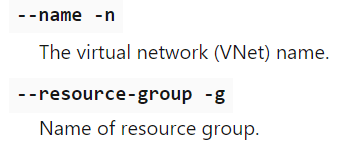
# General stuff

With Azure CLI, parameters have two dashes before them if fully qualified, else only one:



A resource provider is a service that offers the different kinds of Azure resources, and they manage the resource’s lifecycle.

* Microsoft.Compute: in charge of offering virtual machine resources
* Microsoft.Storage

# Virtual Machines

## Provision VM

**Limits:** Every subscription has default quota limits: 20 VMs / Regio (contact Azure support service if needed more)

**Extensions:** After deployment additional capabilities: Run custom scripts, deploy/manage configurations, collect diagnostic data

Virtual disks: .vhd files, stored as Page Blobs in Storage Account (if managed, then it won’t appear in VM configuration).

If your virtual machines use a load balancer, you need to put your VMs that host the application into an availability set.  
This will ensure that any virtual machine in the same availability set will never be on the same hardware and won’t be restarted at the same time because of software upgrades on the servers running the VM.

If you forget to add the VM to an availability set, you need to delete the VM and start from the beginning.

A Virtual Machine is an Infrastructure as a Service (IaaS), because it provides full access to its OS.  
VMs have durable, stateful storages.

**Virtual Machines consist of:**

1. Image: *Deployed onto VM, the OS*
2. Network
3. Block storage: *store VM and app data*

**Types of Virtual Machines:**

* General purpose
* Compute optimized
* Memory optimized
* Storage optimized
* Graphics Processing Unit (GPU)
* ****High performance compute

**Storage options:** HDD or SSD.  
Choose Standard SSD disks if you have normal workloads but want better performance.  
Choose Premium SSD disks if you have I/O intensive workloads or mission-critical systems that need to process data very quickly.

Azure uses virtual hard disks (VHDs) to represent physical disks for the VM.  
VHDs replicate the logical format and data of a disk drive but are stored as page blobs in an Azure Storage account.  
You can choose on a per disk basis what type of storage it should use (SSD or HDD).

For a Linux VM, two VHDs are created:

* **One for the OS:** primary disk, max. 2048GB. labeled as /dev/sda by default.
* **One temporary:** Stores swap files, its size is based on the VM’s size. Labeled as /dev/sdb and is formatted and mounted to /mnt by the Azure Linux Agent. **Not persistent, do not write to this important stuff.**

Where to store data? Either on VHD for OS, or better: create dedicated data disks. Data Disk can be up to 4095GBs. Can be created from a real disk, thus easing migrations.

1. Managed disks
   1. This is the modern, recommended approach
   2. I specify the disk type (Premium or Standard) and the size of the disk, and Azure creates and manages both the disk *and* the storage it uses
   3. **Increased reliability:** Azure ensures that VHDs associated with high-reliability VMs will be placed in different parts of Azure Storage to provide similar levels of resilience.
   4. **Better security:** Real managed resources in the resource group: means they can use role-based access control to restrict who can work with the VHD data.
   5. **Snapshot support:** Create a read-only copy of a VHD. You have to shut down the owning VM, but creating the snapshot only takes a few seconds. Once it's done, you can power on the VM and use the snapshot to create a duplicate VM to troubleshoot a production issue or roll back the VM to the point in time that the snapshot was taken.
   6. **Backup support:** Managed disks can be automatically backed up to different regions for disaster recovery with Azure Backup without affecting the service of the VM.
2. Unmanaged disks
   1. I am responsible for the storage accounts that are used to hold the VHDs that correspond to my VM disks
   2. pay the storage account rates for the amount of space I use
   3. A single storage account has a fixed rate limit of 20,000 I/O operations/sec. This means that a single storage account is capable of supporting 40 standard virtual hard disks at full throttle. If you need to scale out, then you need more than one storage account, which can get complicated.

**VNet**

Virtual machines communicate with external resources using a virtual network (VNet).

* A VNet represents a private network in a single region that your resources communicate on.
* Virtual Networks can be divided up with subnets to isolate resources, connect them to other networks (including on-premises networks), and apply traffic rules to govern inbound and outbound connections.

In a production environment where we already have other components, you'd want to utilize an *existing* virtual network. That way, your VM can communicate with the other cloud services in your solution. If there isn't one defined in this location yet, you can create it here and configure

**Remote connections:**

There are two approaches we can use to authenticate an SSH connection:

1. username and password
2. SSH key pair.

Although SSH provides an encrypted connection, using passwords with SSH connections leaves the VM vulnerable to brute-force attacks of passwords. A more secure and preferred method of connecting to a Linux VM with SSH is a public-private key pair, also known as SSH keys.

With an SSH key pair, you can sign in to Linux-based Azure virtual machines without a password. This is a more secure approach if you only plan to sign in to the VM from a few computers. If you need to be able to access the Linux VM from a variety of locations, a username and password combination might be a better approach.

1. The public key is placed on your Linux VM or any other service that you wish to use with public-key cryptography. This can be shared with anyone.
2. The **private key** is what you present to verify your identity to your Linux VM when you make an SSH connection. Consider this confidential information and protect this like you would a password or any other private data.  
   1. Private key passphrase: You can optionally provide a passphrase while generating your private key. This is a password you must enter when you use the key. This passphrase is used to access the private SSH key file and is not the user account password. When you add a passphrase to your SSH key, it encrypts the private key using 128-bit AES so that the private key is useless without the passphrase to decrypt it.

use the built-in ssh-keygen command to generate the SSH public and private key files.

When creating virtual network for VM, specify a subnet: by default, Azure will create a virtual network, network interface, and public IP for it.

**Create from Portal:**

1. **Virtual Network:** VNet enables many types of Azure resources, such as Azure Virtual Machines (VM), to securely communicate with each other, the internet, and on-premises networks.  
   1. Address Space:  
      A custom private IP address space using public and private (RFC 1918) addresses.  
      Azure assigns resources in a virtual network a private IP address from the address space that you assign.  
      For example, if you deploy a VM in a VNet with address space, 10.0.0.0/16, the VM will be assigned a private IP like 10.0.0.4.
   2. Subnet:  
      Enables us to segment the virtual network into one or more sub-networks and allocate a portion of the virtual network's address space to each subnet.  
      We can deploy Azure resources in a specific subnet.  
      Like in a traditional network, subnets allow us to segment VNet address space into segments that are appropriate for the organization's internal network.  
      This also improves address allocation efficiency. Securing resources: Network Security Groups.
      1. Name
      2. Address Range
2. **Virtual Machine**
   1. Managed storage
   2. Make sure to select inbound ports (specify RDP in case of Windows VM, SSH for Linux, but even stuff like HTTP, HTTPS, MS SQL are available)
   3. Post config stuff:
      1. Backups: Whole VM snapshots
      2. Managed Service Identity: Configure VM with Azure Active Directory
      3. Boot diagnostics: Logs, screenshots taken from VM (placed into Storage Account)

Note: When creating Linux VM, we can use an SSH public key, instead of passwords for authentication.

**Create from Azure CLI:**

1. Virtual Network (vnet)
2. IP
3. Network Security Group (nsg)
4. Network Interface (nic)
5. VM
6. Open port
7. Get IP of VM

az login --subscription "Name of my subscription"

az group create \

  --name "name-of-new-resource-group" \

  --location "centralus"

az network vnet create \

  --resource-group "name-of-new-resource-group" \

  --name  "name-of-new-virtual-network" \

  --address-prefix "10.1.0.0/16" \

  --subnet-name "name-of-new-subnet" \

  --subnet-prefix "10.1.1.0/24" \

# Useful: List virtual networks

az network vnet list -o table

# Useful: Check if private IP address is available for use within a VNet:

az network vnet check-ip-address \

  -g "name-of-new-resource-group"

  -n "name-of-new-vnet"

  --ip-address 10.0.0.4

az network public-ip create \

  -g "name-of-new-resource-group" \

  -n "name-of-new-ip-address"

# List network security groups, then create new

az network nsg list -o table

az network nsg create \

  -g "name-of-new-resource-group" \

  -n "name-of-new-network-security-group"

# Create Virtual Network Interface, associate with:

# 1) IP address

# 2) Network Sec. Group

az network nic create \

 -g "name-of-new-resource-group" \

 -n "name-of-new-nic" \

 --vnet-name "name-of-new-virtual-network"

 --subnet "name-of-new-subnet" \

 --network-security-group "name-of-new-network-security-group"

 --public-ip-address "name-of-new-ip-address"

az network nic list -o table

az vm create \

  -g "name-of-new-resource-group" \

  -l "centralus" \

  -n "name-of-new-vm" \

  --nics "name-of-new-nic"

  --image "rhel" \

  --admin-username "demo-admin" \

  --authentication-type "ssh" \

  --ssh-key-value ~/.ssh/id\_rsa.pub

az vm open-port \

  -g "name-of-new-resource-group" \

  -n "name-of-new-vm" \

  --port "22"

az vm list-ip-addresses \

  -n "name-of-new-vm"

  -o table

# SSH into VM with the returned IP:

ssh -l demoadmin nnn.nn.nnn.nn

If things are confusing:

az vm create --help | more

If we want to simplify it:

az vm create \

  -g "name-of-new-resource-group" \

  -n "name-of-new-vm" \

  --nics "name-of-new-nic" THIS CAN BE OMMITTED

  --image "rhel" \

  --admin-username "demo-admin" \

  --authentication-type "ssh" \

  --ssh-key-value ~/.ssh/id\_rsa.pub

For Windows VMs:

az vm create \

  -g "name-of-new-resource-group" \

  -l "centralus" \

  -n "name-of-new-vm" \

  --nics "name-of-new-nic"

  --image "win2016datacenter" \

  --admin-username "demo-admin" \

  --admin-password "password1234"

Same, but with PowerShell:

$SubscriptionId = 123456789

Connect-AzureRmAccount -Subscription $SubscriptionId

$resourceGroup = Get-AzureRmResourceGroup `

  -Name 'new-resource-group' `

  -Location 'centralus'

$subnetConfig = New-AzureRmVirtualNetworkSubnetConfig `

  -Name 'new-subnet-config' `

  -AddressPrefix '10.2.1.0/24'

$vnet = New-AzureRmVirtualNetwork `

  -ResourceGroupName $resourceGroup.ResourceGroupName `

  -Location $resourceGroup.Location `

  -Name 'new-vnet' `

  -AddressPrefix '10.2.0.0/16' `

  -Subnet $subnetConfig

$publicIp = New-AzureRmPublicIpAddress `

  -ResourceGroupName $resourceGroup.ResourceGroupName `

  -Location $resourceGroup.Location `

  -Name 'new-public-ip' `

  -AllocationMethod Static

$networkSecRuleConfig = New-AzureRmNetworkSecurityRuleConfig `

  -Name 'network-sec-rule-config' `

  -Description 'Allow SSH' `

  -Access Allow `

  -Protocol Tcp `

  -Direction Inbound `

  -Priority 100 `

  -SourceAddressPrefix 'Internet' `

  -SourcePortRange \* `

  -DestinationAddressPrefix \* `

  -DestinationPortRange 22

# With Azure CLI, the port had to be opened manually, but not with this

# Priority is sequential (100 gets executed before 101)

$networkSecGroup = New-AzureRmNetworkSecurityGroup `

  -ResourceGroupName $resourceGroup.ResourceGroupName `

  -Location $resourceGroup.Location `

  -Name 'new-network-sec-group' `

  -SecurityRules $networkSecRuleConfig `

$subnet = $vnet.Subnets | Where-Object { $\_.Name -eq $subnetConfig.Name }

$nic = New-AzureRmNetworkInterface `

  -ResourceGroupName $resourceGroup.ResourceGroupName `

  -Location $resourceGroup.Location `

  -Name 'new-nic' `

  -Subnet $subnet `

  -PublicIpAddress $publicIp `

  -NetworkSecurityGroup $networkSecGroup

$vmConfig = New-AzureRmVMConfig `

  -VMName 'name-of-new-vm' `

  -VMSize 'Standard\_D1'

$password = ConvertTo-SecureString 'password1234' `

  -AsPlainText `

  -Force

$credential = New-Object System.Management.Automation.PSCredential ('demoadmin', $password)

$vmConfig = Set-AzureRmVMOperatingSystem `

  -VM $vmConfig `

  -Linux `

  -ComputerName 'vm-computer-name' `

  -DisablePasswordAuthentication `

  -Credential $credential

$sshPublicKey = Get-Content "~/.ssh/id\_rsa.pub"

Add-AzureRmVMSshPublicKey `

  -VM $vmConfig `

  -KeyData $sshPublicKey `

  -Path "/home/demoadmin/.ssh/authorized\_keys"

$vmConfig = Set-AzureRmVMSourceImage `

  -VM $vmConfig `

  -PublisherName 'Redhat' `

  -Offer 'rhel' `

  -Skus '7.4' `

  -Version 'latest'

# Assign NIC

$vmConfig = Add-AzureRmVMNetworkInterface `

  -VM $vmConfig `

  -Id $nic.Id

New-AzureRmVM `

  -ResourceGroupName $resourceGroup.ResourceGroupName `

  -Location $resourceGroup.Location `

  -VM $vmConfig

$IpOfVm = Get-AzureRmPublicIpAddress `

  -ResourceGroupName $resourceGroup.ResourceGroupName `

  -Location $resourceGroup.Location | Select-Object -ExpandProperty -PublicIpAddress

$IpOfVm

## ARM template

{

"$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",

"contentVersion": "",

"parameters": {},

"variables": {},

"functions": [],

"resources": [],

"outputs": {}

}

Schema: Can be ResourceDeployment or SubscriptionDeployment  
Contentversion: My internal versioning, such as 1.0.0 (Azure doesn’t care)

"parameters": {

"virtualNetworks\_az203VNET\_name": {

"defaultValue": "az203demoVNET",

"type": "string"

}

}

Default value can be:

* String
* Securestring
* Bool
* Int
* Object
* SecureObject
* Array

(optional) Default value is used, instead of value, because value is provided during deployment. If value is not set, defaultValue is used.

Values can be stored in separate JSON (**use value there, not defaultvalue**), or provided as arguments

"myParameter": {

"value": "[concat(parameters('myOtherParameter'), 'randomString')]"

}

Variables are more straight forward:

"variables": {

"osDiskName": "\_OSDisk1\_1\_39c654d89d88405e968db84b722002d1"

},

Functions:

* Custom functions cannot access template variables, although you can pass them as a parameter of your function.
* Your custom function cannot access the template’s parameters; instead, they have access only to the parameters that you define in your function.
* Parameters on your custom function cannot have default values.
* Your custom function cannot call other custom functions; only predefined functions may be called.
* You cannot use the reference() predefined function.

## Azure Disk Encryption Configuration

Virtual disks—managed or unmanaged disks—are automatically and transparently encrypted and decrypted by the Azure Storage Service Encryption (SSE) service.

* But only for data at rest
* Virtual disk file is encrypted, but the information inside the virtual disk **is not**
  + Use Azure Disk Encryption service (KeyVault for storing keys) (BitLocker in Windows + DM-Crypt in Linux)
* Enable on new Windows IaaS VM
* Enable on existing VMs: min 7 GB memory
* Enable on VM Scale Sets:
  + Windows: OS + data
  + Linux: data
* Enable on managed disks
* Disable on VMs:
  + Windows: can be done, whether standalone VM or in scale set (OS + data)
  + Linux: same, but only for data disks
* Backup & restore encrypted VMs: use Azure Backup Service
* Disk encryption is **NOT POSSIBLE ON BASIC TIER**
* OS drive encryption **should be disabled on Linux**
  + If OS drive is encrypted, **do not encrypt data disks**
* Encryption on **CUSTOM LINUX IMAGES** should be enabled
* Classic IaaS VMs are not supported
* Encryption can’t be used for:
  + Azure Files
  + Network File Systems (NFS)
  + Dynamic volumes
  + Windows VMs with software-based RAID

General security measures for VMs:

* Antimalware (not on Linux, Windows Server 2008)
* Azure Security Center: detect threats
  + Just in time access: applied to VMs deployment to lock down inbound traffic. When user requests access to VM, Security Center checks the user’s permissions for the VM (if OK, Security Center configures an NSG (network sec group) to allow inbound traffic to the selected ports for limited time)
* Encryption
* Key vault & SSH keys
* Managed identities for Azure resources
  + Provides Azure services with automatically managed ID in Azure AD
  + Use this identity to authenticate to service (no need to store stuff in code, authenticate to KeyVault, etc)
* Policies
* RBAC

VM must be:

* Azure endorsed Linux distro (Ubuntu [both], RHEL [both], CentOS [both], openSUSE [data disk], SLES [data disk])
* NOT Basic, NOT A-Series
* Must have 2GB memory, if encrypting only data disks
* Must have 8GB memory, if encrypting data + OS and root file system (/) usage is <4GB
* Must have root system usage \* 2, if encrypting data + OS and root file system (/) usage is >4GB
* Must have modules present: **dm-crpypt**, **vfat**

Uses DM-Crpyt of **Linux**.  
Provides volume encryption for the OS and data disks.  
Integrated with Key Vault to manage keys and secrets.

If using Azure Security Center, if a VM isn’t encrypted, an alert is sent (High Severity).

Works on Premium Disks as well.

Needs a **Key Vault**, where Access Policy is set to “**Azure Disk Encryption for volume encryption**”  
In VM: Settings/Disks -> Encryption

With CLI:

Create VM, etc

az keyvault create \

  --name "<your-unique-keyvault-name>" \

  --resource-group "myResourceGroup" \

  --location "eastus" \

  --enabled-for-disk-encryption

az vm encryption enable \

  -g "MyResourceGroup" \

  -n "myVM" \

  --disk-encryption-keyvault "<your-unique-keyvault-name>"

To monitor progress: az vm show --name "myVM" -g "MyResourceGroup

With PowerShell:

Create VM, etc

New-AzKeyvault `

  -name "<your-unique-keyvault-name>" `

  -ResourceGroupName "myResourceGroup" `

  -Location EastUS `

  -EnabledForDiskEncryption

Set-AzVMDiskEncryptionExtension `

  -ResourceGroupName MyResourceGroup `

  -VMName "MyVM" `

  -DiskEncryptionKeyVaultUrl $KeyVault.VaultUri `

  -DiskEncryptionKeyVaultId $KeyVault.ResourceId `

  -SkipVmBackup `

  -VolumeType All

# Batch jobs

## Manage Batch Jobs via Batch Service API

## Run batch job with CLI, Portal, other tools

## Write code to run as Azure Batch Services batch job

# Containerized solutions

## Create AKS cluster

You won’t be able to change the size of the nodes after you have created the cluster.  
You can change the number of nodes associated to the cluster.

At least 3 nodes are recommended.

Ensure that the option **HTTP Application Routing** is **not enabled**.This option will create an HTTP ingress controller that routes HTTP traffic to your application. This configuration is not recommended for production (only testing, dev)

* **Cluster master nodes** These are the nodes that manage the cluster and provide the core orchestration services.
* **Nodes** A node contains the VMs where your container will be executed.

You are not charged for any of these cluster master nodes.  
All master nodes are dedicated to the same tenant

**Master nodes have these:**

* API server: expose APIs needed by Kubernetes (managed by kubectl, Dashboard)
* Scheduler
* Etcd: Key value store, maintains config and state of cluster
* Controller Manager: Manages controllers (which are control loop watching for shared state of the cluster; they work with API server to query state, makes changes to achieve desired state
  + Controllers can be for: deployments, cron jobs, garbage collecting, namespaces, etc

Because the master nodes are managed by Azure, you cannot directly connect to these nodes to perform any confi guration or troubleshooting. (check AKS cluster logs, Azure Monitor logs)

**Nodes have these:**

* **Kubernetes agent:** (kubelet), receives order from orchestrator and executes them
* **Proxy**
* **Container runtime**

How to deploy containers to AKS cluster

* **Pod:** logical resource Kubernetes uses to running an instance of container (1:1 relationship)  
  a structure needed by Kubernetes to define resource limits, plan the schedule, and deploy the container
* **Deployments:** Identical Pods can be grouped in deployment  
  Every deployment is managed by the Deployment Controller.
* **YAML manifests:** contains the entire defi nition of the resource that needs to be deployed to the AKS cluster
* **StatefulSets:** Similar to Deployments, managed by the StatefulSet Controller. any pod managed by a StatefulSet is assigned a static network identifi er, the storage assigned to the pod is persistent, the deployment and scaling of the pod is ordered and graceful across the nodes
* **DaemonSets:** Pods in this are guaranteed to be deployed in everty node specified in the manifest (usually for log collection, monitoring)

## Create container images

Docker is used by everything (Kubernetes, Service Fabric, Container Registry, Container Instaces etc)

Create Docker images: **docker build (use tags to specify version)**

**docker image ls**

**Docker Compose:** the definition of images for prod (multiple images? Define relationships with this)

Service: in docker, a service is each of the pieces of the app (1:1 relationship with the image)  
BUT, a service can have multiple instances of the same image (multiple containers)

## Publish to Azure Container Registry

Microsoft’s implementation of Docker registry (2.0 definition)

Privately store images, build images on the fly, automating the build of images based on commits, etc.

**Tagging an image is required:** <acr\_name>.azurecr.io/[repository\_name][:version]

* **acr\_name**: custom name of our registry
* **repo name**: optional, for a repo within the registry. ACR allows to use multi level repos within the registry
* **version**

docker tag foobar <acr\_name>.azurecr.io/<repository\_name>/<image\_name>

docker push <acr\_name>.azurecr.io/<repository\_name>/<image\_name>

## Run containers - Azure Container Instances/AKS

need to authenticate before you can pull the image from your ACR

**For production environments, the recommended way to pull images is using service principals for authentication with the ACR.**

# Get the registry ID. You will need this ID for creating the authorization to the

# service principal

ACR\_ID = $(az acr show --name $ACR\_NAME --query id --output tsv)

#Get the ACR login server

ACR\_SERVER=$(az acr show --name $ACR\_NAME --query loginServer --output tsv)

#Get the service principal password. We will grant pull only privileges to the service

SP\_PASS = $(az ad sp create-for-rbac \

  --name http://$SP\_NAME \

  --scopes $ACR\_ID \

  --role acrpull \

  --query password \

  --output tsv)

#Get the App ID associated to the service principal

SP\_ID = $(az ad sp show --id http://$SP\_NAME --query appId --output tsv)

#Create the container in the Container Instance service

az container create

  --resource-group $RESOURCE\_GROUP \

  --name $APP\_NAME \

  --image $IMAGE\_TAG \

  --cpu 1 \

  --memory 1 \

  --registry-login-server $ACR\_SERVER \

  --registry-username $SP\_ID \

  --registry-password $SP\_PASS \

  --dns-name-label $APP\_DNS\_NAME \

  --ports 80

What is the most secure way of accessing the Azure Container Registry?

Azuire AD security principal (available ways: Azure AD login, or Admin account, or Azure AD service principals)

# Azure App Service – Web Apps

## Create Web App

The group of virtual machines that host your web application is managed by an App Service plan  
We can configure them with some high-level stuff.

App service plan is like a farm, multiple App services can run within it, sharing resources.

For production deployment, you should use Standard or, Premium pricing tiers.

**Deployments Slots:** Available from Standard pricing tier

* These slots are deployments of your web application that reside in the same App Service of your web application.

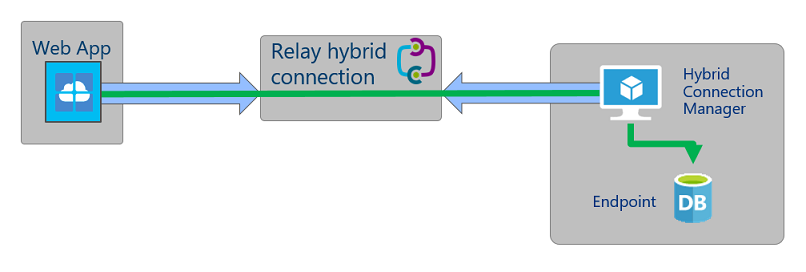
Ways to deploy:

* ZIP/WAR files (use Kudu)
* FTP
* Cloud synchronization (use Kudu, grab from OneDrive, Dropbox)
* CD (GitHub, BitBucket, Azure DevOps)
* Local Git repo (App Service is a remote repo, then Kudu built server compiles, deploys)
* ARM template

Kudu is the platform that is in charge of the Git deployments in Azure App Service.

**What if app require access to resources from on-premise infrastructure?**

* **VNet Integration:** Create site-to-site VPN with onpremise infrastructure, we can access those resources from App Service
* **Hybrid connections:** Depends on Azure Service Bus Relay, creates network connection between App Service and application endpoint (enable traffic between specific TCP host and port combos)

Hybrid Connections is both a **service in Azure** and a **feature in Azure App Service**.  


## Create WebJobs

You cannot run WebJobs in an App Service running Linux.

Execute background jobs (no user interaction) using the same resources from App Service Plan as the App Service itself.

1. Continuous (endless loop, can be remotely debugged)
2. Triggered (can’t debug, CRON expression)

**Turn on ‘Always On’ setting for the App Service, when using WebJobs**

.cmd, .bat, .exe, .ps1, .sh, .php, .py, .js, .jar, and zip fi les. If you use a zip fi le, you can only add supported fi le types to it

CRON expression: <seconds> <minutes> <hours> <days of month> <months> <days of week>  
Use asterisk if not needed, eg: 0 15 10 \* \* 6 means: 10:15 AM on 6th day of the week (Friday, watch out)

Every 10 seconds from 30th sec: **30/10**

**WebJobs .NET SDK**

* V2 does not support .NET Core, only V3
* V3 add Microsoft.Azure.WebJobs.Extensions.Storage

Concepts:

* **host**: Runtime container in which functions are executed (IHost)
* **trigger**
  + Automatic
  + Manual
* Binding: How job can interact with external world (input/output)

builder.ConfigureWebJobs(wj =>  
{  
wj.AddAzureStorageCoreServices();  
wj.AddAzureStorage();  
});

  public static void NewMessageQueueTrigger(

    [QueueTrigger("<put\_your\_queue\_name\_here>")] string message,

    ILogger logger)

   {

    logger.LogInformation($"New message from queue (<put\_your\_queue\_name\_here>):

   {message}");

   }

## Enable diagnostics logging

* **Web Server Diagnostics**
  + **Detailed Error Logging** (HTTP status code 400 or higher): HTML page for each error, 50 can be kept, then 26 oldest are deleted
    - **Failed Request Tracing:** IIS logs
    - **Web Server Logging:** W3C log format, info about the failed request
* **Application diagnostics:** System.Diagnostics.Trace (not App Insights)
* **Deployment diagnostics**

Levels of logging: Disabled, Information, Warning, Error, Verbose

Storing application logs in the fi le system is intended for debugging purposes (**gets automatically disabled after 12h**) **use blob storage instead**

Where are logs?

* Logs from app logging: **/LogFiles/Application/**
* Logs from Failed Request Tracing: **/LogFiles/W3SVC#########/**
* Logs from Detailed Error logging: **/LogFiles/DetailedErrors/**
* Web Server logs: **/LogFiles/http/RawLogs/**
* Deployment logs: **/LogFiles/Git**

**Fetch logs:** az webapp log download --resource-group --name

**Log Streams:** see log messages as they are being saved, in almost real time:  
az webapp log tail --resource-group --name

## Create Web App for containers

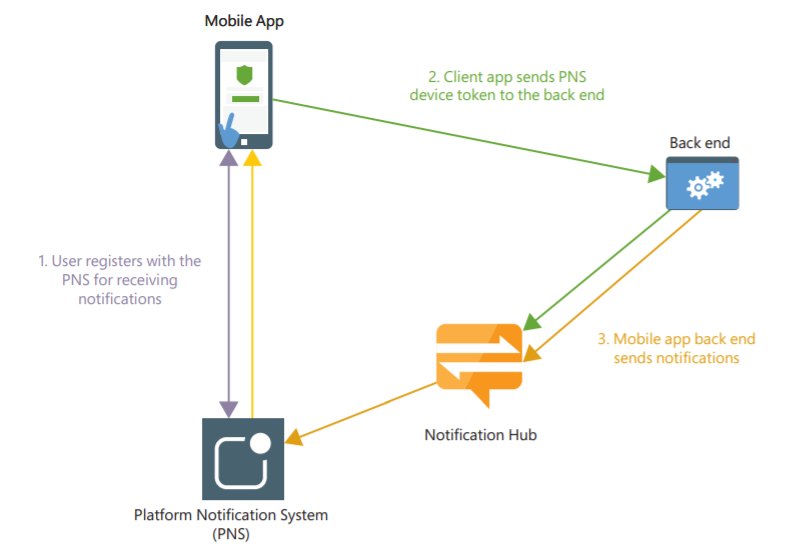
## Azure Monitor

# Azure App Service – Mobile Apps

## Add push notification to Mobile App

* **Mobile app client:** user must register it with Platform Notification System (PNS) to receive notifications
* **Mobile app backend:** stores PNS handler (with this, we can send notifications to clients)
* **Platform Notification System (PNS):** delivers the actual notification (each vendor has its own, Apple Push Notification Service, Google uses Firebase Cloud Messages, Windows Notification Service)

Device tokens need to be refreshed every time you release a new version of your app.  
Backend deals with a large amount of traffic and databases updates simply to keep device tokes updated (let alone if we decide to use multiple vendors…)



There is a possibility to use this client side with an SDK, but **removes all tags that we can associate with the device, for security purposes (eg. We can’t send to only ‘n’ group of users notifications)**

using Microsoft.Azure.NotificationHubs;

using Microsoft.Azure.Mobile.Server.Config;

// We need to get the configuration for sending the logs to the correct tracer.

HttpConfiguration config = this.Configuration;

// We get the mobile settings for getting the Notification Hub credentials.

MobileAppSettingsDictionary mobileSettings = this.Configuration

  .GetMobileAppSettingsProvider()

  .GetMobileAppSettings();

// Get the Notification Hubs name and connection string. We will use these for creating

// a Notification Hub client.

string notificationHubName = mobileSettings.NotificationHubName;

string notificationHubConnection = mobileSettings

  .Connections[MobileAppSettingsKeys.NotificationHubConnectionString]

  .ConnectionString;

// Create a new Notification Hub client that will perform all the interactions with the

// Notification Hub

NotificationHubClient hubClient = NotificationHubClient

  .CreateClientFromConnectionString(notificationHubConnection, notificationHubName);

// We want to send notifications to all registered templates that contains the "messageParam" parameter

// This includes templates for Apple, Google, Windows and Windows Phone PNS platforms

var templateParams = new Dictionary<string,string>();

templateParams["messageParam"] = item.Text + " was processed.";

try

{

  // Send the actual push notification.

  var result = await hubClient.SendTemplateNotificationAsync(templateParams);

  // We register the notification was sent succesfully.

  config.Services.GetTraceWriter().Info(result.State.ToString());

}

catch (System.Exception ex)

{

  // There were some issues that we need to register in the logs.

  config.Services.GetTraceWriter().Error(ex.Message, null, "Push.SendAsync Error");

}

## Enable offline sync

While your app is in offline mode, Mobile Apps SDKs allow your application to create, delete, or modify data.

* Saved to local store
* When there is network available, SDK syncs the data with backend
  + Conflicts can be handled on client side or backend

When you work with the Azure Mobile App SDK, your client uses the /tables endpoint for performing CRUD.

Online requests are managed by the **IMobileServiceTable** (.NET client SDK) or **MSTable** (iOS) interfaces.  
To implement offline sync in your application, you should use the “sync” version of these interfaces: **IMobileServiceSyncTable** and **MSSyncTable**.

The local storage is a data-persistence layer provided by the Azure Mobile App client SDK (SQLite or Cora Data for iOs).

IMobileServicesSyncContext.InitializeAsync(localstore)

## Implement remote instrumentation strategy for mobile devices

# Azure App Service – API Apps

## Create app

## Create documentation for the API by using open source and other tools

# Azure Functions

## Implement input/output bindings

## Implement triggers by using data operations, timers, webhooks

## Implement Azure Durable Functions

## Create Azure Function app by Visual Studio

## Implement Python Azure functions

# Storage Tables

## Design and implement policies

Query table storage by code

Implement partitioning schemes

# Cosmos DB

CRUD data by APIs

Implement partitioning schemes

Set appropriate consistency level for operations

# Relational DB

Provision and configure DBs

Configure elastic pools for Azure SQL DB

CRUD data by code

Provision and configure Azure SQL DB serverless instances

Provision and configure Azure SQL and Azure PostreSQL Hyperscale instances

# Blob Storage

Move items between storage accounts or containers

Set and retrieve properties, metadata

Implement blob leasing

Implement data archiving and retention

Implement Geo Zone Redundant storage

# Authentication

## Implement auth by certifications, form-based authentications, tokens

## Implement multifactor/Windows authentication via Azure AD

## Implement OAuth2 authentication

## Implement Managed identities/Service Principal authentication

## Implement Microsoft identity platform

# Access Control

## Implement CBAC authorization (Claims-Based Access Control)

## Implement RBAC authorization (Role-Based Access Control)

Grant users least rights (but sufficient enough)

* Users
* Service Principals (run by apps, not people)

Owner: Access to everything, can grant access to others  
Contributor: Access to everything, but can’t grant access to others  
Reader: read-only

## Create SAS (shared access signatures)

ACCESS KEYS:

Who have access key, has full access to resource: No need for portal account, etc

Key recycling: when pressing refresh/regenerate 🡪 invalidates old, generates new  
There’s two keys, so key recycling is possible without down time

SAS:

More restricted than Access Keys

* Allowed resource types: Service, Container, Object
* Expiry dates (start-ebd date)
* HTTPS only
* Allowed permissions: Read, Write, Delete, List, Add, Create, Update, Process
* Sign with Access Key
* Only way to invalidate SAS is to regenerate the access key, with which the SAS was signed with

# Implement secure data solution

## Encrypt/decrypt data at rest, and in transit

Storage account does encryption by default (can’t be turned off, but we can use our own keys [key URI or Key Vault]).  
SSE: Storage Service Encryption

This is encryption at rest, however, when an app tries to access something from the storage account, it gets decrypted and sent over the network as unencrypted file.

* Secure Transfer Required 🡪 in Configuration (must use HTTPS [doesn’t really work with own domains])

SQL DB: **server level**  
- Security: Transparent data encryption: MS creates own key and encrypts the DB Server by default  
- This can be turned off on a **database level** (why would we do that? Eg. client already does encrypt stuff and sends that to the DB)  
-There’s a hidden database ‘master’: generated by Azure, this cannot be encrypted (because keys used to encrypt user DBs are stored here)

Key vault: A1 standard (geo availability) or P1 Premium (HSM: hardware security solution, it generates keys)  
- Can be set for which Virtual Network can access it  
- Secrets have access URL, but it requires token

## CRUD keys, secrets and certificates via KeyVault API

# Develop code to support scalability of apps and services

Scale up: move to bigger/more expensive plan (or scale down, move down)  
Scale out: run app on three instances

## Implement autoscaling rules and patterns (schedule, operational/system metrics, singleton applications)

Autoscaling is only available on S1 and upwards (production grade) App Service Plans.

Virtual Machine Scale Set: group of unique set of VMs

* Low availability is supported
* Placement group (availability zone): VMs are distributed evenly (all of them won’t go down at the same time)
* Load Balanc

Single VM: can be resized, but disrupt will happen (stop and start)  
…or just create a new VM and redirect traffic from the old VM

## Implement code that handles transient faults

VM, NIC, DBs, etc: these are far away from each other, there’s more chance of timeouts, slowdowns, etc

Transient fault: network error, service slowdown, DB timeout, etc (temporary, one time error)

Eg: App is doing 5 operation, but in the middle of it, Azure decides that it’s time to scale down, so App gets disrupted after completing the 3rd operation:

* Retry/backoff logic: what happens with the completed 1,2,3 operations? 4, 5 are pending, but those are done (rollback/unprocess or save process? Put 4, 5 back to queue)
* Less coupling, use messaging and queue: DB reads from queue, app writes to queue

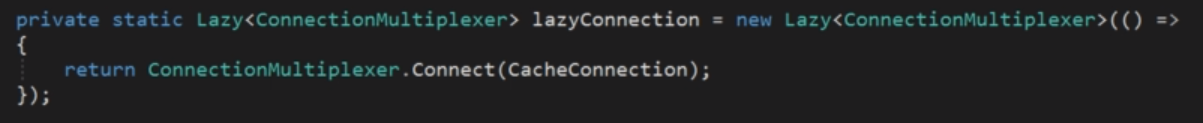
## Implement AKS scaling strategies

# Integrate caching and content delivery within solutions

## Store and retrieve data in Azure Redis cache

Backend type of cache

StackExchange Nuget package: Azure has a connection string for that:  

Then:  






## Develop code to implement CDNs in solutions

Frontend caching (content delivery network):

* sits infront of site
* takes static files, store them closer to the end user that the backend server

CDN Profile:

* runs global scale
* Pricing: three providers, same price:
  + Microsoft
  + Verizon: preload content to CDN, dynamic delivery
  + Akamai: medi optimization, no custom domains

CDN Endpoint: URL used by frontend

* **Origin type:** Storage, Cloud Service, Web App, Custom origin (onpremise)

## Invalidate cache content (Redis/CDN)

# Instrument solutions to support monitoring and logging

## Configure instrumentation in app or service by Application Insights

Azure Functions: Supports Application insights (Ilogger injected to Run(), log.LogInformation())

## Analyze and troubleshoot solutions by using Azure Monitor

In the VM: Monitoring -> Diagnostic settings -> Enable guest-level monitoring  
Azure Monitor collect host level metrics (CPU utilization, disk/network usage): no additional software required  
Guest level monitoring collects metrics, logs, etc using Azure Diagnostics agent

Uses a Storage Account

(but VM can send diagnostic data to Application Insights too)

## Implement Application Insights Web Test and Alerts

# Develop an App Service Logic App

## Create a Logic App

## Create a custom connector for Logic Apps

## Create a custom template for Logic Apps

# Integrate Azure Search

## Create an Azure Search index

## Import searchable data

## Query the Azure Search index

## Implement cognitive search

## Implement API management

## Create an APIM instance

## Configure authentication for APIs

## Define policies for APIs

# Develop event-based solutions

## Azure Event Grid

## Azure Notification Hubs

## Azure Event Hub

# Develop message-based solutions

## Azure Service Bus

## Azure Queue Storage queues