces and Visual Studio Code
- How to use the `kube-advisor` tool to check for issues with deployments

Define pod resource requests and limits

Best practice guidance - Set pod requests and limits on all pods in your YAML manifests. If the AKS cluster uses *resource quotas*, your deployment may be rejected if you don't define these values.

A primary way to manage the compute resources within an AKS cluster is to use pod requests and limits. These requests and limits let the Kubernetes scheduler know what compute resources a pod should be assigned.

- **Pod requests** define a set amount of CPU and memory that the pod needs. These requests should be the amount of compute resources the pod needs to provide an acceptable level of performance.

  - When the Kubernetes scheduler tries to place a pod on a node, the pod requests are used to determine which node has sufficient resources available.
  - Monitor the performance of your application to adjust these requests to make sure you don't define less resources that required to maintain an acceptable level of performance.
- **Pod limits** are the maximum amount of CPU and memory that a pod can use. These limits help prevent one or two runaway pods from taking too much CPU and memory from the node. This scenario would reduce the performance of the node and other pods that run on it.

  - Don't set a pod limit higher than your nodes can support. Each AKS node reserves a set amount of CPU and memory for the core Kubernetes components. Your application may try to consume too many resources on the node for other pods to successfully run.
  - Again, monitor the performance of your application at different times during the day or week. Determine when the peak demand is, and align the pod limits to the resources required to meet the application's needs.

In your pod specifications, it's best practice to define these requests and limits. If you don't include these values, the Kubernetes scheduler doesn't understand what resources are needed. The scheduler may schedule the pod on a node without sufficient resources to provide acceptable application performance. The cluster administrator may set *resource quotas* on a namespace that requires you to set resource requests and limits.

When you define a CPU request or limit, the value is measured in CPU units. *1.0* CPU equates to one underlying virtual CPU core on the node. The same measurement is used for GPUs. You can also define a fractional request or limit, typically in millicpu. For example, *100m* is *0.1* of an underlying virtual CPU core.

In the following basic example for a single NGINX pod, the pod requests *100m* of CPU time, and *128Mi* of memory. The resource limits for the pod are set to *250m* CPU and *256Mi* memory:

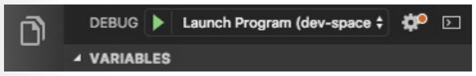
```
kind: Pod
apiVersion: v1
metadata:
  name: mypod
spec:
  containers:
  - name: mypod
    image: nginx:1.15.5
    resources:
      requests:
        cpu: 100m
        memory: 128Mi
      limits:
        cpu: 250m
        memory: 256Mi
```

For more information about resource measurements and assignments, see **Managing compute resources for containers**[6].

Develop and debug applications against an AKS cluster

Best practice guidance - Development teams should deploy and debug against an AKS cluster using Dev Spaces. This development model makes sure that role-based access controls, network, or storage needs are implemented before the app is deployed to production.

With Azure Dev Spaces, you develop, debug, and test applications directly against an AKS cluster. Developers within a team work together to build and test throughout the application lifecycle. You can continue to use existing tools such as Visual Studio or Visual Studio Code. An extension is installed for Dev Spaces that gives an option to run and debug the application in an AKS cluster:



This integrated development and test process with Dev Spaces reduces the need for local test environments, such as `minikube`. Instead, you develop and test against an AKS cluster. This cluster can be secured and isolated as noted in previous section on the use of namespaces to logically isolate a cluster. When your apps are ready to deploy to production, you can confidently deploy as your development was all done against a real AKS cluster.

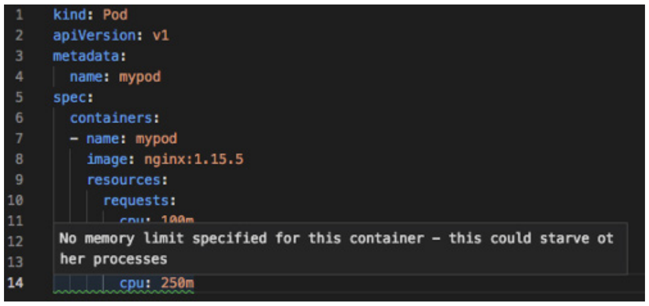
Use the Visual Studio Code extension for Kubernetes

Best practice guidance - Install and use the VS Code extension for Kubernetes when you write YAML manifests. You can also use the extension for integrated deployment solution, which may help application owners that infrequently interact with the AKS cluster.

The **Visual Studio Code extension for Kubernetes**[7] helps you develop and deploy applications to AKS. The extension provides intellisense for Kubernetes resources, and Helm charts and templates. You can

6 https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/
7 https://github.com/Azure/vscode-kubernetes-tools

also browse, deploy, and edit Kubernetes resources from within VS Code. The extension also provides an intellisense check for resource requests or limits being set in the pod specifications:



```yaml
kind: Pod
apiVersion: v1
metadata:
  name: mypod
spec:
  containers:
  - name: mypod
    image: nginx:1.15.5
    resources:
      requests:
        cpu: 100m
No memory limit specified for this container - this could starve ot
her processes
        cpu: 250m
```

Regularly check for application issues with kube-advisor

Best practice guidance - Regularly run the latest version of `kube-advisor` to detect issues in your cluster. If you apply resource quotas on an existing AKS cluster, run `kube-advisor` first to find pods that don't have resource requests and limits defined.

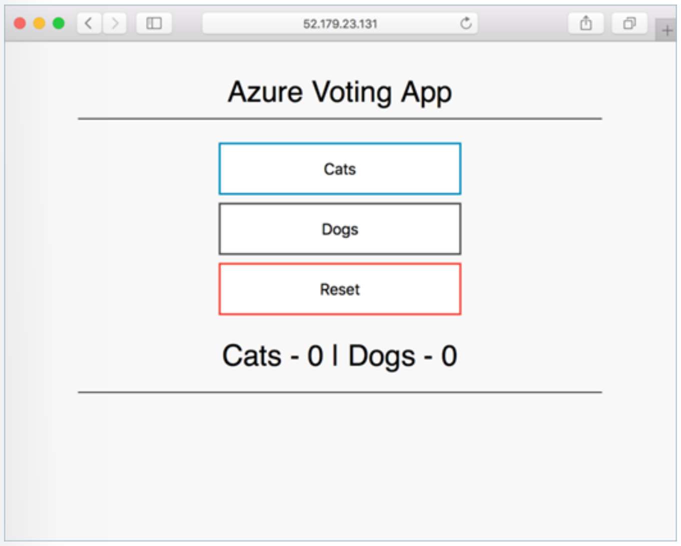
The `kube-advisor` tool scans a Kubernetes cluster and reports on issues that it finds. One useful check is to identify pods that don't have resource requests and limits in place.

In an AKS cluster that hosts many development teams and applications, it can be hard to track pods without these resource requests and limits set. As a best practice, regularly run `kube-advisor` on your AKS clusters.

Deploy an AKS cluster

Deploy an AKS cluster using Azure CLI

In this section of the coruse, an AKS cluster is deployed using the Azure CLI. A multi-container application consisting of web front end and a Redis instance is then run on the cluster. Once completed, the application is accessible over the internet. You'll need a basic understanding of Kubernetes concepts, for detailed information on Kubernetes see the **Kubernetes documentation**[8]. Once completed, the application is accessible over the internet.



Create a resource group

Login to the Azure Portal (https://portal.azure.com) and launch the Azure Cloud Shell.

Create a resource group with the `az group create` command. An Azure resource group is a logical group in which Azure resources are deployed and managed. When you create a resource group, you are asked to specify a location. This location is where your resources run in Azure.

The following example creates a resource group named *myAKSCluster* in the eastus location.

```
az group create --name myAKSCluster --location eastus
```

Output:

```
{
  "id": "/subscriptions/00000000-0000-0000-0000-000000000000/resource-
Groups/myAKSCluster",
  "location": "eastus",
  "managedBy": null,
  "name": "myAKSCluster",
  "properties": {
    "provisioningState": "Succeeded"
  },
```

8 https://kubernetes.io/docs/home/

```
    "tags": null
  }
```

Create AKS cluster

Use the `az aks create` command to create an AKS cluster. The following example creates a cluster named myAKSCluster with one node. Container health monitoring is also enabled using the *–enable-addons monitoring* parameter.

```
az aks create --resource-group myAKSCluster --name myAKSCluster --node-
count 1 --enable-addons monitoring --generate-ssh-keys
```

After several minutes, the command completes and returns JSON-formatted information about the cluster.

Connect to the cluster

To manage a Kubernetes cluster, use `kubectl`, the Kubernetes command-line client.

If you're using Azure Cloud Shell, `kubectl` is already installed. If you want to install it locally, use the `az aks install-cli` command.

To configure `kubectl` to connect to your Kubernetes cluster, use the `az aks get-credentials` command. This step downloads credentials and configures the Kubernetes CLI to use them.

```
az aks get-credentials --resource-group myAKSCluster --name myAKSCluster
```

To verify the connection to your cluster, use the `kubectl get` command to return a list of the cluster nodes. It can take a few minutes for the nodes to appear.

```
kubectl get nodes
```

Output:

```
NAME                          STATUS    ROLES     AGE       VERSION
k8s-myAKSCluster-36346190-0   Ready     agent     2m        v1.7.7
```

Run the application

A Kubernetes manifest file defines a desired state for the cluster, including what container images should be running. For this example, a manifest is used to create all objects needed to run the Azure Vote application. This manifest includes two Kubernetes deployments, one for the Azure Vote Python applications, and the other for a Redis instance. Also, two Kubernetes Services are created, an internal service for the Redis instance, and an external service for accessing the Azure Vote application from the internet.

Create a file named `azure-vote.yaml` and copy into it the following YAML code. If you are working in Azure Cloud Shell, this file can be created using `vi` or `Nano` as if working on a virtual or physical system.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-back
```

```
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-back
  template:
    metadata:
      labels:
        app: azure-vote-back
    spec:
      containers:
      - name: azure-vote-back
        image: redis
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 250m
            memory: 256Mi
        ports:
        - containerPort: 6379
          name: redis
--- #separator
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-back
spec:
  ports:
  - port: 6379
  selector:
    app: azure-vote-back
--- #separator
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-front
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-front
  template:
    metadata:
      labels:
        app: azure-vote-front
    spec:
      containers:
      - name: azure-vote-front
        image: microsoft/azure-vote-front:v1
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 250m
            memory: 256Mi
        ports:
        - containerPort: 80
        env:
        - name: REDIS
          value: "azure-vote-back"
--- #separator
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-front
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: azure-vote-front
```

Use the `kubectl apply` command to run the application.

```
kubectl apply -f azure-vote.yaml
```

Output:

```
deployment "azure-vote-back" created
service "azure-vote-back" created
deployment "azure-vote-front" created
service "azure-vote-front" created
```

Test the application

As the application is run, a Kubernetes service is created that exposes the application front end to the internet. This process can take a few minutes to complete.

To monitor progress, use the `kubectl get service` command with the `--watch` argument.

```
kubectl get service azure-vote-front --watch
```

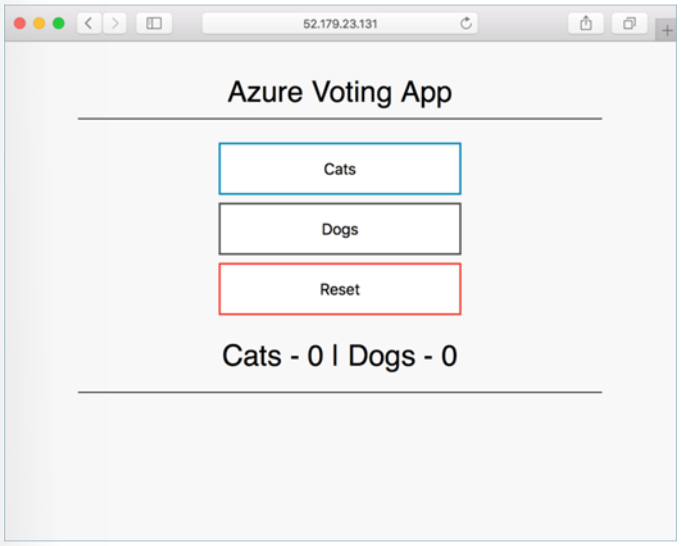
Initially the EXTERNAL-IP for the azure-vote-front service appears as pending.

```
NAME               TYPE           CLUSTER-IP   EXTERNAL-IP   PORT(S)
AGE
azure-vote-front   LoadBalancer   10.0.37.27   <pending>     80:30572/TCP
6s
```

Once the EXTERNAL-IP address has changed from pending to an IP address, use CTRL-C to stop the kubectl watch process.

```
azure-vote-front   LoadBalancer   10.0.37.27   52.179.23.131   80:30572/TCP
2m
```

Now browse to the external IP address to see the Azure Vote App.



Monitor health and logs

Monitor health and logs

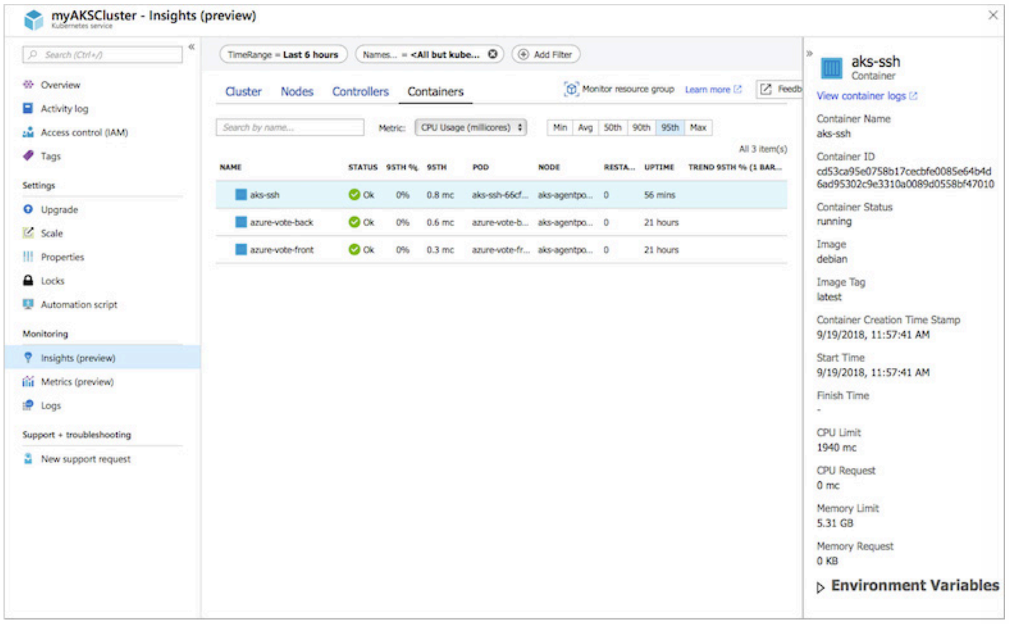
When the AKS cluster was created, monitoring was enabled to capture health metrics for both the cluster nodes and pods. These health metrics are available in the Azure portal. For more information on container health monitoring, see, see **Monitor Azure Kubernetes Service health**[9].

It may take a few minutes for this data to populate in the Azure portal. To see current status, uptime, and resource usage for the Azure Vote pods, complete the following steps:

1. Open a web browser to the Azure portal https://portal.azure.com.
2. Select your resource group, such as *myResourceGroup*, then select your AKS cluster, such as *myAKSCluster*.
3. Under **Monitoring** on the left-hand side, choose **Insights (preview)**
4. Across the top, choose to **+ Add Filter**
5. Select **Namespace** as the property, then choose *<All but kube-system>*
6. Choose to view the **Containers**.

The *azure-vote-back* and *azure-vote-front* containers are displayed, as shown in the following example:

9 https://docs.microsoft.com/en-us/azure/monitoring/monitoring-container-health



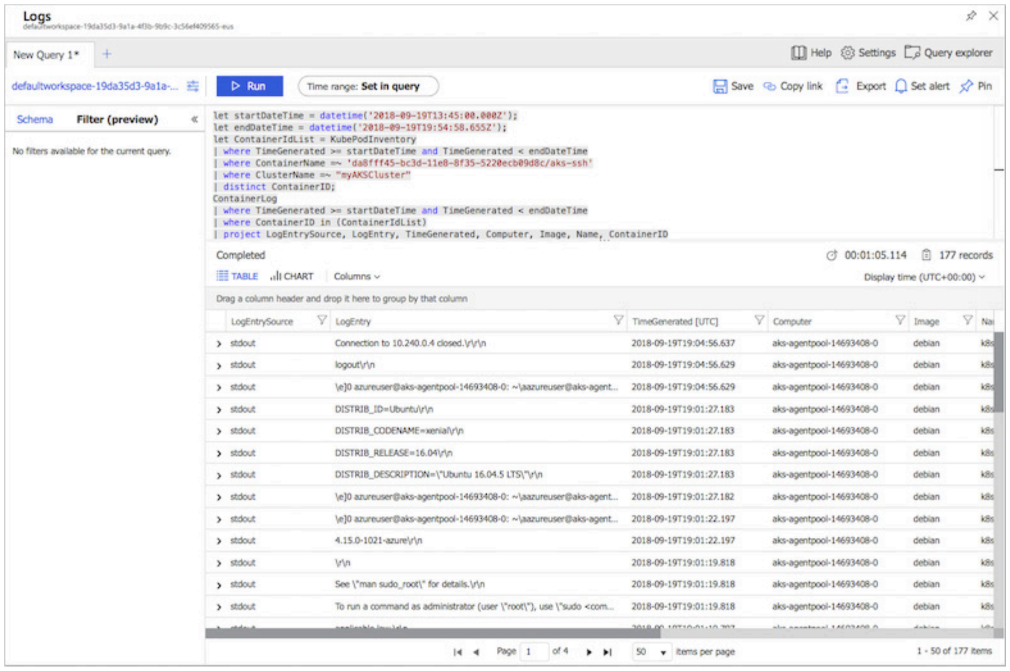
NAME	STATUS	95TH %	95TH	POD	NODE	RESTA...	UPTIME	TREND 95TH % (1 BAR...
aks-ssh	Ok	0%	0.8 mc	aks-ssh-66cf...	aks-agentpo...	0	56 mins	
azure-vote-back	Ok	0%	0.6 mc	azure-vote-b...	aks-agentpo...	0	21 hours	
azure-vote-front	Ok	0%	0.3 mc	azure-vote-fr...	aks-agentpo...	0	21 hours	

aks-ssh
Container
View container logs

Container Name	aks-ssh
Container ID	cd53ca95e0758b17cecbfe0085e64b4d6ad95302c9e3310a0089d0558bf47010
Container Status	running
Image	debian
Image Tag	latest
Container Creation Time Stamp	9/19/2018, 11:57:41 AM
Start Time	9/19/2018, 11:57:41 AM
Finish Time	-
CPU Limit	1940 mc
CPU Request	0 mc
Memory Limit	5.31 GB
Memory Request	0 KB

▷ Environment Variables

To see logs for the `azure-vote-front` pod, select the **View container logs** link on the right-hand side of the containers list. These logs include the *stdout* and *stderr* streams from the container.


Logs
defaultworkspace-19da35d3-9a1a-4f3b-9b9c-3c56ef409565-eus
New Query 1*
Help
Settings
Query explorer
defaultworkspace-19da35d3-9a1a-...
Run
Time range: Set in query
Save
Copy link
Export
Set alert
Pin
Schema
Filter (preview)
No filters available for the current query.

```
let startDateTime = datetime('2018-09-19T13:45:00.000Z');
let endDateTime = datetime('2018-09-19T19:54:58.655Z');
let ContainerIdList = KubePodInventory
| where TimeGenerated >= startDateTime and TimeGenerated < endDateTime
| where ContainerName =~ 'da8fff45-bc3d-11e8-8f35-5220ecb09d8c/aks-ssh'
| where ClusterName =~ "myAKSCluster"
| distinct ContainerID;
ContainerLog
| where TimeGenerated >= startDateTime and TimeGenerated < endDateTime
| where ContainerID in (ContainerIdList)
| project LogEntrySource, LogEntry, TimeGenerated, Computer, Image, Name, ContainerID
```

Completed   00:01:05.114   177 records

TABLE   CHART   Columns   Display time (UTC+00:00)

Drag a column header and drop it here to group by that column

LogEntrySource	LogEntry	TimeGenerated [UTC]	Computer	Image	Na
stdout	Connection to 10.240.0.4 closed.\r\r\n	2018-09-19T19:04:56.637	aks-agentpool-14693408-0	debian	k8s
stdout	logout\r\n	2018-09-19T19:04:56.629	aks-agentpool-14693408-0	debian	k8s
stdout	\e]0 azureuser@aks-agentpool-14693408-0: ~\aazureuser@aks-agent...	2018-09-19T19:04:56.629	aks-agentpool-14693408-0	debian	k8s
stdout	DISTRIB_ID=Ubuntu\r\n	2018-09-19T19:01:27.183	aks-agentpool-14693408-0	debian	k8s
stdout	DISTRIB_CODENAME=xenial\r\n	2018-09-19T19:01:27.183	aks-agentpool-14693408-0	debian	k8s
stdout	DISTRIB_RELEASE=16.04\r\n	2018-09-19T19:01:27.183	aks-agentpool-14693408-0	debian	k8s
stdout	DISTRIB_DESCRIPTION=\"Ubuntu 16.04.5 LTS\"\r\n	2018-09-19T19:01:27.183	aks-agentpool-14693408-0	debian	k8s
stdout	\e]0 azureuser@aks-agentpool-14693408-0: ~\aazureuser@aks-agent...	2018-09-19T19:01:27.182	aks-agentpool-14693408-0	debian	k8s
stdout	\e]0 azureuser@aks-agentpool-14693408-0: ~\aazureuser@aks-agent...	2018-09-19T19:01:22.197	aks-agentpool-14693408-0	debian	k8s
stdout	4.15.0-1021-azure\r\n	2018-09-19T19:01:22.197	aks-agentpool-14693408-0	debian	k8s
stdout	\r\n	2018-09-19T19:01:19.818	aks-agentpool-14693408-0	debian	k8s
stdout	See \"man sudo_root\" for details.\r\n	2018-09-19T19:01:19.818	aks-agentpool-14693408-0	debian	k8s
stdout	To run a command as administrator (user \"root\"), use \"sudo <com...	2018-09-19T19:01:19.818	aks-agentpool-14693408-0	debian	k8s

Page 1 of 4   50 items per page   1 - 50 of 177 items

Delete cluster

When the cluster is no longer needed, delete the cluster resource, which deletes all associated resources. This operation can be completed in the Azure portal by selecting the **Delete** button on the AKS cluster dashboard. Alternatively, the az aks delete command can be used in the Cloud Shell:

```
az aks delete --resource-group myResourceGroup --name myAKSCluster --no-
wait
```

Note: When you delete the cluster, the Azure Active Directory service principal used by the AKS cluster is not removed. For steps on how to remove the service principal, see **AKS service principal considera-tions and deletion**[10].

Get the code

In this tutorial, pre-created container images have been used to create a Kubernetes deployment. The related application code, Dockerfile, and Kubernetes manifest file are available on GitHub.

https://github.com/Azure-Samples/azure-voting-app-redis

Deploy an AKS cluster using Azure Portal

Now, we'll cover deploying an AKS cluster using the Azure portal. The first step is to sign in to the Azure portal: https://portal.azure.com.

Create an AKS cluster

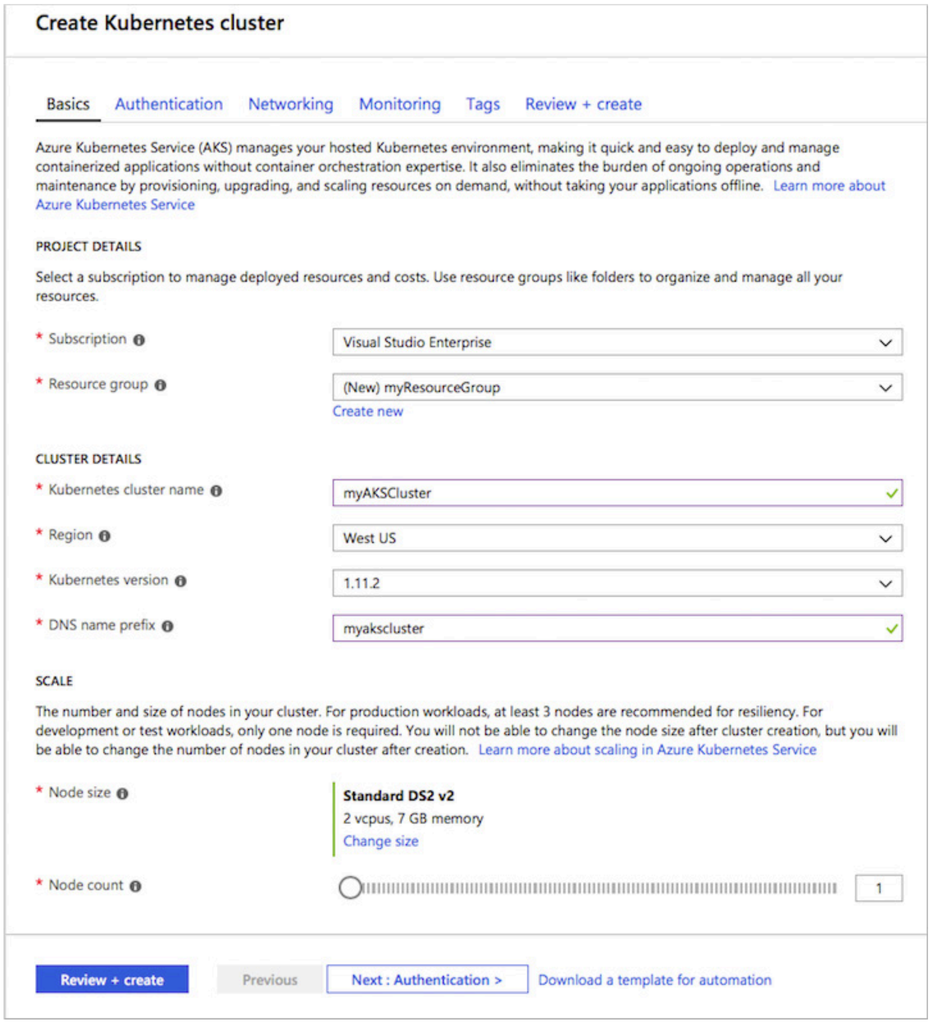
In the top left-hand corner of the Azure portal, select **Create a resource > Kubernetes Service**.

To create an AKS cluster, complete the following steps:

1. **Basics** - Configure the following options:
   - PROJECT DETAILS: Select an Azure subscription, then select or create an Azure resource group, such as *myResourceGroup*. Enter a **Kubernetes cluster name**, such as *myAKSCluster*.
   - CLUSTER DETAILS: Select a region, Kubernetes version, and DNS name prefix for the AKS cluster.
   - SCALE: Select a VM size for the AKS nodes. The VM size cannot be changed once an AKS cluster has been deployed.

     - Select the number of nodes to deploy into the cluster. For this tutorial, set **Node count** to 1. Node count can be adjusted after the cluster has been deployed.

10 https://docs.microsoft.com/en-us/azure/aks/kubernetes-service-principal#additional-considerations



Azure Kubernetes Service (AKS) manages your hosted Kubernetes environment, making it quick and easy to deploy and manage containerized applications without container orchestration expertise. It also eliminates the burden of ongoing operations and maintenance by provisioning, upgrading, and scaling resources on demand, without taking your applications offline. Learn more about Azure Kubernetes Service

PROJECT DETAILS

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

* Subscription	Visual Studio Enterprise
* Resource group	(New) myResourceGroup Create new

CLUSTER DETAILS

* Kubernetes cluster name	myAKSCluster
* Region	West US
* Kubernetes version	1.11.2
* DNS name prefix	myakscluster

SCALE

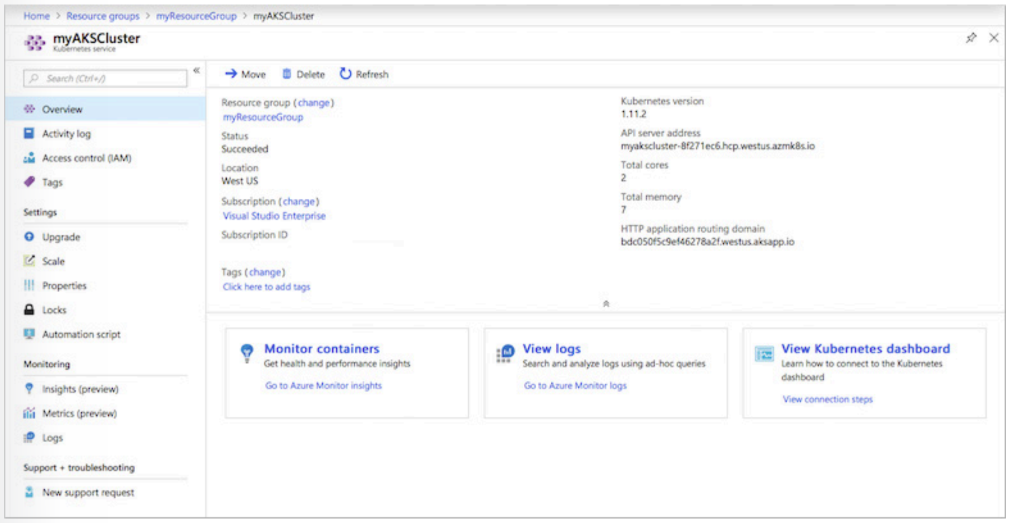
The number and size of nodes in your cluster. For production workloads, at least 3 nodes are recommended for resiliency. For development or test workloads, only one node is required. You will not be able to change the node size after cluster creation, but you will be able to change the number of nodes in your cluster after creation. Learn more about scaling in Azure Kubernetes Service

* Node size	Standard DS2 v2 2 vcpus, 7 GB memory Change size
* Node count	1

1.
2. Select **Next: Authentication** when complete.
3. **Authentication:** Configure the following options:
    - Create a new service principal or *Configure* to use an existing one. When using an existing SPN, you need to provide the SPN client ID and secret.
    - Enable the option for Kubernetes role-based access controls (RBAC). These controls provide more fine-grained control over access to the Kubernetes resources deployed in your AKS cluster.
4. Select **Next: Networking** when complete.
5. **Networking:** Configure the following networking options, which should be set as default:
    - **Http application routing** - Select **Yes** to configure an integrated ingress controller with automatic public DNS name creation.
    - **Network configuration** - Select the **Basic** network configuration using the `kubenet` Kubernetes plugin, rather than advanced networking configuration using Azure CNI.
6. Select **Next: Monitoring** when complete.

7. When deploying an AKS cluster, Azure Container Insights can be configured to monitor health of the AKS cluster and pods running on the cluster.
   - Select **Yes** to enable container monitoring and select an existing Log Analytics workspace, or create a new one.
   - Select **Review + create** and then **Create** when ready.

It takes a few minutes to create the AKS cluster and to be ready for use. Browse to the AKS cluster resource group, such as *myResourceGroup*, and select the AKS resource, such as *myAKSCluster*. The AKS cluster dashboard is shown, as in the following example screenshot:



Resource group (change)
myResourceGroup
Status
Succeeded
Location
West US
Subscription (change)
Visual Studio Enterprise
Subscription ID

Tags (change)
Click here to add tags

Kubernetes version
1.11.2
API server address
myakscluster-8f271ec6.hcp.westus.azmk8s.io
Total cores
2
Total memory
7
HTTP application routing domain
bdc050f5c9ef46278a2f.westus.aksapp.io

Monitor containers
Get health and performance insights
Go to Azure Monitor insights

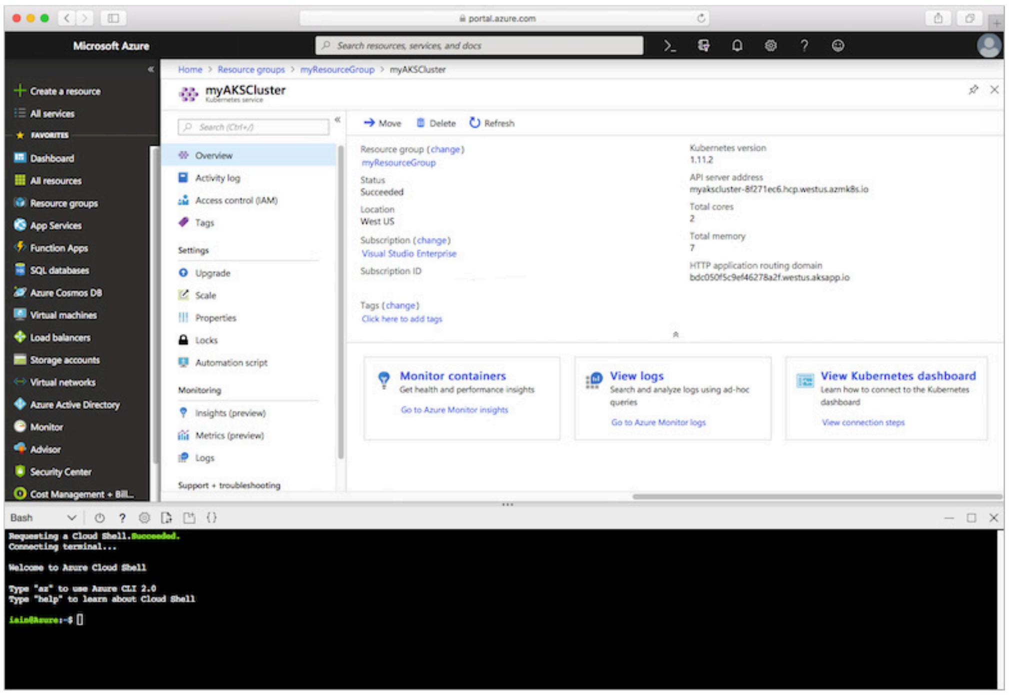
View logs
Search and analyze logs using ad-hoc queries
Go to Azure Monitor logs

View Kubernetes dashboard
Learn how to connect to the Kubernetes dashboard
View connection steps

Connect to the cluster

To manage a Kubernetes cluster, use `kubectl`, the Kubernetes command-line client. The `kubectl` client is pre-installed in the Azure Cloud Shell.

Open Cloud Shell using the button on the top right-hand corner of the Azure portal.



Resource group (change)
myResourceGroup
Status
Succeeded
Location
West US
Subscription (change)
Visual Studio Enterprise
Subscription ID

Tags (change)
Click here to add tags

Kubernetes version
1.11.2
API server address
myakscluster-8f271ec6.hcp.westus.azmk8s.io
Total cores
2
Total memory
7
HTTP application routing domain
bdc050f5c9ef46278a2f.westus.aksapp.io

```
Requesting a Cloud Shell.Succeeded.
Connecting terminal...

Welcome to Azure Cloud Shell

Type "az" to use Azure CLI 2.0
Type "help" to learn about Cloud Shell

iain@Azure:~$ 
```

Use the az `aks get-credentials` command to configure `kubectl` to connect to your Kubernetes cluster. The following example gets credentials for the cluster name *myAKSCluster* in the resource group named *myResourceGroup*:

```
az aks get-credentials --resource-group myResourceGroup --name myAKSCluster
```

To verify the connection to your cluster, use the `kubectl get` command to return a list of the cluster nodes.

```
kubectl get nodes
```

The following example output shows the single node created in the previous steps.

```
NAME                        STATUS    ROLES    AGE    VERSION
aks-agentpool-14693408-0    Ready     agent    10m    v1.11.2
```

Run the application

Kubernetes manifest files define a desired state for a cluster, including what container images should be running. In this quickstart, a manifest is used to create all the objects needed to run a sample Azure Vote application. These objects include two Kubernetes deployments - one for the Azure Vote front end, and the other for a Redis instance. Also, two Kubernetes Services are created - an internal service for the Redis instance, and an external service for accessing the Azure Vote application from the internet.

Create a file named `azure-vote.yaml` and copy into it the following YAML code. If you are working in Azure Cloud Shell, create the file using `vi` or `Nano`, as if working on a virtual or physical system.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-back
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-back
  template:
    metadata:
      labels:
        app: azure-vote-back
    spec:
      containers:
      - name: azure-vote-back
        image: redis
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 250m
            memory: 256Mi
        ports:
        - containerPort: 6379
          name: redis
--- #separator
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-back
spec:
  ports:
  - port: 6379
  selector:
    app: azure-vote-back
--- #separator
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-front
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-front
  template:
    metadata:
      labels:
        app: azure-vote-front
    spec:
      containers:
      - name: azure-vote-front
```

```
        image: microsoft/azure-vote-front:v1
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 250m
            memory: 256Mi
        ports:
        - containerPort: 80
        env:
        - name: REDIS
          value: "azure-vote-back"
--- #separator
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-front
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: azure-vote-front
```

Use the `kubectl apply` command to run the application.

```
kubectl apply -f azure-vote.yaml
```

The following example output shows the Kubernetes resources created on your AKS cluster:

```
deployment "azure-vote-back" created
service "azure-vote-back" created
deployment "azure-vote-front" created
service "azure-vote-front" created
```

Test the application

As the application is run, a Kubernetes service is created that exposes the application front end to the internet. This process can take a few minutes to complete.

To monitor progress, use the `kubectl get service` command with the `--watch` argument.

```
kubectl get service azure-vote-front --watch
```

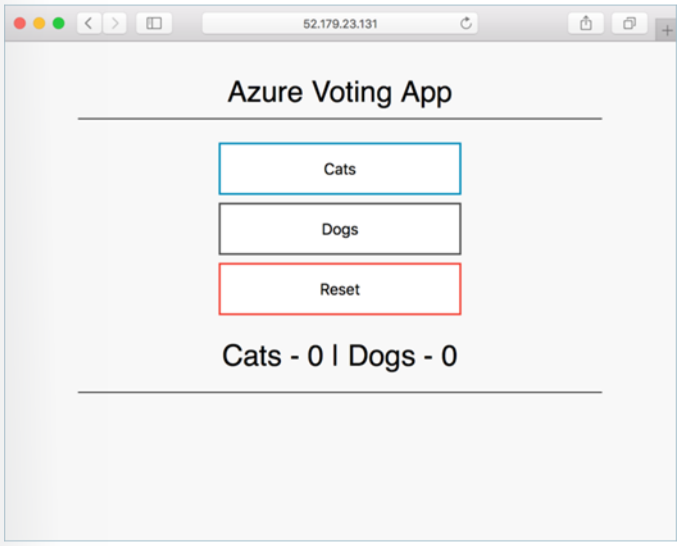
Initially the *EXTERNAL-IP* for the *azure-vote-front* service appears as pending.

```
NAME               TYPE           CLUSTER-IP    EXTERNAL-IP   PORT(S)
AGE
azure-vote-front   LoadBalancer   10.0.37.27    <pending>     80:30572/TCP
6s
```

Once the *EXTERNAL-IP* address has changed from pending to an IP address, use `CTRL-C` to stop the kubectl watch process.

```
azure-vote-front   LoadBalancer   10.0.37.27   52.179.23.131   80:30572/TCP
2m
```

Open a web browser to the external IP address of your service to see the Azure Vote App, as shown in the following example:



Monitor health and logs

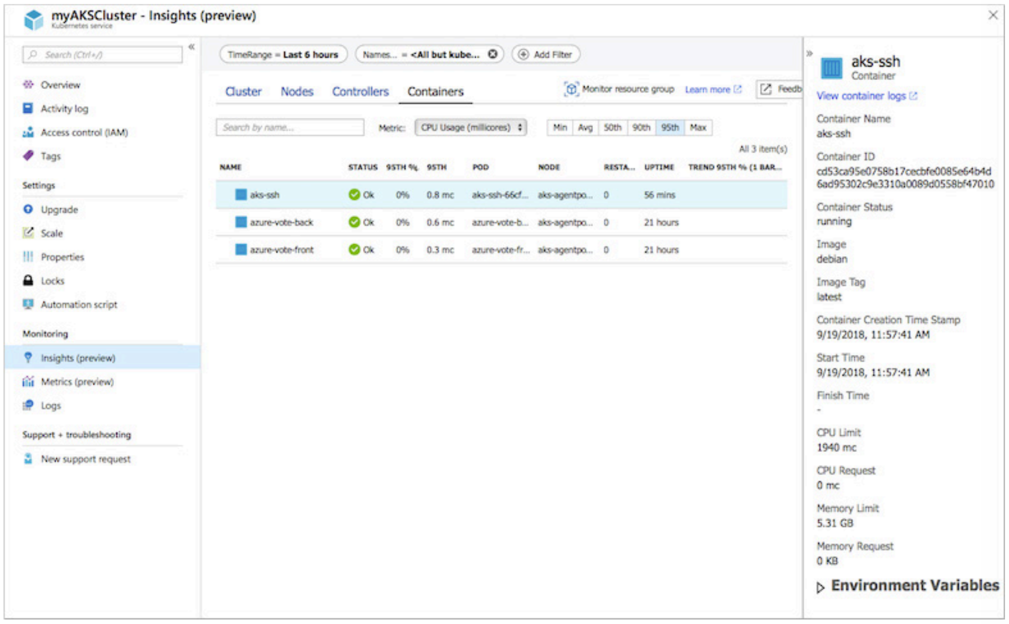
When the AKS cluster was created, monitoring was enabled to capture health metrics for both the cluster nodes and pods. These health metrics are available in the Azure portal. For more information on container health monitoring, see, see **Monitor Azure Kubernetes Service health**[11].

It may take a few minutes for this data to populate in the Azure portal. To see current status, uptime, and resource usage for the Azure Vote pods, complete the following steps:

1. Open a web browser to the Azure portal https://portal.azure.com.
2. Select your resource group, such as *myResourceGroup*, then select your AKS cluster, such as *myAKSCluster*.
3. Under **Monitoring** on the left-hand side, choose **Insights (preview)**
4. Across the top, choose to **+ Add Filter**
5. Select **Namespace** as the property, then choose *<All but kube-system>*
6. Choose to view the **Containers**.

The *azure-vote-back* and *azure-vote-front* containers are displayed, as shown in the following example:

11 https://docs.microsoft.com/en-us/azure/monitoring/monitoring-container-health



NAME	STATUS	95TH %	95TH	POD	NODE	RESTA...	UPTIME	TREND 95TH % (1 BAR...
aks-ssh	Ok	0%	0.8 mc	aks-ssh-66cf...	aks-agentpo...	0	56 mins	
azure-vote-back	Ok	0%	0.6 mc	azure-vote-b...	aks-agentpo...	0	21 hours	
azure-vote-front	Ok	0%	0.3 mc	azure-vote-fr...	aks-agentpo...	0	21 hours	

aks-ssh
Container

Container Name
aks-ssh

Container ID
cd53ca95e0758b17cecbfe0085e64b4d
6ad95302c9e3310a0089d0558bf47010

Container Status
running

Image
debian

Image Tag
latest

Container Creation Time Stamp
9/19/2018, 11:57:41 AM

Start Time
9/19/2018, 11:57:41 AM

Finish Time
-

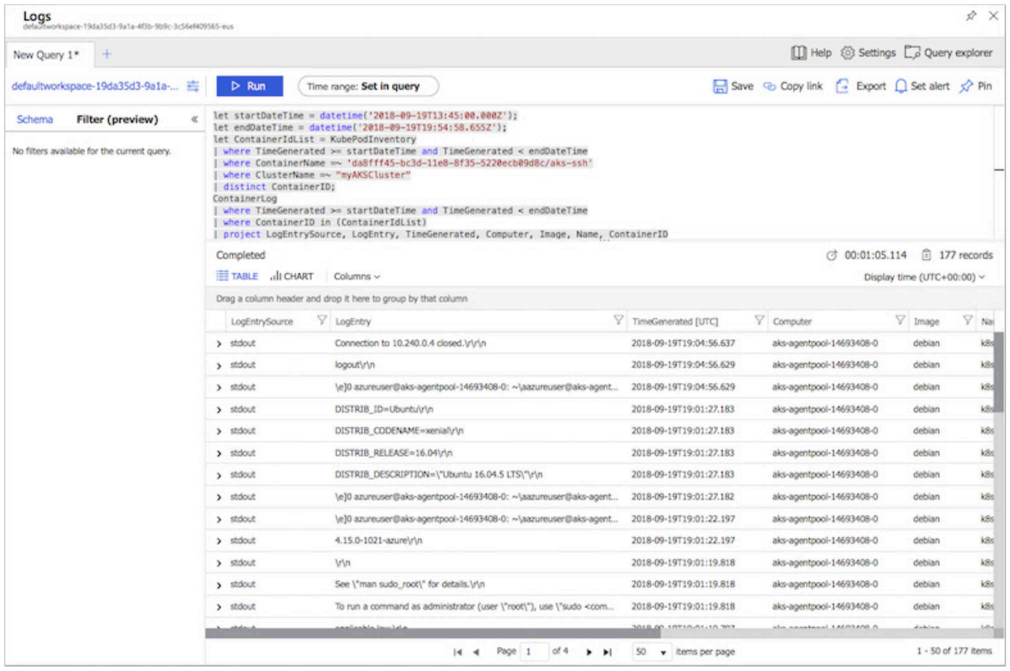
CPU Limit
1940 mc

CPU Request
0 mc

Memory Limit
5.31 GB

Memory Request
0 KB

To see logs for the `azure-vote-front` pod, select the **View container logs** link on the right-hand side of the containers list. These logs include the *stdout* and *stderr* streams from the container.



```
let startDateTime = datetime('2018-09-19T13:45:00.000Z');
let endDateTime = datetime('2018-09-19T19:54:58.655Z');
let ContainerIdList = KubePodInventory
| where TimeGenerated >= startDateTime and TimeGenerated < endDateTime
| where ContainerName =~ 'da8fff45-bc3d-11e8-8f35-5220ecb09d8c/aks-ssh'
| where ClusterName =~ "myAKSCluster"
| distinct ContainerID;
ContainerLog
| where TimeGenerated >= startDateTime and TimeGenerated < endDateTime
| where ContainerID in (ContainerIdList)
| project LogEntrySource, LogEntry, TimeGenerated, Computer, Image, Name, ContainerID
```

LogEntrySource	LogEntry	TimeGenerated [UTC]	Computer	Image	Na
stdout	Connection to 10.240.0.4 closed.\r\r\n	2018-09-19T19:04:56.637	aks-agentpool-14693408-0	debian	k8s
stdout	logout\r\n	2018-09-19T19:04:56.629	aks-agentpool-14693408-0	debian	k8s
stdout	\e]0 azureuser@aks-agentpool-14693408-0: ~\aazureuser@aks-agent...	2018-09-19T19:04:56.629	aks-agentpool-14693408-0	debian	k8s
stdout	DISTRIB_ID=Ubuntu\r\n	2018-09-19T19:01:27.183	aks-agentpool-14693408-0	debian	k8s
stdout	DISTRIB_CODENAME=xenial\r\n	2018-09-19T19:01:27.183	aks-agentpool-14693408-0	debian	k8s
stdout	DISTRIB_RELEASE=16.04\r\n	2018-09-19T19:01:27.183	aks-agentpool-14693408-0	debian	k8s
stdout	DISTRIB_DESCRIPTION=\"Ubuntu 16.04.5 LTS\"\r\n	2018-09-19T19:01:27.183	aks-agentpool-14693408-0	debian	k8s
stdout	\e]0 azureuser@aks-agentpool-14693408-0: ~\aazureuser@aks-agent...	2018-09-19T19:01:27.182	aks-agentpool-14693408-0	debian	k8s
stdout	\e]0 azureuser@aks-agentpool-14693408-0: ~\aazureuser@aks-agent...	2018-09-19T19:01:22.197	aks-agentpool-14693408-0	debian	k8s
stdout	4.15.0-1021-azure\r\n	2018-09-19T19:01:22.197	aks-agentpool-14693408-0	debian	k8s
stdout	\r\n	2018-09-19T19:01:19.818	aks-agentpool-14693408-0	debian	k8s
stdout	See \"man sudo_root\" for details.\r\n	2018-09-19T19:01:19.818	aks-agentpool-14693408-0	debian	k8s
stdout	To run a command as administrator (user \"root\"), use \"sudo <com...	2018-09-19T19:01:19.818	aks-agentpool-14693408-0	debian	k8s

Delete cluster

When the cluster is no longer needed, delete the cluster resource, which deletes all associated resources. This operation can be completed in the Azure portal by selecting the **Delete** button on the AKS cluster dashboard. Alternatively, the az aks delete command can be used in the Cloud Shell:

```
az aks delete --resource-group myResourceGroup --name myAKSCluster --no-
wait
```

Note: When you delete the cluster, the Azure Active Directory service principal used by the AKS cluster is not removed. For steps on how to remove the service principal, see **AKS service principal considerations and deletion**[12].

Get the code

In this tutorial, pre-created container images have been used to create a Kubernetes deployment. The related application code, Dockerfile, and Kubernetes manifest file are available on GitHub.

https://github.com/Azure-Samples/azure-voting-app-redis

12 https://docs.microsoft.com/en-us/azure/aks/kubernetes-service-principal#additional-considerations

Publish a container image to Azure Container Registry

Azure Container Registry overview

Azure Container Registry is a managed Docker registry service based on the open-source Docker Registry 2.0. Create and maintain Azure container registries to store and manage your private Docker container images.

Use container registries in Azure with your existing container development and deployment pipelines. Use Azure Container Registry Build (ACR Build) to build container images in Azure. Build on demand, or fully automate builds with source code commit and base image update build triggers.

For background about Docker and containers, see the **Docker overview**[13].

Use cases

Pull images from an Azure container registry to various deployment targets:

- **Scalable orchestration systems** that manage containerized applications across clusters of hosts, including Kubernetes, DC/OS, and Docker Swarm.
- **Azure services** that support building and running applications at scale, including Azure Kubernetes Service (AKS), App Service, Batch, Service Fabric, and others.

Developers can also push to a container registry as part of a container development workflow. For example, target a container registry from a continuous integration and deployment tool such as Azure DevOps Services or Jenkins.

Configure ACR Tasks to automatically rebuild application images when their base images are updated. Use ACR Tasks to automate image builds when your team commits code to a Git repository.

Key concepts

- **Registry** - Create one or more container registries in your Azure subscription. Registries are available in three SKUs: Basic, Standard, and Premium, each of which support webhook integration, registry authentication with Azure Active Directory, and delete functionality. Take advantage of local, network-close storage of your container images by creating a registry in the same Azure location as your deployments. Use the geo-replication feature of Premium registries for advanced replication and container image distribution scenarios. A fully qualified registry name has the form `myregistry.azurecr.io`.
- You control access to a container registry using an Azure Active Directory-backed service principal or a provided admin account. Run the standard docker login command to authenticate with a registry.
- **Repository** - A registry contains one or more repositories, which store groups of container images. Azure Container Registry supports multilevel repository namespaces. With multilevel namespaces, you can group collections of images related to a specific app, or a collection of apps to specific development or operational teams. For example:
  - `myregistry.azurecr.io/aspnetcore:1.0.1` represents a corporate-wide image

13 https://docs.docker.com/engine/docker-overview/

  - `myregistry.azurecr.io/warrantydept/dotnet-build` represents an image used to build .NET apps, shared across the warranty department
  - `myregistry.azurecr.io/warrantydept/customersubmissions/web` represents a web image, grouped in the customer submissions app, owned by the warranty department
- **Image** - Stored in a repository, each image is a read-only snapshot of a Docker-compatible container. Azure container registries can include both Windows and Linux images. You control image names for all your container deployments. Use standard Docker commands to push images into a repository, or pull an image from a repository. In addition to container images, Azure Container Registry stores related content formats such as Helm charts, used to deploy applications to Kubernetes.
- **Container** - A container defines a software application and its dependencies wrapped in a complete filesystem including code, runtime, system tools, and libraries. Run Docker containers based on Windows or Linux images that you pull from a container registry. Containers running on a single machine share the operating system kernel. Docker containers are fully portable to all major Linux distros, macOS, and Windows.

Azure Container Registry Tasks

Azure Container Registry Tasks (ACR Tasks) is a suite of features within Azure Container Registry that provides streamlined and efficient Docker container image builds in Azure. Use ACR Tasks to extend your development inner-loop to the cloud by offloading `docker build` operations to Azure. Configure build tasks to automate your container OS and framework patching pipeline, and build images automatically when your team commits code to source control.

Multi-step tasks, a preview feature of ACR Tasks, provides step-based task definition and execution for building, testing, and patching container images in the cloud. Task steps define individual container image build and push operations. They can also define the execution of one or more containers, with each step using the container as its execution environment.

Deploy an image to ACR using Azure CLI

Azure Container Registry is a managed Docker container registry service used for storing private Docker container images. This guide details creating an Azure Container Registry instance using the Azure CLI. Then, use Docker commands to push a container image into the registry, and finally pull and run the image from your registry.

This guide requires that you are running the Azure CLI (version 2.0.55 or later recommended). Run `az --version` to find the version. If you need to install or upgrade, see **Install Azure CLI**[14].

You must also have Docker installed locally. Docker provides packages that easily configure Docker on any macOS, Windows, or Linux system.

Because the Azure Cloud Shell doesn't include all required Docker components (the `dockerd` daemon), you can't use the Cloud Shell for this guide.

Create a resource group

Create a resource group with the `az group create` command. An Azure resource group is a logical container into which Azure resources are deployed and managed.

The following example creates a resource group named *myResourceGroup* in the eastus location.

14 https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest

```
az group create --name myResourceGroup --location eastus
```

Create a container registry

In this guide you create a *Basic* registry. Azure Container Registry is available in several different SKUs, described briefly in the following table. For extended details on each, see **Container registry SKUs**[15].

Create an ACR instance using the `az acr create` command. The registry name must be unique within Azure, and contain 5-50 alphanumeric characters. In the following example, *myContainerRegistry007* is used. Update this to a unique value.

```
az acr create --resource-group myResourceGroup --name myContainerRegis-
try007 --sku Basic
```

When the registry is created, the output is similar to the following:

```
{
  "adminUserEnabled": false,
  "creationDate": "2017-09-08T22:32:13.175925+00:00",
  "id": "/subscriptions/00000000-0000-0000-0000-000000000000/resource-
Groups/myResourceGroup/providers/Microsoft.ContainerRegistry/registries/
myContainerRegistry007",
  "location": "eastus",
  "loginServer": "myContainerRegistry007.azurecr.io",
  "name": "myContainerRegistry007",
  "provisioningState": "Succeeded",
  "resourceGroup": "myResourceGroup",
  "sku": {
    "name": "Basic",
    "tier": "Basic"
  },
  "status": null,
  "storageAccount": null,
  "tags": {},
  "type": "Microsoft.ContainerRegistry/registries"
}
```

Important: Take note of `loginServer` in the output, which is the fully qualified registry name (all lowercase). Throughout the rest of this guide `<acrName>` is a placeholder for the container registry name.

Log in to registry

Before pushing and pulling container images, you must log in to the ACR instance. To do so, use the `az acr login` command.

```
az acr login --name <acrName>
```

The command returns a `Login Succeeded` message once completed.

15 https://docs.microsoft.com/en-us/azure/container-registry/container-registry-skus

Push image to registry

To push an image to an Azure Container registry, you must first have an image. If you don't yet have any local container images, run the following `docker pull` command to pull an existing image from Docker Hub. For this example, pull the `hello-world` image.

```
docker pull hello-world
```

Before you can push an image to your registry, you must tag it with the fully qualified name of your ACR login server. The login server name is in the format *<registry-name>.azurecr.io* (all lowercase), for example, *mycontainerregistry007.azurecr.io*.

Tag the image using the `docker tag` command. Replace `<acrLoginServer>` with the login server name of your ACR instance.

```
docker tag hello-world <acrLoginServer>/hello-world:v1
```

Finally, use `docker push` to push the image to the ACR instance. Replace `<acrLoginServer>` with the login server name of your ACR instance. This example creates the **hello-world** repository, containing the `hello-world:v1` image.

```
docker push <acrLoginServer>/hello-world:v1
```

After pushing the image to your container registry, remove the `hello-world:v1` image from your local Docker environment. (Note that this `docker rmi` command does not remove the image from the **hello-world** repository in your Azure container registry.)

```
docker rmi <acrLoginServer>/hello-world:v1
```

List container images

The following example lists the repositories in your registry:

```
az acr repository list --name <acrName> --output table
```

Output:

```
Result
----------------
busybox
```

The following example lists the tags on the **busybox** repository.

```
az acr repository show-tags --name <acrName> --repository busybox --output
table
```

Output:

```
Result
--------
v1
```

Run image from registry

Now, you can pull and run the hello-world:v1 container image from your container registry by using `docker run`:

```
docker run <acrLoginServer>/hello-world:v1
```

Example output:

```
Unable to find image 'mycontainerregistry007.azurecr.io/hello-world:v1' 
locally
v1: Pulling from hello-world
Digest: sha256:662dd8e65ef7ccf13f417962c2f77567d3b132f12c95909de6c85ac-
3c326a345
Status: Downloaded newer image for mycontainerregistry007.azurecr.io/
hello-world:v1

Hello from Docker!
This message shows that your installation appears to be working correctly.

[...]
```

Clean up resources

When no longer needed, you can use the `az group delete` command to remove the resource group, the container registry, and the container images stored there.

```
az group delete --name myResourceGroup
```

Create and run container images in Azure Container Instances

Azure Container Instances overview

Containers are becoming the preferred way to package, deploy, and manage cloud applications. Azure Container Instances offers the fastest and simplest way to run a container in Azure, without having to manage any virtual machines and without having to adopt a higher-level service.

Azure Container Instances is a great solution for any scenario that can operate in isolated containers, including simple applications, task automation, and build jobs. For scenarios where you need full container orchestration, including service discovery across multiple containers, automatic scaling, and coordinated application upgrades, we recommend Azure Kubernetes Service (AKS).

Below is an overview of the features of Azure Container Instances.

Feature	Description
Fast startup times	Containers offer significant startup benefits over virtual machines. Azure Container Instances can start containers in Azure in seconds, without the need to provision and manage VMs.
Public IP connectivity and DNS name	Azure Container Instances enables exposing your containers directly to the internet with an IP address and a fully qualified domain name (FQDN). When you create a container instance, you can specify a custom DNS name label so your application is reachable at *customlabel.azureregion.azurecontainer.io*.
Hypervisor-level security	Historically, containers have offered application dependency isolation and resource governance but have not been considered sufficiently hardened for hostile multi-tenant usage. Azure Container Instances guarantees your application is as isolated in a container as it would be in a VM.
Custom sizes	As demand for resources increases, the nodes of an AKS cluster can be scaled out to match. If resource demand drops, nodes can be removed by scaling in the cluster. AKS scale operations can be completed using the Azure portal or the Azure CLI. For compute-intensive jobs such as machine learning, Azure Container Instances can schedule Linux containers to use NVIDIA Tesla GPU resources (preview).
Persistent storage	To retrieve and persist state with Azure Container Instances, we offer direct mounting of Azure Files shares.

Linux and Windows containers	Azure Container Instances can schedule both Windows and Linux containers with the same API. Simply specify the OS type when you create your container groups. Some features are currently restricted to Linux containers. While we work to bring feature parity to Windows containers, you can find current platform differences in **Quotas and region availability for Azure Container Instances** *(https://docs.microsoft.com/en-us/azure/container-instances/container-instances-quotas)*. Azure Container Instances supports Windows images based on Long-Term Servicing Channel (LTSC) versions. Windows Semi-Annual Channel (SAC) releases like 1709 and 1803 are unsupported.
Co-scheduled groups	Azure Container Instances supports scheduling of multi-container groups that share a host machine, local network, storage, and lifecycle. This enables you to combine your main application container with other supporting role containers, such as logging sidecars.
Virtual network deployment (preview)	Currently in preview, this feature of Azure Container Instances enables deployment of container instances into an Azure virtual network. By deploying container instances into a subnet within your virtual network, they can communicate securely with other resources in the virtual network, including those that are on premises (through VPN gateway or ExpressRoute).

Important: Certain features of Azure Container Instances are in preview, and some **limitations apply**[16]. Previews are made available to you on the condition that you agree to the **supplemental terms of use**[17]. Some aspects of these features may change prior to general availability (GA).

Create container for deployment to ACI

Azure Container Instances enables deployment of Docker containers onto Azure infrastructure without provisioning any virtual machines or adopting a higher-level service. In this tutorial, you package a small Node.js web application into a container image that can be run using Azure Container Instances.

In this article you:

- Clone application source code from GitHub
- Create a container image from application source
- Test the image in a local Docker environment

16 https://docs.microsoft.com/en-us/azure/container-instances/container-instances-vnet#preview-limitations
17 https://azure.microsoft.com/support/legal/preview-supplemental-terms/

In tutorial parts two and three, you upload your image to Azure Container Registry, and then deploy it to Azure Container Instances.

Before you begin

You must satisfy the following requirements to complete this tutorial:

Azure CLI: You must have Azure CLI version 2.0.29 or later installed on your local computer. Run `az --version` to find the version. If you need to install or upgrade, see **Install the Azure CLI**[18].

Docker: This tutorial assumes a basic understanding of core Docker concepts like containers, container images, and basic docker commands. For a primer on Docker and container basics, see the [Docker overview]https://docs.docker.com/engine/docker-overview/.

Docker Engine: To complete this tutorial, you need Docker Engine installed locally. Docker provides packages that configure the Docker environment on **macOS**[19], **Windows**[20], and **Linux**[21].

Important: Because the Azure Cloud shell does not include the Docker daemon, you must install both the Azure CLI and Docker Engine on your local computer to complete this tutorial. You cannot use the Azure Cloud Shell for this tutorial.

Get application code

The sample application in this tutorial is a simple web app built in Node.js. The application serves a static HTML page, and you can use Git to clone the sample application's repository:

```
git clone https://github.com/Azure-Samples/aci-helloworld.git
```

You can also **download the ZIP archive**[22] from GitHub directly.

Build the container image

The Dockerfile in the sample application shows how the container is built. It starts from an **official Node.js image**[23] based on Alpine Linux, a small distribution that is well suited for use with containers. It then copies the application files into the container, installs dependencies using the Node Package Manager, and finally, starts the application.

```
FROM node:8.9.3-alpine
RUN mkdir -p /usr/src/app
COPY ./app/ /usr/src/app/
WORKDIR /usr/src/app
RUN npm install
CMD node /usr/src/app/index.js
```

Use the docker build command to create the container image and tag it as *aci-tutorial-app*:

```
docker build ./aci-helloworld -t aci-tutorial-app
```

18 https://docs.microsoft.com/cli/azure/install-azure-cli
19 https://docs.docker.com/docker-for-mac/
20 https://docs.docker.com/docker-for-windows/
21 https://docs.docker.com/engine/installation/#supported-platforms
22 https://github.com/Azure-Samples/aci-helloworld/archive/master.zip
23 https://store.docker.com/images/node

Output from the docker build command is similar to the following (truncated for readability):

```
$ docker build ./aci-helloworld -t aci-tutorial-app
Sending build context to Docker daemon  119.3kB
Step 1/6 : FROM node:8.9.3-alpine
8.9.3-alpine: Pulling from library/node
88286f41530e: Pull complete
84f3a4bf8410: Pull complete
d0d9b2214720: Pull complete
Digest: sha256:c73277ccc763752b42bb2400d1aaecb4e3d32e3a9dbed-
d0e49885c71bea07354
Status: Downloaded newer image for node:8.9.3-alpine
 ---> 90f5ee24bee2
...
Step 6/6 : CMD node /usr/src/app/index.js
 ---> Running in f4a1ea099eec
 ---> 6edad76d09e9
Removing intermediate container f4a1ea099eec
Successfully built 6edad76d09e9
Successfully tagged aci-tutorial-app:latest
```

Use the `docker images` command to see the built image:

```
docker images
```

Your newly built image should appear in the list:

```
$ docker images
REPOSITORY          TAG       IMAGE ID        CREATED           SIZE
aci-tutorial-app    latest    5c745774dfa9    39 seconds ago    68.1 MB
```

Run the container locally

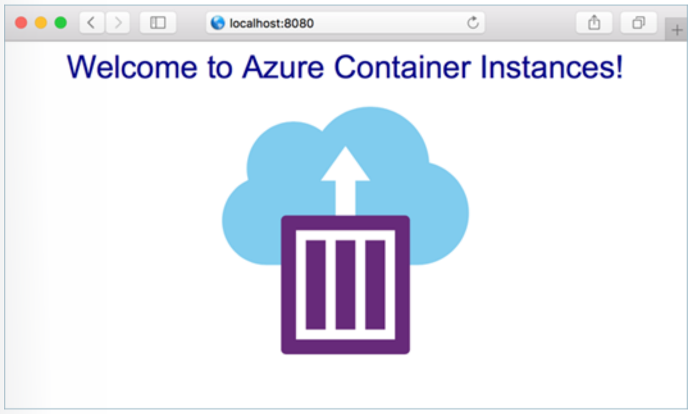
Before you deploy the container to Azure Container Instances, use `docker run` to run it locally and confirm that it works. The `-d` switch lets the container run in the background, while `-p` allows you to map an arbitrary port on your computer to port 80 in the container.

```
docker run -d -p 8080:80 aci-tutorial-app
```

Output from the `docker run` command displays the running container's ID if the command was successful:

```
$ docker run -d -p 8080:80 aci-tutorial-app
a2e3e4435db58ab0c664ce521854c2e1a1bda88c9cf2fcff46aedf48df86cccf
```

Now, navigate to http://localhost:8080 in your browser to confirm that the container is running. You should see a web page similar to the following:



Next Steps

Now that you have created the container, and confirmed that it works, you need to publish the container image to the Azure Container Registry. Follow the steps in the previous lesson.

Deploy a container to ACI

In the last tutorial a container image was create and pushed to the Azure Container Registry. Now, you will deploy the container to Azure Container Instances.

In this tutorial, you:

- Deploy the container from Azure Container Registry to Azure Container Instances
- View the running application in the browser
- Display the container's logs

Before you begin

You must satisfy the following requirements to complete this tutorial:

Azure CLI: You must have Azure CLI version 2.0.29 or later installed on your local computer. Run `az --version` to find the version. If you need to install or upgrade, see **Install the Azure CLI**[24].

Docker: This tutorial assumes a basic understanding of core Docker concepts like containers, container images, and basic docker commands. For a primer on Docker and container basics, see the [Docker overview]https://docs.docker.com/engine/docker-overview/.

Docker Engine: To complete this tutorial, you need Docker Engine installed locally. Docker provides packages that configure the Docker environment on **macOS**[25], **Windows**[26], and **Linux**[27].

Important: Because the Azure Cloud shell does not include the Docker daemon, you must install both the Azure CLI and Docker Engine on your local computer to complete this tutorial. You cannot use the Azure Cloud Shell for this tutorial.

24 https://docs.microsoft.com/cli/azure/install-azure-cli
25 https://docs.docker.com/docker-for-mac/
26 https://docs.docker.com/docker-for-windows/
27 https://docs.docker.com/engine/installation/#supported-platforms

Deploy the container using the Azure CLI

In this section, you use the Azure CLI to deploy the image you have already built and pushed to Azure Container Registry. Be sure you've completed those steps before proceeding.

Get registry credentials

When you deploy an image that's hosted in a private container registry you must supply the registry's credentials.

First, get the full name of the container registry login server (replace `<acrName>` with the name of your registry):

```
az acr show --name <acrName> --query loginServer
```

Next, get the container registry password:

```
az acr credential show --name <acrName> --query "passwords[0].value"
```

Deploy container

Now, use the az container create command to deploy the container. Replace `<acrLoginServer>` and `<acrPassword>` with the values you obtained from the previous two commands. Replace `<acrName>` with the name of your container registry and `<aciDnsLabel>` with desired DNS name.

```
az container create --resource-group myResourceGroup --name aci-tutori-
al-app --image <acrLoginServer>/aci-tutorial-app:v1 --cpu 1 --memory 1
--registry-login-server <acrLoginServer> --registry-username <acrName>
--registry-password <acrPassword> --dns-name-label <aciDnsLabel> --ports 80
```

Within a few seconds, you should receive an initial response from Azure. The `--dns-name-label` value must be unique within the Azure region you create the container instance. Modify the value in the preceding command if you receive a **DNS name label** error message when you execute the command.

Verify deployment progress

To view the state of the deployment, use `az container show`:

```
az container show --resource-group myResourceGroup --name aci-tutorial-app
--query instanceView.state
```

Repeat the `az container show` command until the state changes from *Pending* to *Running*, which should take under a minute. When the container is *Running*, proceed to the next step.

View the application and container logs

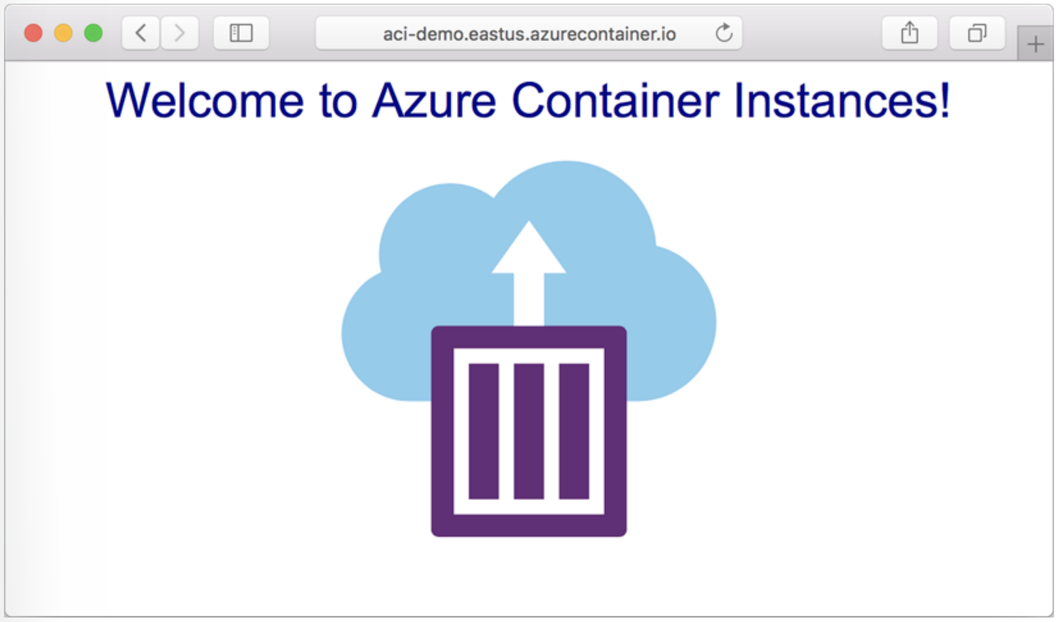
Once the deployment succeeds, display the container's fully qualified domain name (FQDN) with the `az container show` command:

```
az container show --resource-group myResourceGroup --name aci-tutorial-app
--query ipAddress.fqdn
```

For example:

```
$ az container show --resource-group myResourceGroup --name aci-tutori-
al-app --query ipAddress.fqdn
"aci-demo.eastus.azurecontainer.io"
```

To see the running application, navigate to the displayed DNS name in your favorite browser:


aci-demo.eastus.azurecontainer.io
Welcome to Azure Container Instances!

You can also view the log output of the container:

```
az container logs --resource-group myResourceGroup --name aci-tutorial-app
```

Example output:

```
$ az container logs --resource-group myResourceGroup --name aci-tutori-
al-app
listening on port 80
::ffff:10.240.0.4 - - [21/Jul/2017:06:00:02 +0000] "GET / HTTP/1.1" 200 1663
"-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36"
::ffff:10.240.0.4 - - [21/Jul/2017:06:00:02 +0000] "GET /favicon.ico
HTTP/1.1" 404 150 "http://aci-demo.eastus.azurecontainer.io/" "Mozilla/5.0
(Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/59.0.3071.115 Safari/537.36"
```

Clean up resources

If you no longer need any of the resources you created in this tutorial series, you can execute the az group delete command to remove the resource group and all resources it contains. This command deletes the container registry you created, as well as the running container, and all related resources.

```
az group delete --name myResourceGroup
```

Implement an application using Virtual Kublet

When using the Virtual Kubelet provider for Azure Container Instances, both Linux and Windows containers can be scheduled on a container instance as if it is a standard Kubernetes node. This configuration allows you to take advantage of both the capabilities of Kubernetes and the management value and cost benefit of container instances.

Note: Virtual Kubelet is an experimental open source project and should be used as such. To contribute, file issues, and read more about virtual kubelet, see the **Virtual Kubelet GitHub project**[28].

Prerequisites

This guide assumes that you have an AKS cluster. You also need the Azure CLI version 2.0.33 or later. Run `az --version` to find the version.

To install the Virtual Kubelet, **Helm**[29] is also required.

For RBAC-enabled clusters

If your AKS cluster is RBAC-enabled, you must create a service account and role binding for use with Tiller.

A *ClusterRoleBinding* must also be created for the Virtual Kubelet. To create a binding, create a file named *rbac-virtualkubelet.yaml* and paste the following definition:

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: virtual-kubelet
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: default
  namespace: default
```

Apply the binding with `kubectl apply` and specify your *rbac-virtualkubelet.yaml* file, as shown in the following example:

```
$ kubectl apply -f rbac-virtual-kubelet.yaml

clusterrolebinding.rbac.authorization.k8s.io/virtual-kubelet created
```

You can now continue to installing the Virtual Kubelet into your AKS cluster.

28 https://github.com/virtual-kubelet/virtual-kubelet
29 https://docs.helm.sh/using_helm/#installing-helm

Installation

Use the `az aks install-connector` command to install Virtual Kubelet. The following example deploys both the Linux and Windows connector.

```
az aks install-connector --resource-group myAKSCluster --name myAKSCluster
--connector-name virtual-kubelet --os-type Both
```

These arguments are available for the `aks install-connector` command.

Argument	Description	Required
`--connector-name`	Name of the ACI Connector.	Yes
`--name`	Name of the managed cluster.	Yes
`--resource-group`	Name of resource group.	Yes
`--os-type`	Container instances operating system type. Allowed values: Both, Linux, Windows. Default: Linux.	No
`--aci-resource-group`	The resource group in which to create the ACI container groups.	No
`--location`	The location to create the ACI container groups.	No
`--service-principal`	Service principal used for authentication to Azure APIs.	No
`--client-secret`	Secret associated with the service principal.	No
`--chart-url`	URL of a Helm chart that installs ACI Connector.	No
`--image-tag`	The image tag of the virtual kubelet container image.	No

Validate Virtual Kubelet

To validate that Virtual Kubelet has been installed, return a list of Kubernetes nodes using the `kubectl get nodes` command.

```
$ kubectl get nodes

NAME                                    STATUS    ROLES     AGE       VER-
SION
aks-nodepool1-23443254-0                Ready     agent     16d
v1.9.6
aks-nodepool1-23443254-1                Ready     agent     16d
v1.9.6
aks-nodepool1-23443254-2                Ready     agent     16d
v1.9.6
virtual-kubelet-virtual-kubelet-linux   Ready     agent     4m
v1.8.3
virtual-kubelet-virtual-kubelet-win     Ready     agent     4m
v1.8.3
```

Run Linux container

Create a file named *virtual-kubelet-linux.yaml* and copy in the following YAML. Replace the `kubernetes.io/hostname` value with the name of the Linux Virtual Kubelet node. Take note that a `nodeSelector` and `toleration` are being used to schedule the container on the node.

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: aci-helloworld
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: aci-helloworld
    spec:
      containers:
      - name: aci-helloworld
        image: microsoft/aci-helloworld
        ports:
        - containerPort: 80
      nodeSelector:
        kubernetes.io/hostname: virtual-kubelet-virtual-kubelet-linux
      tolerations:
      - key: azure.com/aci
        effect: NoSchedule
```

Run the application with the `kubectl create` command.

```
kubectl create -f virtual-kubelet-linux.yaml
```

Use the `kubectl get pods` command with the `-o wide` argument to output a list of pods with the scheduled node. Notice that the `aci-helloworld` pod has been scheduled on the `virtual-kubelet-virtual-kubelet-linux` node.

```
$ kubectl get pods -o wide

NAME                                READY     STATUS    RESTARTS   AGE
IP            NODE
aci-helloworld-2559879000-8vmjw     1/1       Running   0          39s
52.179.3.180  virtual-kubelet-virtual-kubelet-linux
```

Run Windows container

Create a file named *virtual-kubelet-windows.yaml* and copy in the following YAML. Replace the `kubernetes.io/hostname` value with the name of the Windows Virtual Kubelet node. Take note that a `nodeSelector` and `toleration` are being used to schedule the container on the node.

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nanoserver-iis
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: nanoserver-iis
    spec:
      containers:
      - name: nanoserver-iis
        image: nanoserver/iis
        ports:
        - containerPort: 80
      nodeSelector:
        kubernetes.io/hostname: virtual-kubelet-virtual-kubelet-win
      tolerations:
      - key: azure.com/aci
        effect: NoSchedule
```

Run the application with the `kubectl create` command.

```
kubectl create -f virtual-kubelet-windows.yaml
```

Use the `kubectl get pods` command with the `-o wide` argument to output a list of pods with the scheduled node. Notice that the nanoserver-iis pod has been scheduled on the `virtual-kubelet-virtual-kubelet-win` node.

```
$ kubectl get pods -o wide

NAME                                READY     STATUS    RESTARTS   AGE
IP              NODE
nanoserver-iis-868bc8d489-tq4st     1/1       Running   8          21m
138.91.121.91   virtual-kubelet-virtual-kubelet-win
```

Review questions

Module 3 review questions

Kubernetes cluster architecture

Azure Kubernetes Service (AKS) provides a managed Kubernetes service that reduces the complexity for deployment and core management tasks, including coordinating upgrades. The AKS cluster masters are managed by the Azure platform. What are the two components that make up a Kubernetes cluster?

> Click to see suggested answer

A Kubernetes cluster is divided into two components:

- Cluster master nodes provide the core Kubernetes services and orchestration of application workloads. When you create an AKS cluster, a cluster master is automatically created and configured. This cluster master is provided as a managed Azure resource abstracted from the user.
- Nodes run your application workloads. To run your applications and supporting services, you need a Kubernetes node. An AKS cluster has one or more nodes, which is an Azure virtual machine (VM) that runs the Kubernetes node components and container runtime.

Azure Container Registry

How would you use the Azure Container Registry in your workflow?

> Click to see suggested answer

Azure Container Registry is a managed Docker registry service based on the open-source Docker Registry 2.0. Create and maintain Azure container registries to store and manage your private Docker container images.

Use container registries in Azure with your existing container development and deployment pipelines. Use Azure Container Registry Build (ACR Build) to build container images in Azure. Build on demand, or fully automate builds with source code commit and base image update build triggers.

Developers can also push to a container registry as part of a container development workflow. For example, target a container registry from a continuous integration and deployment tool such as Azure DevOps Services or Jenkins.