Microsoft

AZ-400T05 Module 01: Infrastructure and configuration Azure tools

Lesson 01: Infrastructure as Code and Configuration Management

Lesson Overview

- Environment Deployment
- Environment Configuration
- Modularization
- Imperative vs. Declarative
- Idempotence
- Technical Debt
- Configuration Drift
- Database as Code

Infrastructure as Code

- With Infrastructure as Code, you capture your environments in a text file (ie. ARM Template).
- Include any networks, servers, and other compute resources
- You can check the script or definition file into version control, and use it as the source for updating existing environments or creating new ones

Environment Deployment

Manual deployment

- Snowflake servers
- Deployment steps vary by environment
- More verification steps and more elaborate manual processes
- Increased documentation to account for differences
- Deployment on weekends to allow time to recover from errors
- Slower release cadence to minimize pain and long weekends

Infrastructure as code

- Consistent servers between environments
- Environments created or scaled easily
- Fully automate creation and updates of environments
- Transition to immutable infrastructure
- Use blue/green deployments
- Treat servers as cattle, not pets

Environment Configuration

Manual configuration

- Configuration bugs difficult to Identify
- Error prone
- More verification steps and more elaborate manual processes
- Increased documentation
- Deployment on weekends to allow time to recover from errors
- Slower release cadence to minimize requirement for long weekends

Configuration as code

- Bugs easily reproducible
- Consistent configuration
- Increase deployment cadence to reduce amount of incremental change
- Treat environment and configuration as executable documentation

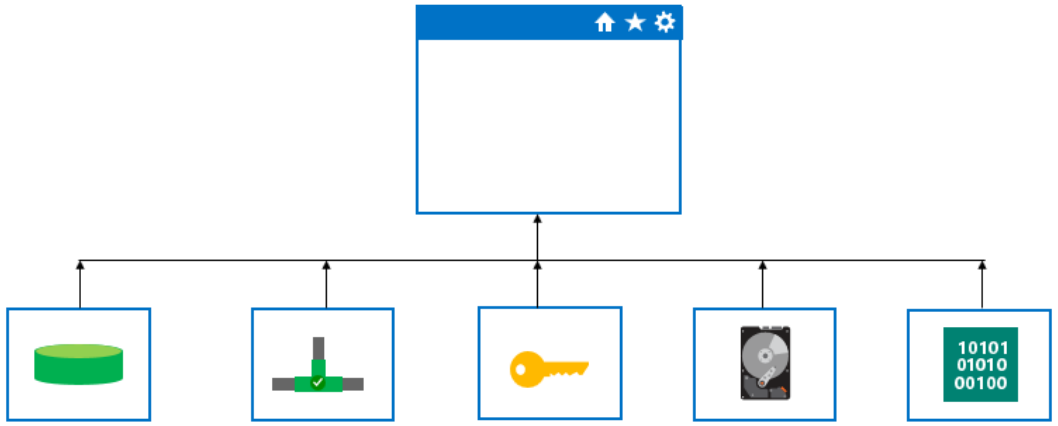
Modularization

Divide automation assets into logical parts like:

- Databases, networks, storage, security, OS, and other components

Benefits of modularization

- Reuse of components
- Manage and maintain your code
- Sub-divide work and responsibilities across teams
- Troubleshooting
- Extend and add to existing infrastructure definitions

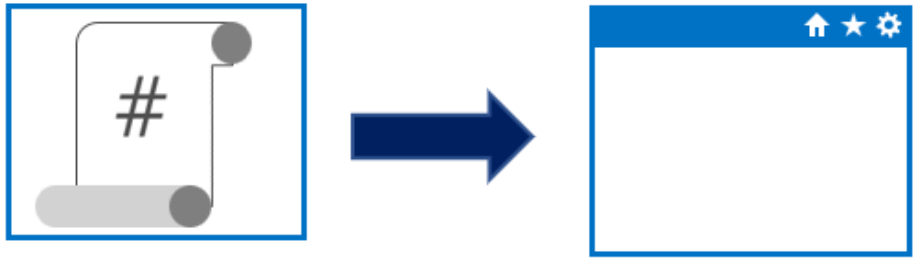


Imperative vs. Declarative

Approaches to implementing infrastructure and configuration as code

Declarative

- Functional
- Defines what the final state should be



Imperative

- Procedural
- Defines the “how” for what the final state should be

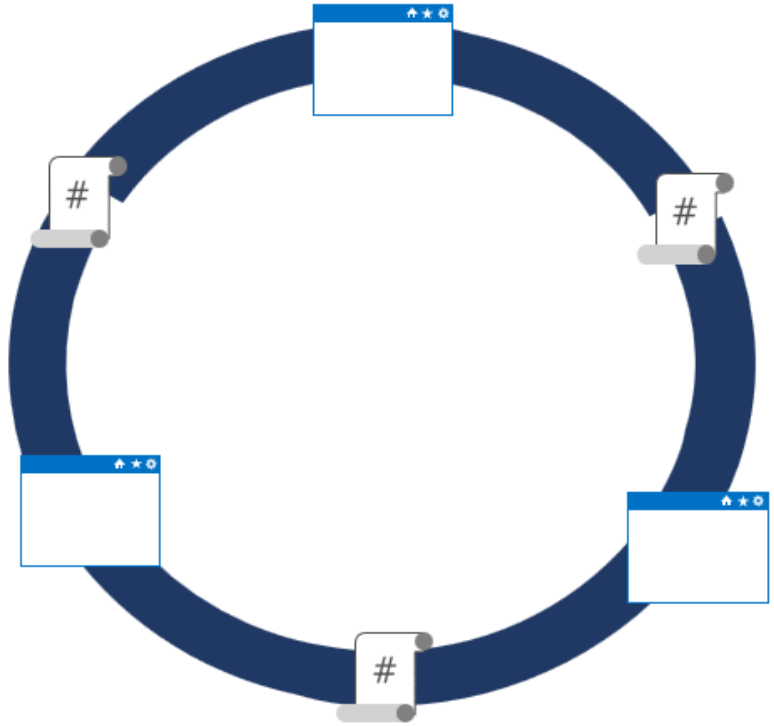
Idempotence

Idempotence – definition

- Mathematical term used in the context of infrastructure and configuration as code
- Ability to apply one or more operations against a resource, resulting in the same outcome

To attain idempotence

- Automatically configure and reconfigure an existing set of resources, or
- Discard existing resources and spin up a fresh environment



Technical Debt

Technical debt

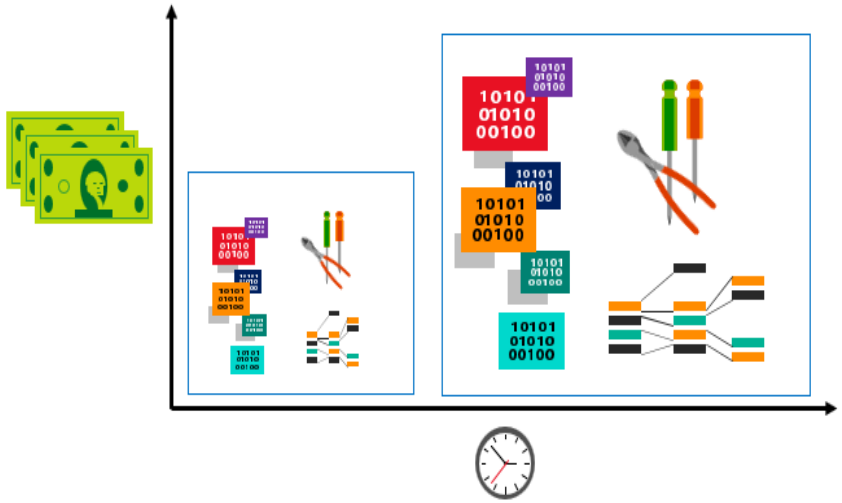
- Term to describe the future penalty incurred today by making easy or quick choices in software development practices
- Over time, accrues in same way that monetary debt does

Examples of technical debt

- Bugs
- Performance issues
- Operational issues
- Switching to technologies or versions not accounted for in your development process
- Updates to platform or services that you were not aware of or have not accounted for

Common sources of technical debt

- Lack of coding style and standards
  - Linting, Code Formatter
- Lack of, or poor design of, unit test cases
- Changes in project scope, such as deciding to localize and already release product
- Making manual changes to resources.
- Not writing self-documenting code
- Taking shortcuts to meet deadlines



Configuration Drift

- Process whereby a set of resources change their state over time
- Can occur from changes made by people, processes, or programs

Potential security risks introduced by configuration drift:

- Open ports that should have been closed
- Inconsistent patching across environments
- Software that doesn’t meet compliance requirements

Solutions that can help

- Windows PowerShell Desired State Configuration
- Azure Policy
- Many non-Microsoft solutions integrated with Azure

Database as Code

The database can be treated as code for higher efficiency and fewer delays

- Can be versioned like applications
- Bugs easily reproduced and configuration remains consistent

Benefits of database as code

- Make changes in smaller increments
- Configuration scripts can be treated as executables
- Verify schema changes between production and development
- Auditability

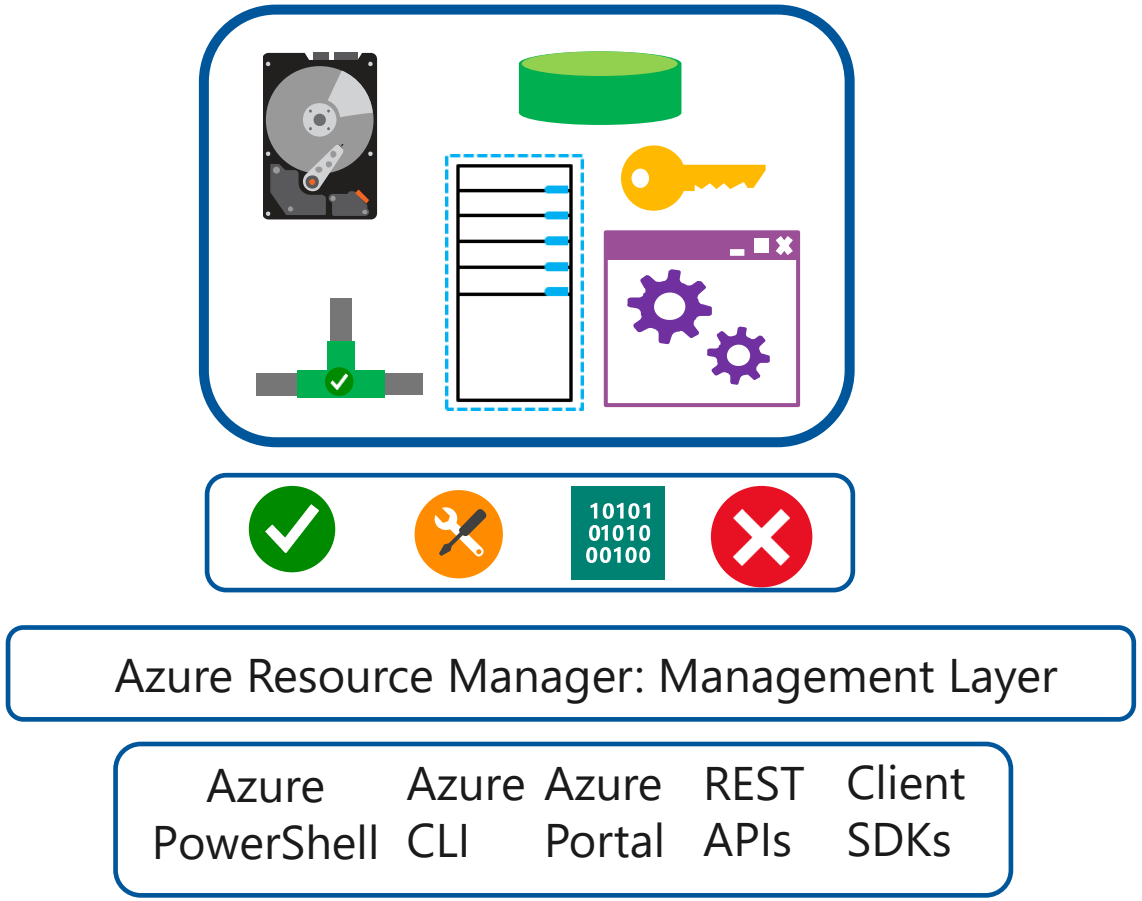
Lesson 02: Create Azure Resources using ARM Templates

Lesson Overview

- What is Azure Resource Manager?
- Azure Resource Manager templates
- Template components
- Quickstart templates
- Manage dependencies
- Modularize templates
- Managing secrets in templates
- Template extensions
- Subscription-level resources
- Why Use ARM templates?

What is Azure Resource Manager?

A management layer in which a resource group and all the resources within it are created, configured, managed, and deleted



Azure Resource Manager templates

Azure Resource Manager Templates

- Precisely define all the Resource Manager resources in a deployment
- Are written in JavaScript Object Notation (JSON) format
- Are Declarative in nature
- Allow you to consistently deploy a solution with confidence that resources are deployed in a consistent state
- Allow you to reuse an existing template across different environments such as dev, test and production
- Allow you to re-use existing templates and customize them to your specific needs

Why Use ARM templates?

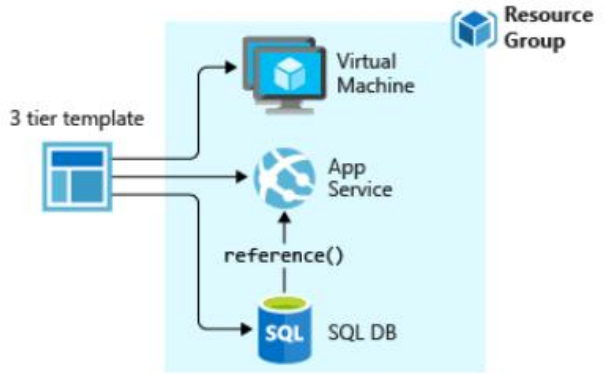
- Make deployments faster and more repeatable
- Improve consistency by providing a common language
- Enable you to deploy multiple resources in the correct order by mapping out resource dependencies
- Reduce manual, error-prone tasks
- Templates can be linked together to provide a modular solution

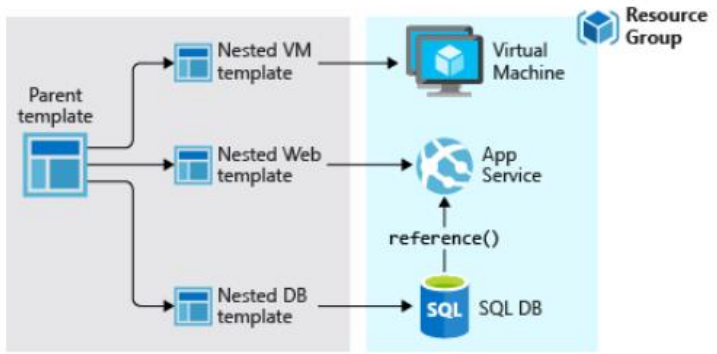
ARM Template Elements

ARM Templates written in JSON files can contain the following sections:

- Parameters
- Variables
- Functions
- Resources
- Outputs

- A Deployment can be done using:
  - A multi tier Template
  - A parent Template with nested Templates





Modularize templates

- Use linked templates to break the solution into individual pieces
- Reuse those elements across different deployments
- Nested Templates are also possible - Nesting done in "ressources"

```
"resources": [
  {
    "name": "linkedTemplate",
    "type": "Microsoft.Resources/deployments",
    "apiVersion": "2018-05-01",
    "properties": {
      "mode": "Incremental",
      "templateLink": {
        "uri":"https://linkedtemplateek1store.blob.core.windows.net/linkedtemplates/linkedStorageAccount.json?sv=2
        se=2018-12-31T14%3A32%3A29Z&sp=r"
      },
      "parameters": {
        "storageAccountName":{"value": "[variables('storageAccountName')]"},
        "location":{"value": "[parameters('location')]"}
      }
    }
  },
```

ARM Template components

```
"parameters": {
  "adminUsername": {
    "type": "string",
    "metadata": {
      "description": "Username for the Virtual Machine."
    }
  },
```

```
"variables": {
  "nicName": "myVMNic",
  "addressPrefix": "10.0.0.0/16",
  "subnetName": "Subnet",
  "subnetPrefix": "10.0.0.0/24",
  "publicIPAddressName": "myPublicIP",
  "virtualNetworkName": "MyVNET"
```

```
"functions": [
  {
    "namespace": "contoso",
    "members": {
      "uniqueName": {
        "parameters": [
          {
            "name": "namePrefix",
            "type": "string"
          }
        ],
        "output": {
          "type": "string",
          "value": "[concat(toLower(parameters('namePrefix')), uniqueS-
tring(resourceGroup().id))]"
        }
      }
    }
  }
],
```

Here's an example that creates a public IP address resource:

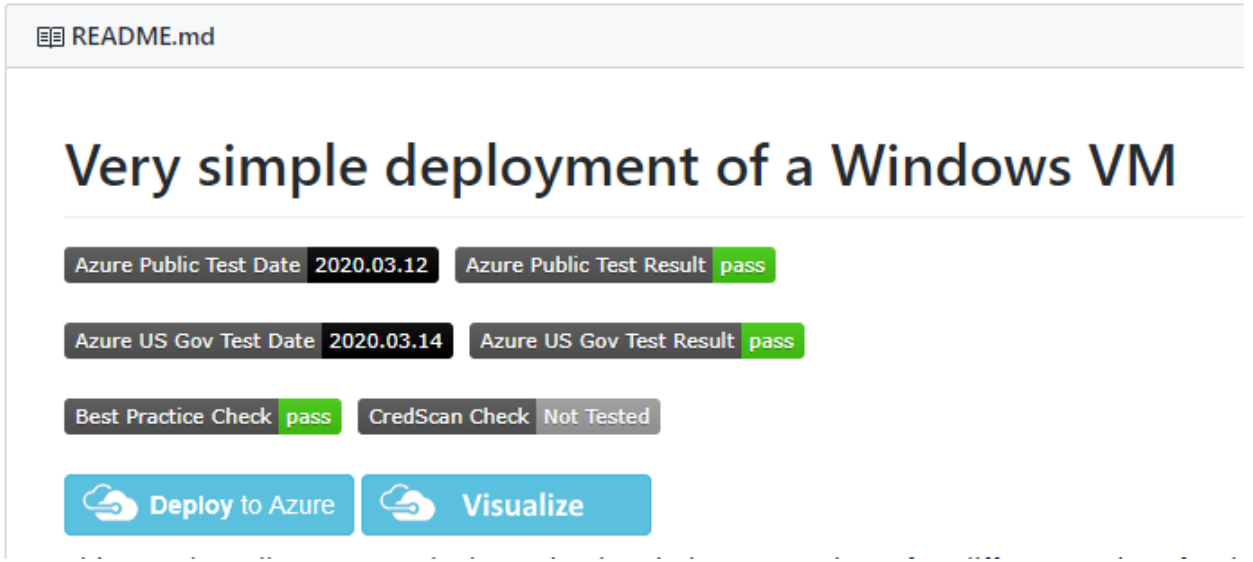
```
{
  "type": "Microsoft.Network/publicIPAddresses",
  "name": "[variables('publicIPAddressName')]",
  "location": "[parameters('location')]",
  "apiVersion": "2018-08-01",
  "properties": {
    "publicIPAllocationMethod": "Dynamic",
    "dnsSettings": {
      "domainNameLabel": "[parameters('dnsLabelPrefix')]"
    }
  }
}
```

Quickstart templates

Resource Manager templates provided by the Azure community

Behind the scenes it consists of:

- ARM Template
- Deploy Button
  - http://azuredeploy.net/
  - armdeploy.json
- Visualization
  - http://armviz.io/



Manage dependencies

Some resources will depend on other resources before you can deploy them

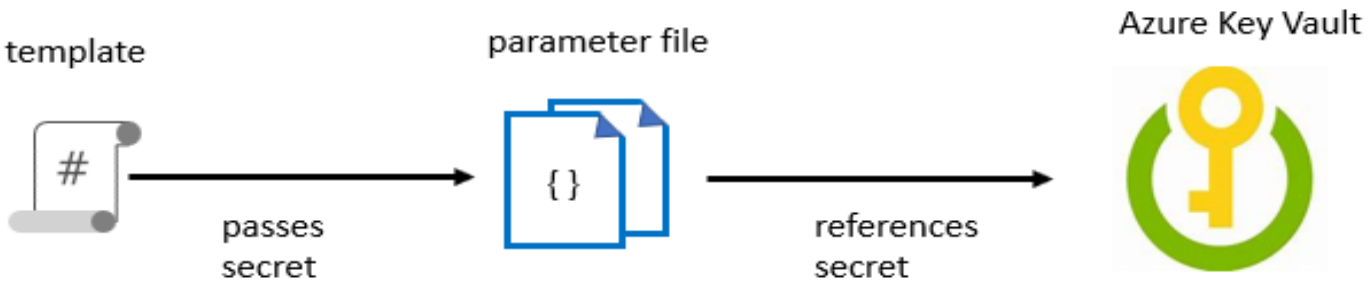
- Define this relationship by marking the dependency with the **dependsOn** element

```
129         "type": "Microsoft.Compute/virtualMachines",
130         "name": "[variables('vmName')]",
131         "location": "[parameters('location')]",
132         "apiVersion": "2018-10-01",
133         "dependsOn": [
134           "[resourceId('Microsoft.Storage/storageAccounts/', variables('storageAccountName'))]",
135           "[resourceId('Microsoft.Network/networkInterfaces/', variables('nicName'))]"
136         ],
```

Managing secrets in templates

When passing a secure value (e.g. a password) as a parameter during deployment

- Create a key vault and secret using Azure CLI or PowerShell
- Set the key vault property enabledForTemplateDeployment to **true**
- Reference the key pair in the parameter file, **not** the template
- Enable access to the secret. **Owner** and **Contributor** roles grant access
- Deploy the template and pass in the parameter file



ARM Template using KeyVault

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/
deploymentParameters.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "adminLogin": {
            "value": "exampleadmin"
        },
        "adminPassword": {
            "reference": {
              "keyVault": {
                "id": "/subscriptions/<subscription-id>/resourceGroups/
examplegroup/providers/Microsoft.KeyVault/vaults/<vault-name>"
              },
              "secretName": "examplesecret"
            }
        },
        "sqlServerName": {
            "value": "<your-server-name>"
        }
    }
}
```

Template extensions

Use the Custom Script Extension (CSE) to download and run scripts on your Azure VMs

- Store scripts in Azure storage or in a public repository, such as GitHub
- Run manually or as part of an automated deployment

Implementing a Custom Script Extension

- Start with an Azure Quickstart template that does most of what you need and adapt it
- Consult the reference documentation for the required properties
- Remove parameters not required and add any additional parameters still needed
- Specify dependent resources

Lesson 03: Create Azure Resources using Azure CLI

Lesson Overview

- What is Azure CLI?
- Walkthrough-Installing Azure CLI
- Working with Azure CLI
- Walkthrough-Run Templates using Azure CLI
- Extensions

What is Azure CLI?

- Command-line program to connect to Azure
- Execute administrative commands on Azure resources through a terminal, command-line prompt, or script, instead of a web browser
  - For example, to restart a VM use the command:

    ```
    az vm restart -g MyResourceGroup -n MyVm
    ```

  - Can be installed on Linux, Mac, or Windows computers
- Can be use interactively or scripted
  - *Interactive*: Issue commands directly at the shell prompt
  - ***Scripted:*** *A*ssemble the CLI commands into a shell script and then execute the script

Walkthrough-Installing Azure CLI

Method for Installing Azure CLI depends on the platform

- Linux
  - For Ubuntu Linux install the Azure CLI using the **Advanced Packaging Tool (apt)** and the Bash command line
  - Package managers are dependent on the Linux version and distribution
- MacOS
  - Use the **Homebrew** package manager
- Windows
  - Install the Azure CLI on Windows using the **MSI installer** or using **Chocolatey** package manager for windows.

Working with Azure CLI

Commands in the CLI are structured in *groups* and *subgroups*

- Example: storage group contains subgroups like account, blob, storage, and queue
- Use az find to find commands you need

```
az find -q blob
```

- Use the - -help argument to get more detail about the command

```
az storage blob --help
```

Creating a new Azure resource typically involves the following process:

Manage Templates using Azure CLI

Done using: az group deployment

az group deployment create -g az-400 --template-file "azuredeploy.json" --parameters storageAcctName="classTest"

az group deployment list -g az-400

Commands

az group deployment create	Start a deployment.
az group deployment delete	Deletes a deployment from the deployment history.
az group deployment export	Export the template used for a deployment.
az group deployment list	Get all the deployments for a resource group.
az group deployment operation	Manage deployment operations.
az group deployment operation list	Gets all deployments operations for a deployment.
az group deployment operation show	Get a deployment's operation.
az group deployment show	Gets a deployment.

Run Templates using Azure CLI

Process to deploy and verify a sample Azure deployment template with custom script extension using Azure CLI

1. Create a resource group to deploy your resources to
2. Run **`curl`** to download the GitHub template
3. Validate the template
4. Deploy the resource
5. Obtain the IP address
6. Run **`curl`** to access your web server and verify the successful deployment and running of the custom script extension

Extensions

- With extensions, you gain access to experimental and pre-release commands along with the ability to write your own Azure CLI interfaces

- Find Azure CLI extensions

```
az extension list-available --output table
```

- Install or uninstall Azure CLI extensions

```
az extension add --name <extension-name>
```

```
az extension remove --name <extension-name>
```

- Run VM extensions using Azure CLI

```
az vm extension <sub-command > <parameter>
```

Lesson 04: Create Azure Resources by using Azure PowerShell

Lesson Overview

- PowerShell components
- What is Azure PowerShell?
- Installing Azure PowerShell on different platforms
- Walkthrough-Install Azure PowerShell
- Walkthrough-Run templates with PowerShell
- VM extensions with PowerShell

PowerShell Components

- PowerShell provides commands known as cmdlets
  - Intended to be small, single purpose functions
  - Performs tasks ranging from OS administration, working with ftp, file access, and much more
- Module added to Windows PowerShell or PowerShell Core
  - Connect to your Azure subscription and manage resources
  - Adds the Azure-specific commands
- Use Get-Help to display the help file for any cmdlet

```
Get-Help Get-ChildItem -detailed
```

- Cmdlets are packaged into Modules that you can add to your PowerShell installation
- Get a list of all currently loaded modules by typing Get-Module

Create an Azure VM by using PowerShell

```
Connect-AzAccount

New-AzResourceGroup -Name myResourceGroup -Location EastUS

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

Walkthrough-Install Azure PowerShell

Install the Az module:

- Shortened Az cmdlet prefix replaces the –AzureRm format
- Modules available from the PowerShell gallery, and require an elevated PowerShell prompt to install

```
Install-Module -Name Az -AllowClobber
```

- Relies on the NuGet package manager to retrieve components

Installing Azure PowerShell on Different Platforms

Two components allow PowerShell to configure and manage Azure resources, regardless of platform

1. Base PowerShell product (Windows PowerShell or PowerShell Core)
2. Azure PowerShell module

- Linux and macOS:
  - Install **PowerShell Core** version on Linux and macOS
  - Linux package manager depends on Linux distribution chosen
  - Like Azure CLI, macOS system uses **Homebrew** package manager
- Windows:
  - PowerShell is included in Windows Server and client by default.
  - Can check the installed version, with the below command

```
$PSVersionTable.PSVersion
```

VM Extensions with PowerShell

- Include Azure VM extensions with ARM templates for making configuration changes on already deployed VMs
- To see a list of individual extensions:

```
Get-Command Set-*Extension* -Module Az.Compute
```

- Example of a Custom Script Extension

```
Set-AzureRmVMCustomScriptExtension `
-ResourceGroupName "ResourceGroup11" `
-Location “Central US" -VMName "VirtualMachine07“`
-Name "ContosoTest" –TypeHandlerVersion "1.1" `
-StorageAccountName "Contoso"    `
-StorageAccountKey <StorageKey> `
-FileName "ContosoScript.exe"
-ContainerName "Scripts"
```

Lesson 05: Additional Automation Tools

Lesson Overview

- Azure SDKs
- Azure REST APIs
- Azure Cloud Shell
- Package Management

Azure SDKs

Install language SDKs for developing with Azure

Examples of available SDKs

- .NET SDK
- Java SDK
- Node SDK
- Python SDK
- PHP SDK
- Go SDK

For a complete list, see Azure Unified SDKs got to:

https://azure.microsoft.com/en-gb/downloads/

Azure REST APIs

REST API request/response components

- Request URI

  ```
  {URI-scheme} :// {URI-host} / {resource-path} ? {query-string}
  ```

- HTTP request message header fields: **GET**, **HEAD**, **PUT**, **POST**, and **PATCH**
- Optional additional header fields, for example an Authorization header that provides a bearer token to authorize a client request
- HTTP response message header fields: HTTP status codes indicating success or error
- Optional HTTP response message body fields: a MIME-encoded response return, such as a GET method returning data
- Response objects typically returned as JSON or XML

Azure Cloud Shell

- Cloud Shell hosted in the cloud and accessible via web browser and multiple access points including:
  - https://shell.azure.com
  - Directly via the Azure Portal,
  - Azure mobile app,
  - Visual Studio Code - Azure Account extension,
- Authenticates automatically and securely with Azure
- Can choose between Bash or PowerShell shell
- Contains integrated script editor
- Requires the creation of a storage account

Package Management

- *Package managers* allow you to manage all aspects of software such as installation, configuration, upgrade, and uninstall.
- Various package management solutions are available depending on the environment
  - **apt:** for Debian Linux
  - **yum**: for CentOS Linux
  - **chocolatey**: for Windows (built on PowerShell)

Walkthrough - Install Chocolatey

- https://chocolatey.org/install

Microsoft

AZ-400T05
Module 02: Azure Automation

Lesson 01: Azure Automation

Lesson Overview

- What is Azure automation
- Automation accounts
- Automation shared resources
- What is a runbook
- Automating processes with runbooks
- Runbook gallery
- Webhooks
- Source control integration
- Update management
- PowerShell workflows
- Creating a workflow
- Walkthrough-Create and run a workflow runbook
- Checkpoint and parallel processing

What is Azure Automation?

- An Automation service integrated with Microsoft Azure for automating the creation, deployment, monitoring and maintenance of Azure resources and resources external to Azure
- Integration with Microsoft Azure remove some complexity of automating in Azure

Process Automation
Orchestrate processes using graphical, PowerShell, and Python runbooks

Configuration Management
Collect inventory
Track changes
Configure desired state

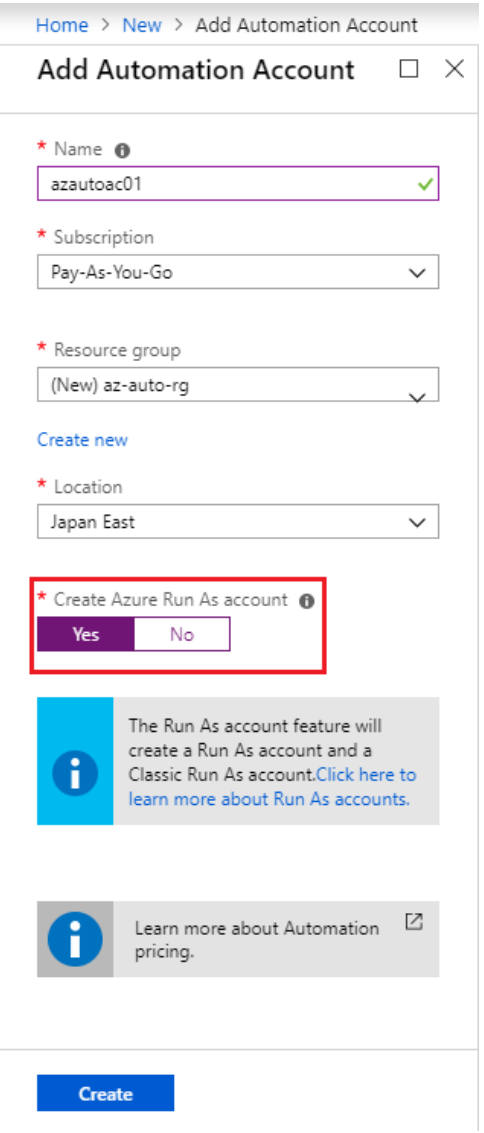
Update Management
Assess compliance
Schedule update installation

Shared capabilities
Role based access control
Secure, global store for variables, credentials, certificates, connections
Flexible scheduling
Shared modules
Source control support
Auditing
Tags

Heterogenous
Windows & Linux
Azure and on-premises

Automation Account

- To use Azure Automation you must create an Azure Automation account
- Acts as a container in which you store, manage and use automation artifacts
- Provides a way to separate your environments or further organize your Automation workflows and resources.
- Requires subscription-owner level access as provides access to all Azure resources via an API
- Need at least 1 but should have multiple for access control

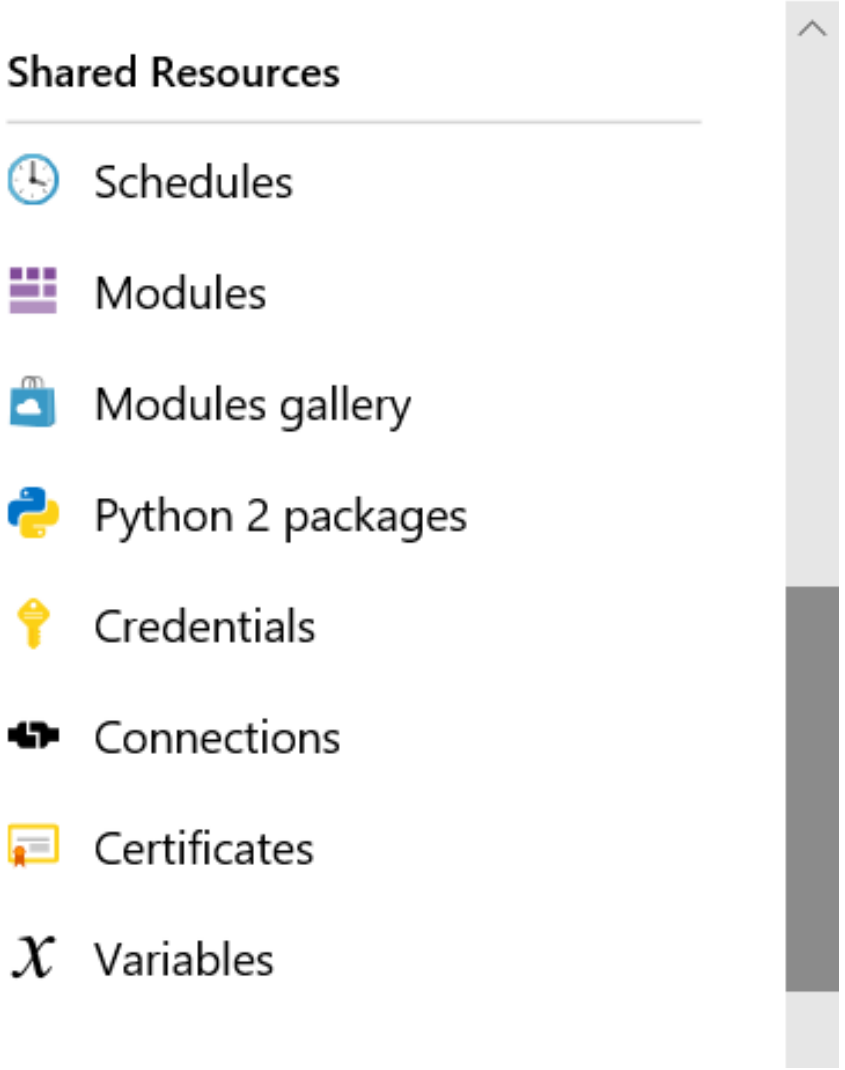


Run As account

- Can create when creating Azure Automation account
- Creates a service principal user in Azure AD which allows access to Azure resources when running automation

Automation shared resources

- Azure Automation contains shared resources that are globally available to be used in, or associated with a runbook
- Currently Eight Categories:
  - Schedules
  - Modules
  - Modules gallery
  - Python 2 packages.
  - Credentials
  - Connections
  - Certificates
  - Variables



What is a Runbook?

- A set of tasks that perform some automated process in Azure Automation
- Runbooks serve as repositories for your custom scripts and workflows
- Can create your own or import and modify from community via Runbook Gallery

Runbook Types available:

- Graphical runbook
- PowerShell runbooks
- PowerShell Workflow runbooks
- Python runbooks

Automating processes with runbooks

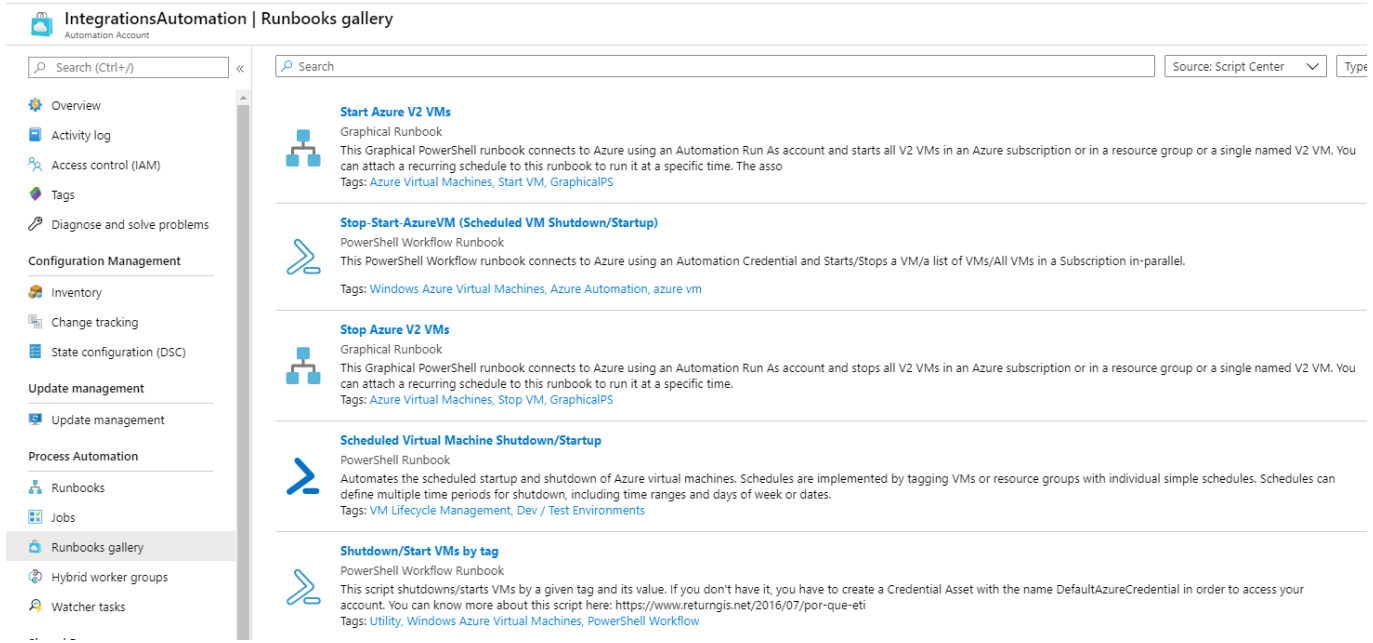
- Runbooks in Azure Automation can do anything that PowerShell can do because they are based on Windows PowerShell or Windows PowerShell Workflow and Python2

Example Usage Scenarios:

- Automating resources in your on-premises datacentre
- Start/Stop Azure virtual machines during off-hours solution in Azure Automation
- Auto-stop VMs based on low CPU usage, to optimize VM costs.
- Automate provisioning of a virtual machine in Amazon Web Services (AWS)
- Send your Automation logs to Log Analytics for analysis and have alerts such as email configured to notify you of any failures or other relevant outcomes.

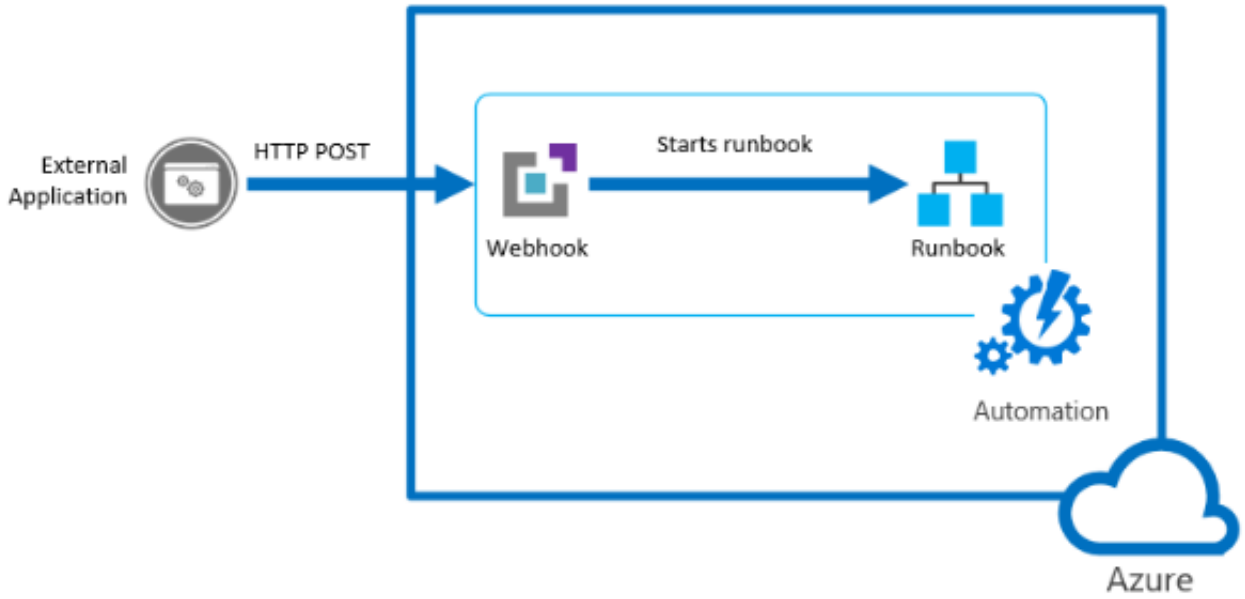
Runbook gallery

- Can import pre-existing runbooks from the runbook gallery at the **Microsoft Script Center**
- Runbooks provided to help eliminate the time it takes to build custom solutions
- Already been built by Microsoft and the Microsoft community
- Can review the code or a visualization of the runbook code on the gallery as well as see source projects, rating etc



Webhooks

- Automate the process of starting a runbook either by scheduling it, or by using a *webhook*.
- Uses a HTTPS request to start a runbook
- Reduces complexity and allows external services such as Azure DevOps, GitHub, or custom applications to use webhooks



- Webhook Syntax: http://< Webhook Server >/token?=< Token Value >

Update management

- A service built-in to Azure Automation
- Can be used to assess and manage operating system updates
- Applicable to Windows or Linux environments in Azure, On-premises, or other clouds providers.
- Onboard machines in multiple Azure subscriptions in the same tenant.

Components used:

- Microsoft Monitoring Agent (MMA) for Windows or Operations Management Suite (OMS) Linux Agent
- PowerShell DSC for Linux
- Automation Hybrid Runbook Worker
- Microsoft Update or Windows Server Update Services (WSUS) for computers running Windows operating systems or yum/apt/zypper package management services available on Linux.

PowerShell workflows

- Allows automation and orchestration of multi-environment tasks
- Built on PowerShell and based on Windows Workflow Foundation,

Characteristics:

- Contain *Activities* – which are core a component of a workflow, are a specific task in a workflow
- Tasks can be run in parallel
- Can be long-running and repeated over and over (idempotent)
- Be interrupted—can be stopped and restarted, suspended and resumed.
- Continue after an unexpected interruption, such as a network outage or computer/server restart.

Creating a workflow

- Can create with any script editor such as the Windows PowerShell Integrated Scripting Environment (ISE)
- There are syntax differences between PowerShell scrips and Workflows
- Requires keyword *workflow* to identify a workflow command
- Add parameter values using keyword *Param*

```
workflow MyFirstRunbook-Workflow

{
Param(
    [string]$VMName,
    [string]$ResourceGroupName
    )
    ....
    Start-AzureRmVM -Name $VMName -ResourceGroupName $ResourceGroupName
}
```

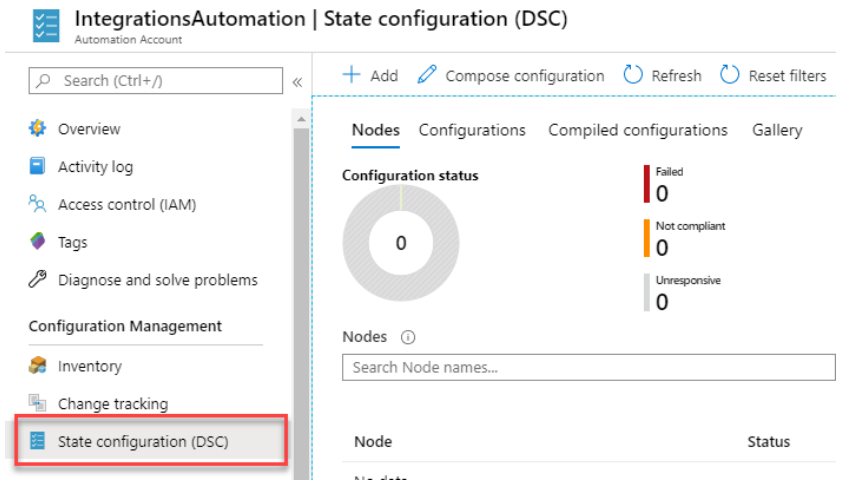
Lesson 02: Azure Automation State Configuration (DSC)

Lesson Overview

- Desired State Configuration (DSC)
- Azure Automation State configuration (DSC)
- DSC configuration file
- Walkthrough-Import and Compile
- Walkthrough-Onboarding machines for management
- Hybrid management
- DSC and Linux automation on Azure

Azure Automation State configuration (DSC)

- **Built-in pull server:** Built-in pull server in Azure Automation eliminates the need to set up and maintain your own pull server.
- **Management of all DSC artifacts:** Manage all DSC configurations, resources, and target nodes in single instance in the Azure portal or PowerShell
- **Ability to Import Reporting Data directly into Azure Log Analytics:** Can send this data to your Log Analytics workspace

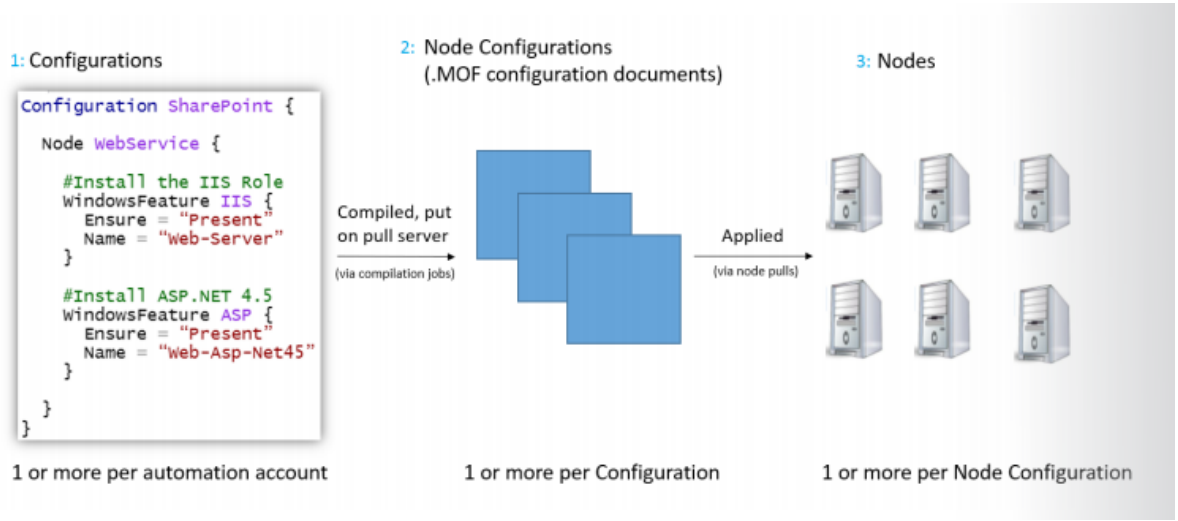


DSC Configuration File

- *Configurations* are Windows PowerShell scripts that define a special type of function.

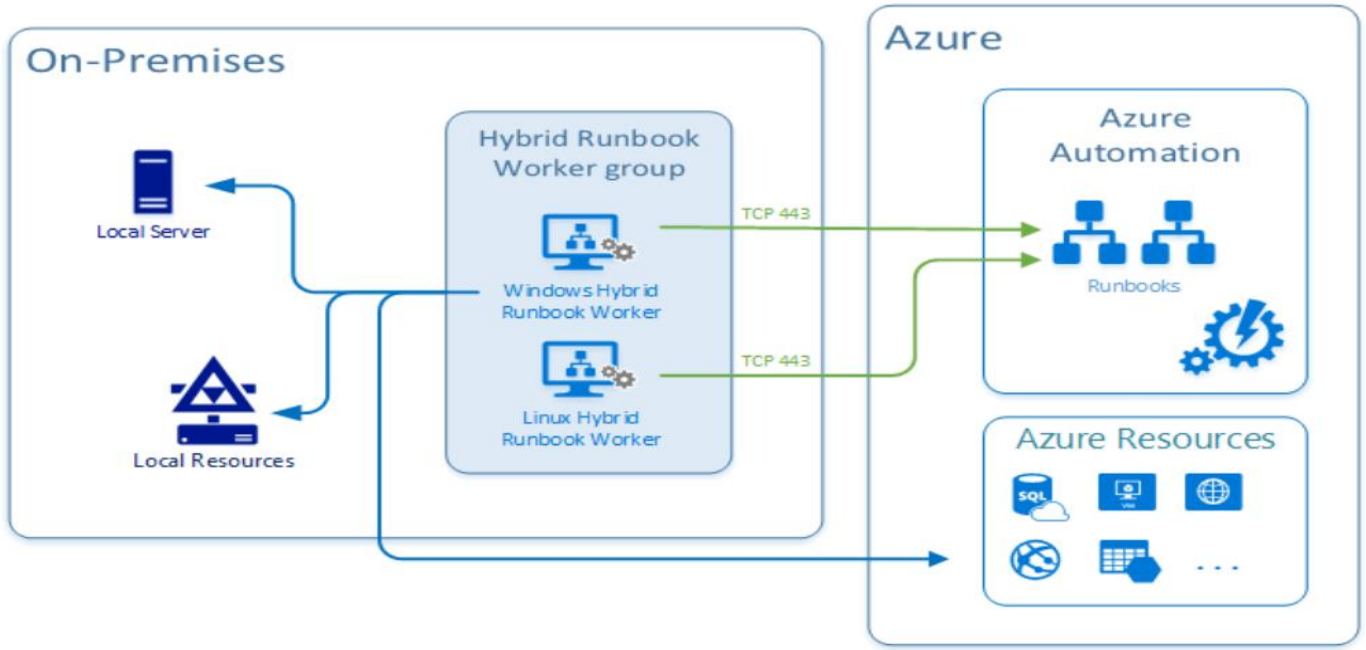
Config File Elements:

- Configuration block: Name of the configuration
- Node block: Define nodes being configured i.e. VMs and servers
- Resource blocks: Defines the actual configuration state for the nodes



Hybrid Management

- *Hybrid Runbook Worker* feature of Azure Automation allows you to run runbooks that manage local resources in your private datacenter, on machines located in your datacenter.



DSC and Linux Automation on Azure

- The following Linux operating system versions are currently supported by both PowerShell DSC and Azure Automation DSC

- CentOS 5, 6, and 7 (x86/x64)
- Debian GNU/Linux 6, 7 and 8 (x86/x64)
- Oracle Linux 5, 6 and 7 (x86/x64)
- Red Hat Enterprise Linux Server 5, 6 and 7 (x86/x64)
- SUSE Linux Enterprise Server 10, 11 and 12 (x86/x64)
- Ubuntu Server 12.04 LTS, 14.04 LTS and 16.04 LTS (x86/x64)

Configure DSC for Linux

- Install Open Management Infrastructure (OMI)
- Install DSC
- Create MOF document
- Push the configuration to the Linux computer

Microsoft

AZ-400T05
Module 03: Azure Compute Services

Lesson 01: Infrastructure as a Service

Lesson Overview

- Azure Virtual Machines
- Scaling Azure VMs
- Walkthrough-Create a Virtual Machine Scale Set
- Availability
- Additional IaaS considerations
- VMs versus containers
- Windows Server and Hyper-V containers
- Azure VMs and Containers
- Automating IaaS Infrastructure
- Azure DevTest Labs

Azure Virtual Machines

Azure VMs provide the benefits of virtualization, but must still be deployed, configured, and maintained

Operating system support for Windows and Linux OS deployments

Windows

- Windows Server and Windows client images available in Azure Marketplace for development test and production
- Images identified by publisher, offer, SKU, and version
- Only 64-bit operating systems supported

Linux

- Endorsed distributions available in Azure Marketplace
- Additional Linux partner-products, such as Docker, Jenkins

Azure Virtual Machines (*Cont.*)

Azure VMs usage scenarios

- Convenient way to create and test specific computer configurations
- Run applications in the cloud to meet changing demand
- Extend an existing datacenter

Deployment, configuration management, and extensions

- ARM templates and Windows PowerShell DSC
- Third party examples, such as Chef, Puppet, Ansible, and Terraform

Azure VMs service limits – Azure Resource Manager

- Each service has its own service limits, quotas, and constraints.
- Example: VM scale sets include a default of 200 VMs per availability set

Scaling Azure VMs

Use virtual machine scale sets (vmss) for managing a group of identical, load balanced VMs

Benefits of scale sets

- Create and manage multiple VMs at large scale
- Provide high availability and application resiliency
- Automatically scale as resource demand fluctuates

Options to configure VMs deployed in a scale set

- Use extensions to configure a VM post deployment
- Deploy a managed disk with a custom disk image

Differences between VMs and scale sets

- Scale sets provide automated versus manual creation, configuration, and integration for different scenarios

Custom Script Extension

- The Custom Script Extension downloads and executes scripts on Azure virtual machines
- Useful for post deployment configuration, software installation, or any other configuration or management tasks
- Deployment:
  - Template | PowerShell

```
Set-AzVMCustomScriptExtension -ResourceGroupName <resourceGroupName> `
    -VMName <vmName> `
    -Location myLocation `
    -FileUri <fileUrl> `
    -Run 'myScript.ps1' `
    -Name DemoScriptExtension
```

Azure DevTest Labs

- Creates environments consisting of pre-configured bases or Azure Resource Manager templates
- Enables developers on teams to efficiently self-manage virtual machines (VMs) and PaaS resources without waiting for approvals.
- Quickly provision Windows and Linux environments by using reusable templates and artifacts

Usage Scenarios:

- Dev or Test environments
- Integrate with CI/CD pipeline for automated tests and deleting images afterwards
- Scale up your load testing by provisioning multiple test agents and create pre-provisioned environments for training and demos

Lesson 02: Platform as a Service

Lesson Overview

- Azure App Service
- App Service Plans
- Walkthrough-Create a Java App in App Service on Linux
- Walkthrough-Deploy a .NET Core-based App
- Scale App Services
- Web App for containers
- Walkthrough-Deploy a Custom Docker image to Web App for containers
- Azure Container Instances
- Walkthrough-Create a container on ACI

Azure App Service

Fully managed environment enabling high productivity development

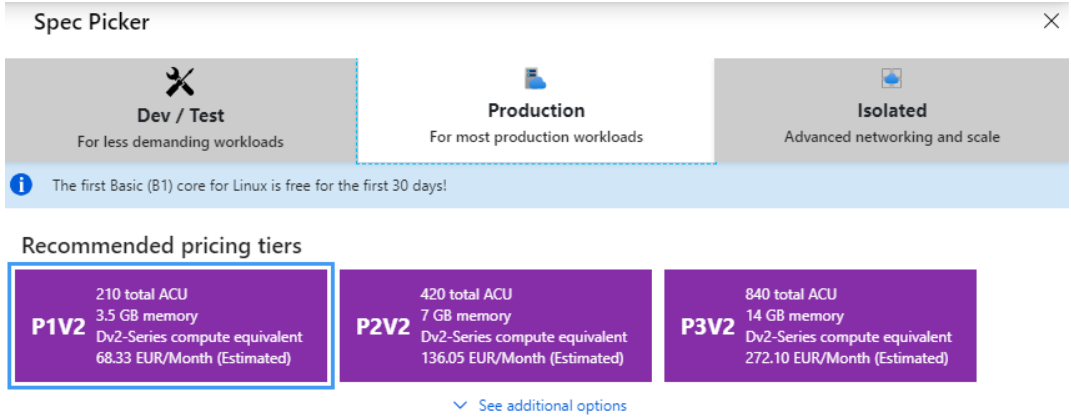
- Platform-as-a-service (PaaS) offering for building and deploying highly available and scalable cloud apps for web and mobile
- Developer productivity using .NET, .NET Core, Java, Python and a host of others
- Pay only for the compute resources used
- Azure platform handles infrastructure so that developers focus on core web apps and services
- Azure provides enterprise-grade security and compliance. Azure Active Directory

App Service Plans

- Define a set of compute resources for a web app to run
  - Practically it is the VM your AppService will be running on
- One or more apps can be configured to run in the same App Service plan
- App Service plans define:
  - Region where compute resources will be created (West US, East US, etc.)
  - Number of virtual machine instances (scale count.)
  - Instance size (Small, Medium, Large)
  - Pricing tier (next slide)

App Service Plans – Pricing Tiers

- **Shared compute.** Free and Shared. Runs an app on the same Azure VM as other App Service apps, and the resources cannot scale out
- **Dedicated compute**. The Basic, Standard, Premium, and PremiumV2 tiers run apps in the same plan in dedicated Azure VMs
- **Isolated**. Dedicated Azure VMs on dedicated Azure Virtual Networks
- **Consumption**. Only available to function apps



Scale App Services

Scaling ensures the right amount of resources to manage an application's needs

- **Scale up**: add additional resources (CPU, memory, etc.)
- **Scale out**: increase the number of VM instances

Configure Autoscale to manage fluctuating load of an application

- **Metric**: e.g., scale in or out based on a percentage CPU value
- **Schedule**: e.g., scale down on weekends or scale up for increased holiday demand

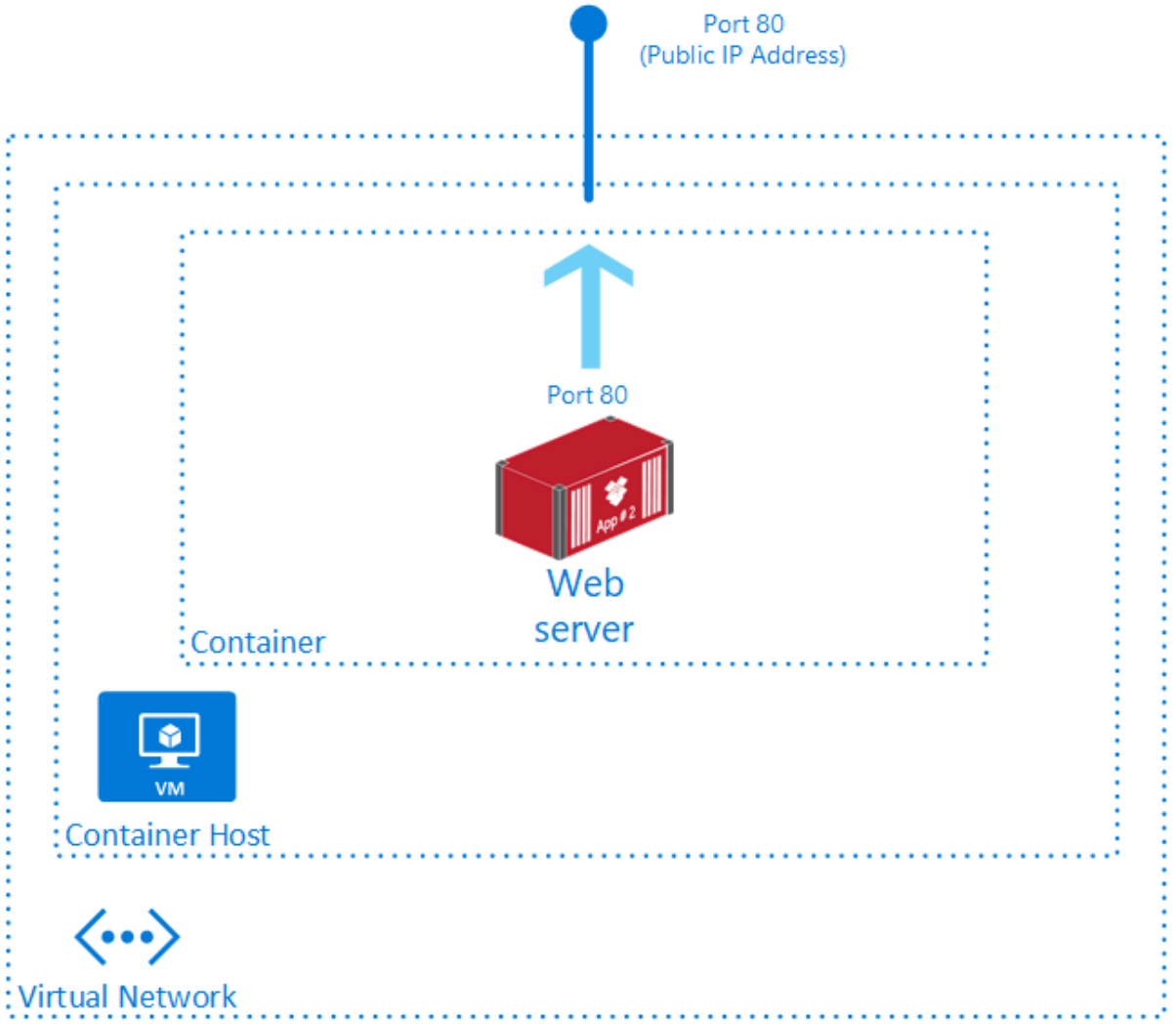
Configure Autoscale profiles: regular, fixed date, and recurrence

Web App for containers

- Web App for Containers is Azure's service container hosting service
- Enables developers to deploy update and scale their container apps much faster
- Platform automatically takes care of OS patching, capacity provisioning, and load balancing
- Integrated continuous integration/continuous delivery capabilities
- Rich configuration features including: add custom domains, Azure AD integration, add SSL certificates, and more
- Ideal environment to run web apps that don't' require extensive infrastructure control
- Supports Linux and Windows

Azure Container Instances

- PaaS service for running both Linux and Windows containers
- Run as single containers, providing isolation for individual containers
- Isolation achieved by using Hyper-V containers
- Fast way to run a container in Azure without the need to own or provision a VM



Azure Container Instances (*Cont.*)

Public IP connectivity and DNS name

- Expose your containers directly to the internet with an IP address and a fully qualified domain name (FQDN).

Custom sizes and resources

- Specify exact specifications of CPU cores and memory
- Fine tune cost based on what's used

Virtual network deployment (preview)

- Deploy container instance into a subnet in the Azure Virtual Network (VNET) for secure communications with other resources in the VNet

Persistent Storage

- Retrieve and persist state with Azure Container Instances by using Azure Files shares

Lesson 04: Azure Kubernetes Service

Lesson Overview

- Kubernetes overview
- Azure Kubernetes Service
- AKS Architectural components
- Kubernetes networking
- Deployment
- Walkthrough-Deploying and connecting to an AKS Cluster
- Continuous deployment
- Updating images

Kubernetes overview

Cluster orchestration technology (originated from Google)

Open-source system for automating deployment, scaling, and management of containerized applications

Kubernetes can be thought of as:

- A container platform
- A microservices platform
- A portable cloud platform (and more)

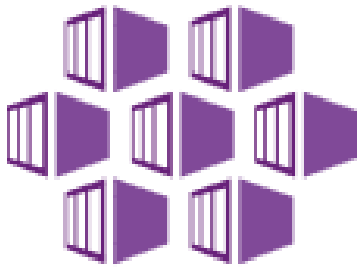
Several other container cluster orchestration technologies are available, including:

- Docker Swarm
- Mesosphere DC/OS

Azure Kubernetes Service

AKS is Microsoft's implementation of Kubernetes

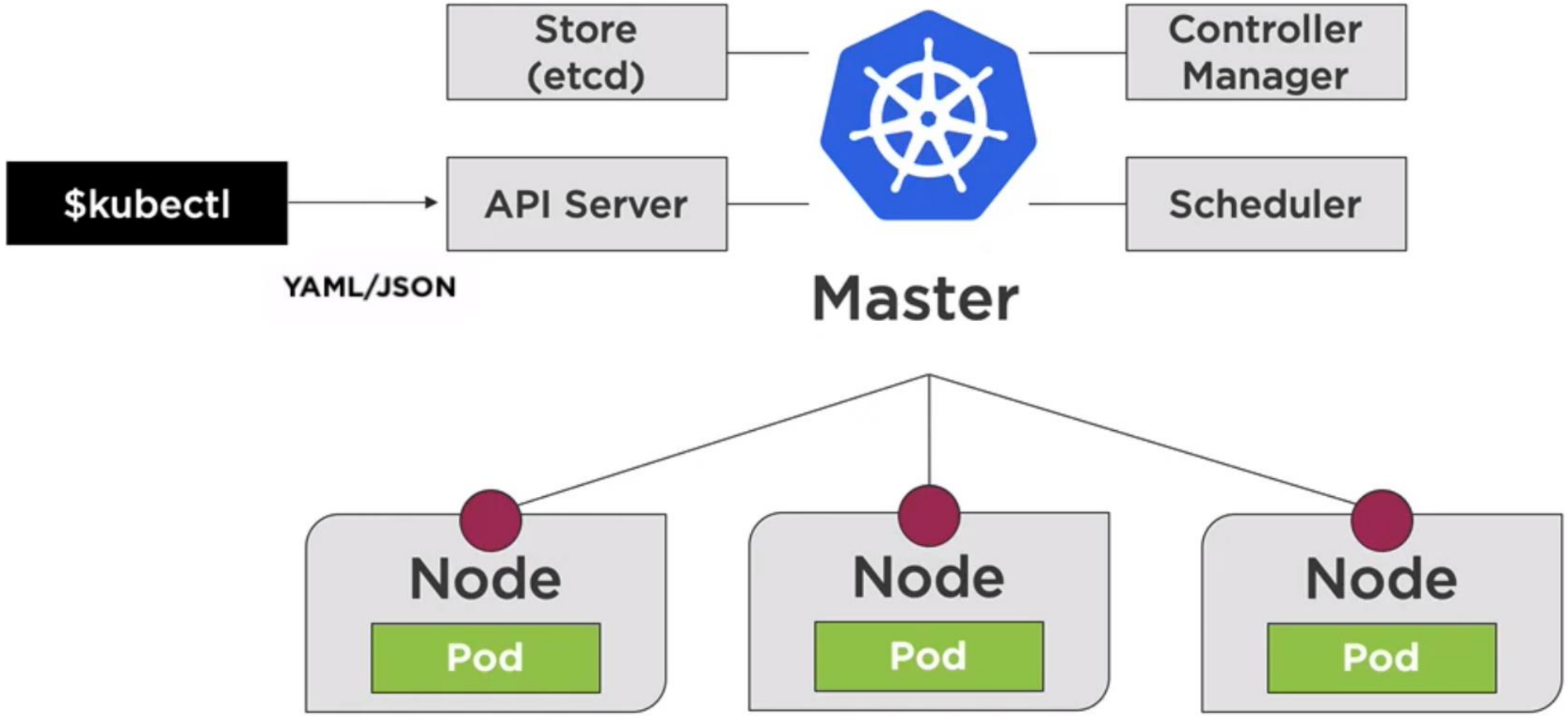
- Reduces the complexity of managing a Kubernetes Cluster by offloading it to Azure
- Easer to deploy and manage container applications without container orchestration expertise



AKS manages the following aspects of a Kubernetes cluster:

- Manages health monitoring and maintenance
- Performs simple cluster scaling
- Enables master nodes to be fully managed by Microsoft
- You’re responsible only for managing the agent nodes
- Master nodes are free, and you pay only for running agent nodes

Big Picture


Store
(etcd)
Controller
Manager
$kubectl
API Server
Scheduler
YAML/JSON
Master
Node
Pod
Node
Pod
Node
Pod

kubectl Commands

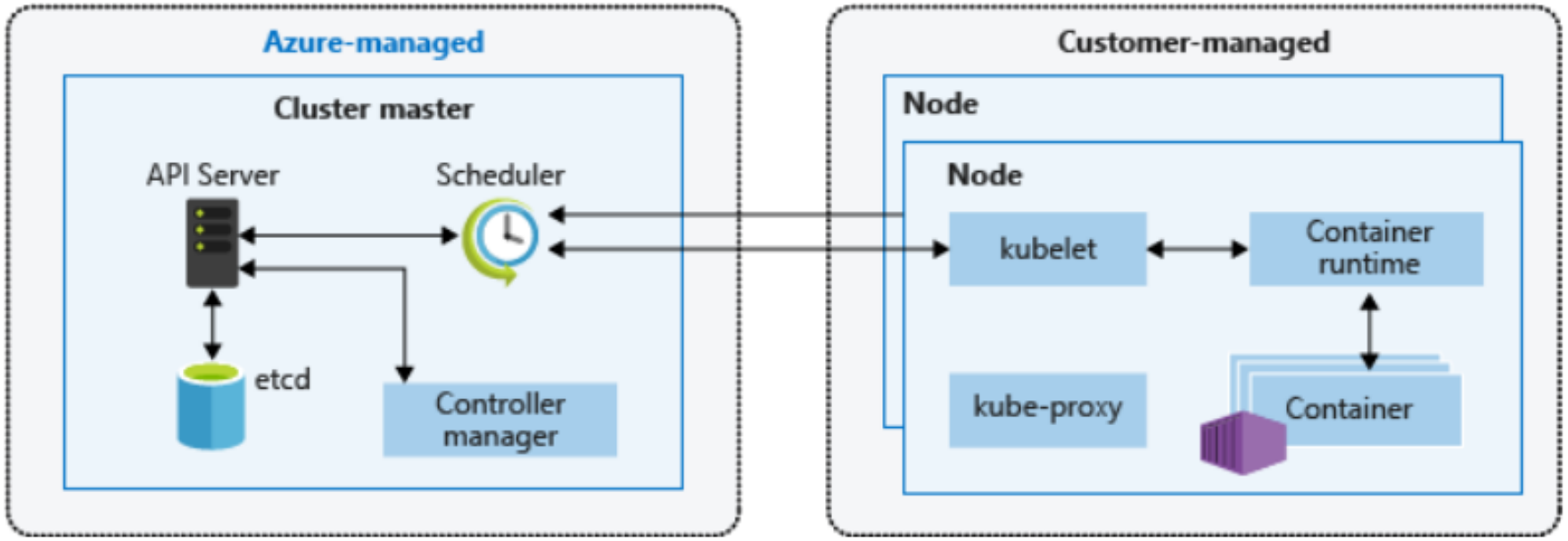
```
kubectl version

kubectl cluster-info

kubectl get all

kubectl run [container-name]
  --image=[image-name]

kubectl port-forward [pod] [ports]

kubectl expose …

kubectl create [resource]

kubectl apply [resource]
```

- Check Kubernetes version
- View cluster information
- Retrieve information about Kubernetes Pods, Deployments, Services, and more
- Simple way to create a Deployment for a Pod
- Forward a port to allow external access
- Expose a port for a Deployment/Pod
- Create a resource
- Create or modify a resource

AKS Architectural components

Kubernetes cluster is divided into two components:

- Cluster master nodes – provide core Kubernetes services and orchestration of application workloads
- Nodes that run your application workloads



Kubernetes networking

Kubernetes uses Services to logically group a set of pods together to provide network connectivity

Cluster IP

- Creates an internal IP address for use within the AKS cluster

NodePort

- Creates a port mapping on the underlying node, so an application can be accessed directly with the node IP address and port

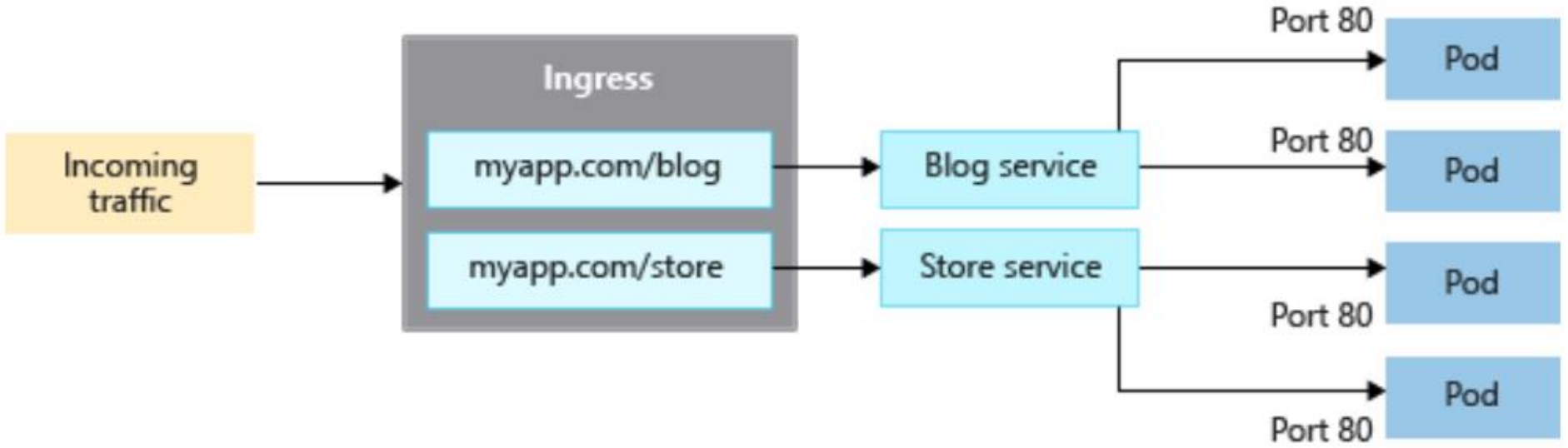
Load Balancer

- Creates an Azure LB resource, configures an external IP address, and connects the pods to the load balancer backend pool
- Customer traffic allowed through load balancing rules on the desired ports

Kubernetes networking (*Cont.*)

Ingress Controllers

- *Ingress controllers* work at Layer 7 so can use more intelligent rules to distribute traffic
- For example, route HTTP traffic to different applications based on the inbound URL



Deployment

Pod – deployment unit representing a running process on the cluster

- Used by Kubernetes to package applications
- Consists of one or more containers, Configuration, storage resources, and networking support
- Can be grouped into services to create microservices
- Described using YAML or JSON

Deploy an application to Kubernetes using the kubectl CLI

- Deploy and manage Kubernetes pods from Azure DevOps

Continuous delivery

- Build-and-release pipelines are run for every check-in on the Source repository

Walkthrough-Deploying and connecting to an AKS Cluster

Part 1: Deploy an AKS cluster using the Azure CLI

1. Open Azure Cloud Shell and choose the **Bash** environment.
2. Create an Azure resource group and an AKS cluster.
3. Using `az aks get-credentials`, connect to the Kubernetes cluster, and verify the connection status.
4. Create a YAML file called **azure-vote.yaml**.
5. Deploy the application using the **kubectl** command. It should result in the below output:

```
deployment "azure-vote-back" created
service "azure-vote-back" created
deployment "azure-vote-front" created
service "azure-vote-front" created
```

Walkthrough-Deploying and connecting to an AKS Cluster

Part 2: Monitor the health of cluster and pods running in the application

6. When the application runs, a Kubernetes service exposes the application front end to the internet.
7. Initially the **EXTERNAL-IP** for the azure-vote-front service is shown as pending.
8. Eventually changes from pending to an actual public IP address.
9. To view the application, open a web browser to the external IP address of your service.
10. Monitor health and logs under Monitoring in the Azure portal.
11. Examine container logs for the azure-vote-front pod.
12. Delete resources when finished to avoid incurring charges.

Continuous deployment

In Kubernetes, you can update a service with a rolling update

- Existing traffic to a container is first drained, and the container replaced
- Traffic is then rerouted to the container

Syntax for running a rolling update

```
kubectl apply -f nameofyamlfile
```

- The **apply** command works for an initial deployment or an update to an existing deployment

Kubernetes applies a rolling update when the name of the image for a service changes in the YAML file

- The cluster will take care of updating the images without downtime (on the assumption the application container is built as stateless)

Lesson 03: Serverless and HPC Compute Services

Lesson Overview

- Serverless computing
- Functions as a Service
- Azure Functions
- Walkthrough-Create Azure Function using Azure CLI
- Batch services

Serverless computing

Definition of *serverless computing*

- Cloud-hosted execution environment with abstraction of the underlying hosting environment
- Consumption based service, on demand provisioning and pay only for what you use. Low management overhead.
- Event-driven scalability, application components react to events and triggers in near real-time

Benefits of serverless computing include efficiency, flexibility, and focus on solving business problems

Examples of serverless Azure services: Event Grid, Azure Functions, Automation, and Logic Apps

Functions as a Service (FaaS)

Programming model that uses Functions to achieve serverless compute

Functions used this way have the following characteristics:

- Single purpose, reusable pieces of code, processing input and returning a result
- Short-lived, freeing up resources when finished execution
- Stateless, and so independent on the state of other processes
- Event driven and scalable

Azure Functions

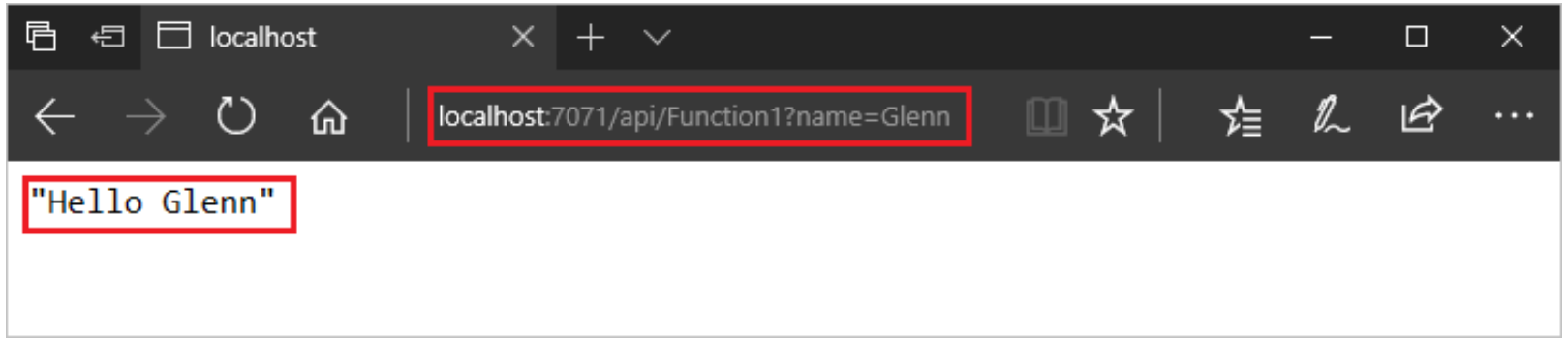
Key features of Azure Functions

- Choice of language
- Pay-per-use pricing model
- Support for NuGet and Node Package Manager
- Integrated security with OAuth providers, such as Azure AD, Facebook, Microsoft Account, and others
- Simplified integration with other Azure services and SaaS offerings
- Options for code development and deployment (portal, continuous integration, GitHub, Azure DevOps Services, and more)
- Functions runtime is open-source and available on GitHub

Walkthrough-Create Azure Functions using Azure CLI

Part 1: Create a function to run locally using the Azure CLI

1. Open Azure Cloud Shell and choose the **Bash** environment.
2. Create a local function app project.
3. Create an HTTP-triggered function and update to allow anonymous access.
4. Run the function locally and confirm the runtime output.
5. Function returns the response “Hello Glenn” to the HTTP GET request:



Walkthrough-Create Azure Functions using Azure CLI

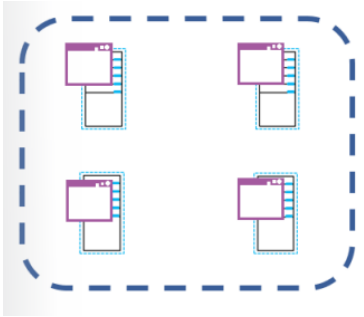
Part 2: Deploy the function app to Azure

6. Using **`az group create`**, create a resource group to store the function app and related resources.
7. Create a general-purpose storage account in the newly created resource group.
8. Use **`az functionapp create`** to create a function app, which provides the environment for serverless execution.
9. Use **`func azure functionapp publish`** to deploy the function app project to Azure.
10. Test the function (cURL on Linux or Mac, Bash on Windows).

Batch services

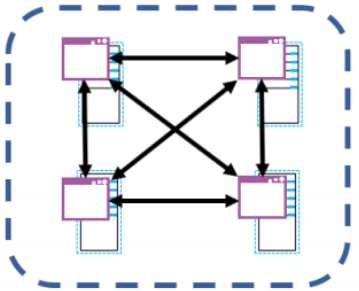
Azure Batch – fully managed cloud service for job scheduling and compute resource management

- Creates, manages, and scales a pool of compute nodes (VMs) to run largest, resource-intensive workloads.
- Provides a job scheduler to submit jobs into the VM pool (or set of VM pools)
- Two types Independent /parallel vs tightly coupled



Requirements to run applications

- Application (doesn't need to be cloud aware)
- Pool of VMs (Batch service creates, manages, and monitors the pool)
- Job scheduler to define the tasks that make up the job
- Location to store output, typically Blob storage



Batch services (*Cont.*)

Usage scenarios and application types

Independent/parallel

- Applications or tasks operate independently
- Adding VMs will speed up task completion

Tightly coupled

- High performance computing (HPC) applications (e.g. scientific/engineering)
- Examples: car crash simulations, of Artificial Intelligence (AI) training frameworks

Multiple tightly coupled in parallel

- Example: instead of four nodes to carry out a job, use 40 nodes and run the job 10 times, to scale the job task, running those jobs in parallel

Lesson 05: Application Architecture Models

Lesson Overview

- Architecture models
- N-Tier
- Web-queue-worker
- Microservices
- CQRS
- Event-driven
- Big Data
- Big Compute

Architecture models

Architecture models, or *style*s are:

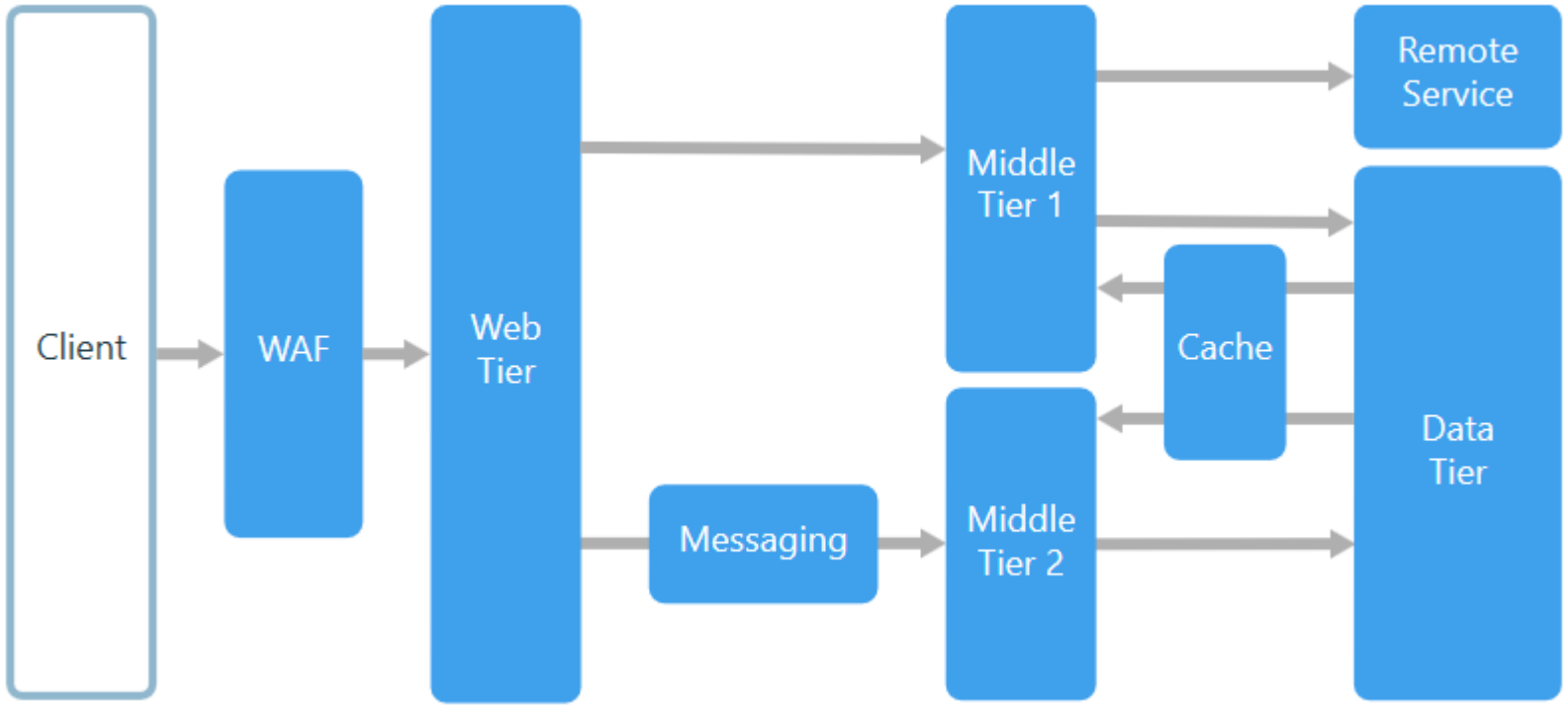
- A definition of how applications, and the services and data they consume and output, are structured.

- Not necessarily absolute but provide guidance on how you can optimize and develop applications.

- Don't require the use of particular technologies, but some technologies are well-suited for certain architectures.

N-Tier

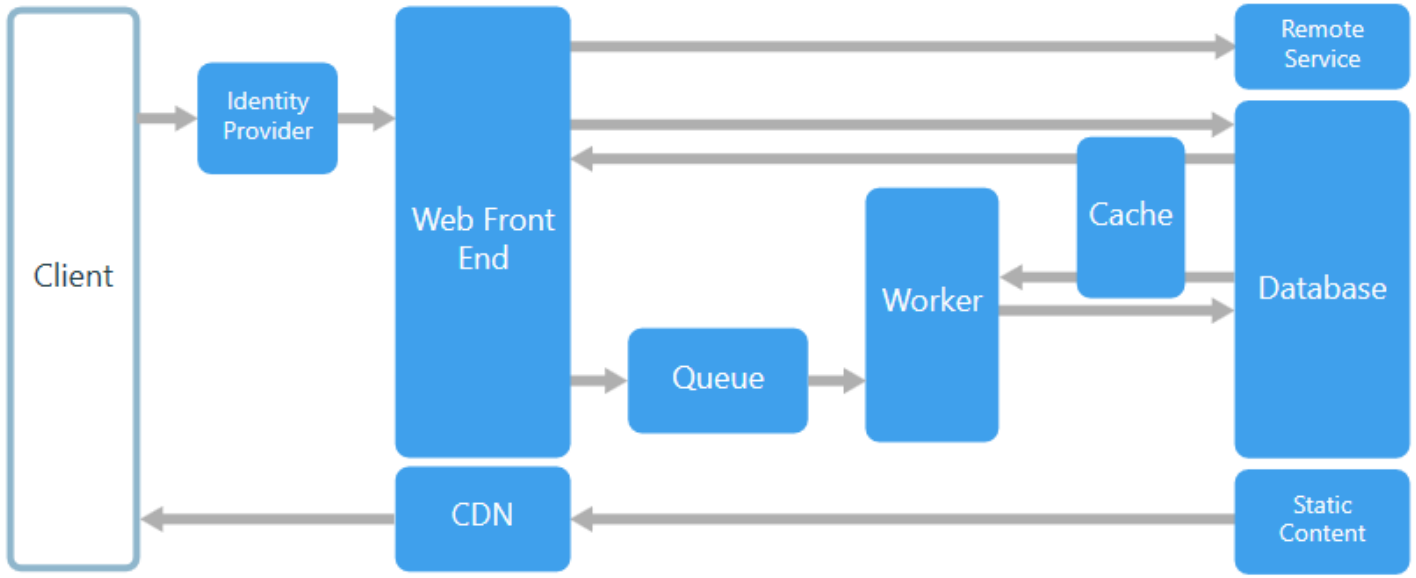
An N-tier architecture divides an application into logical layers and physical tiers



Web-Queue-Worker

Core components

- Web front end that serves client requests
- worker that performs resource-intensive tasks, long-running workflows, or batch jobs
- web front end communicates with the worker through a message queue

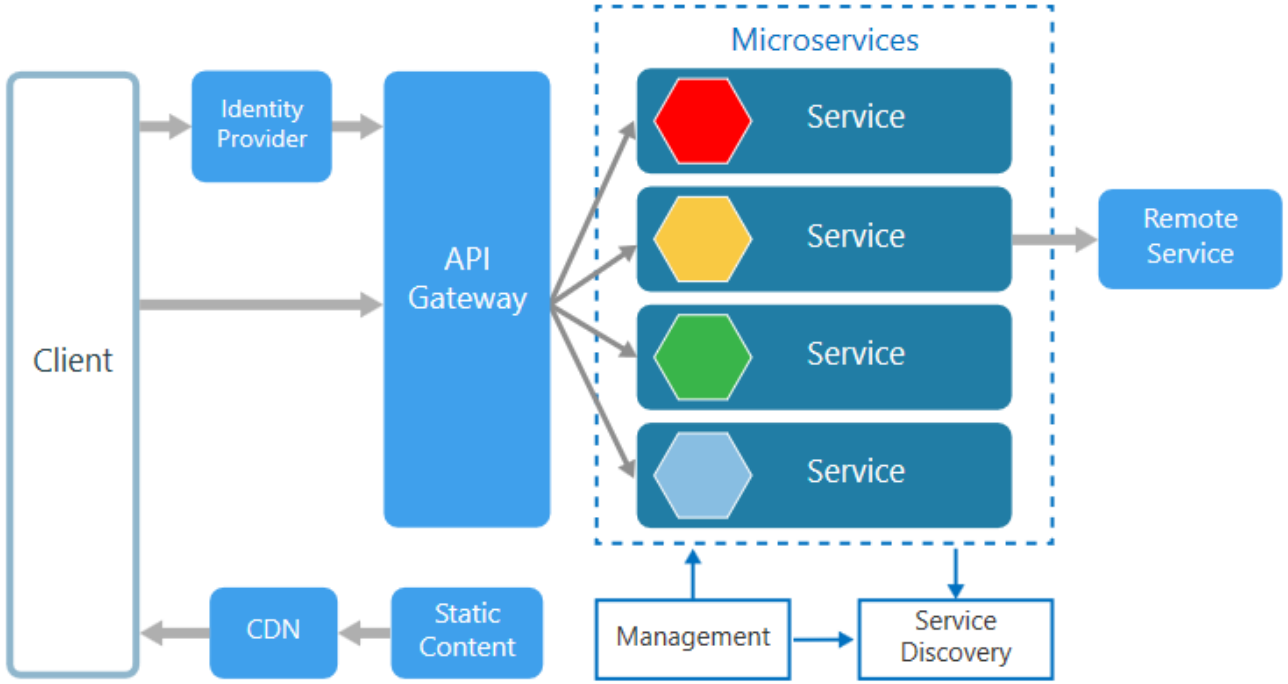


Microservices

A *microservices* architecture consists of a collection of small, autonomous services

Use microservices architecture for:

- Large applications that require a high release velocity
- Complex applications that need to be highly scalable
- Applications with rich domains or many subdomains
- An organization that consists of small development teams



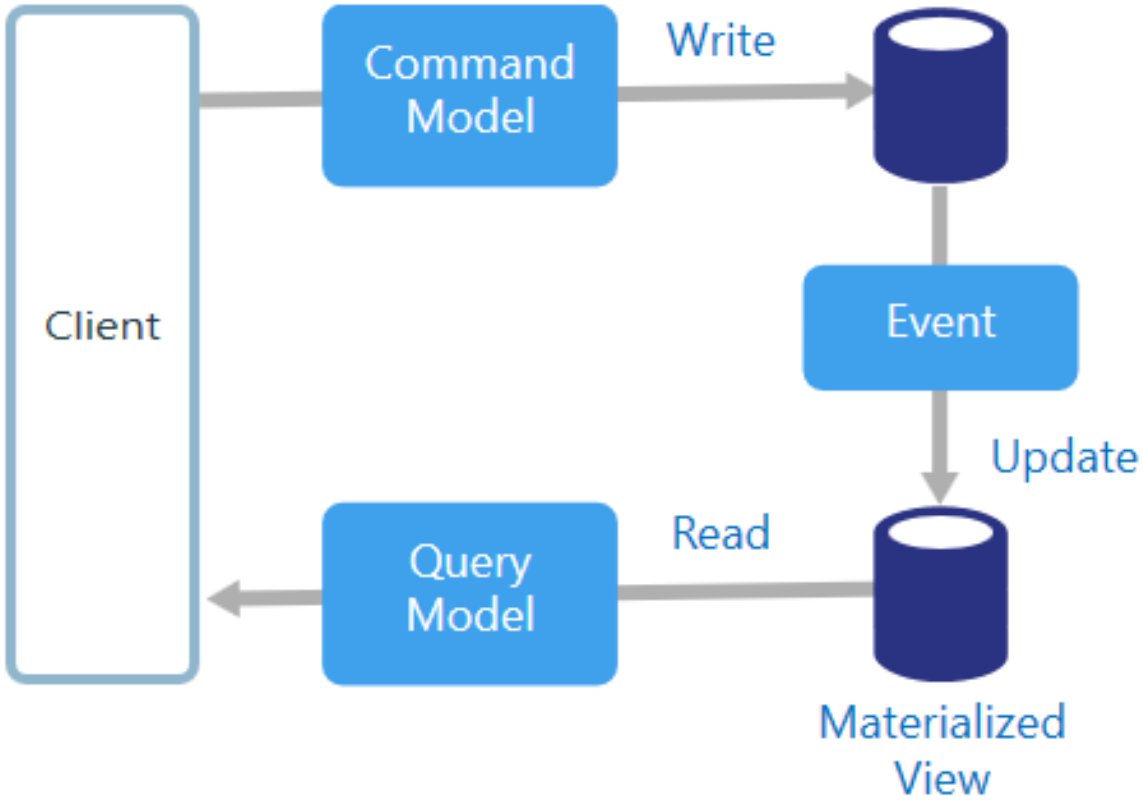
Command and Query Responsibility Segregation (CQRS)

Architecture style that separates read operations from write operations

- Commands should be task-based
  rather than data-centric
- Commands can be placed on a queue
  for asynchronous processing rather
  than being processed synchronously
- Queries never modify the database
  (returns a data transfer object)

When to use:

- Collaborative domains
  with asymmetrical read/write workloads,
  where there is clear value in separating reads and writes



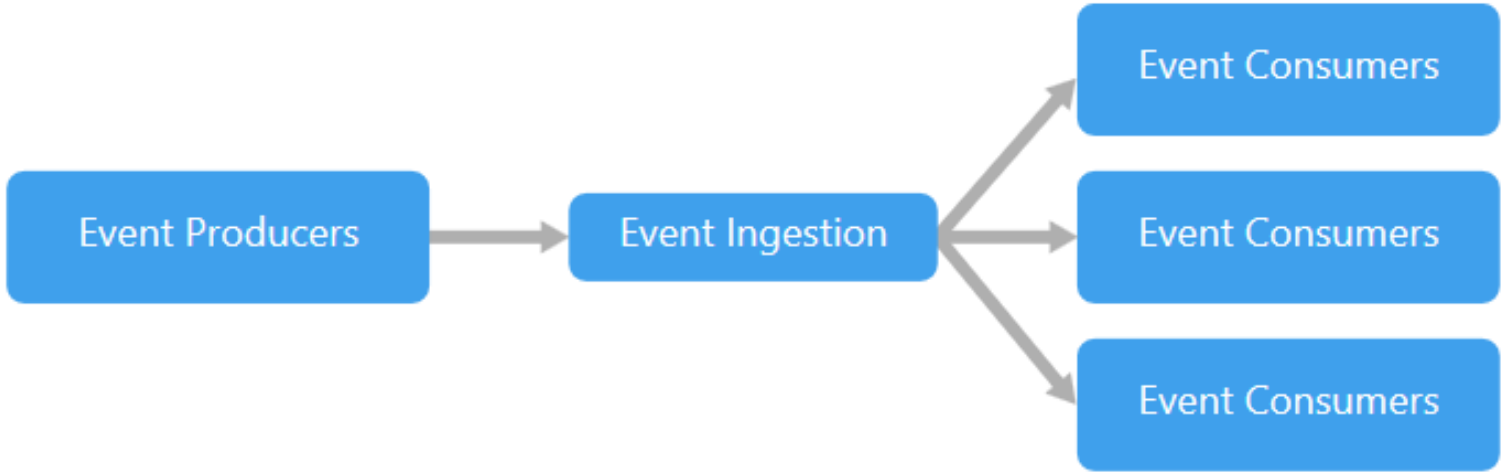
Event-Driven

Event-driven architecture

- Event producers generate stream of events
- Event consumers listen for events

Characteristics

- Delivered in near real time
- Consumers respond immediately
- Producers decoupled from consumers
- Ideal for when events must be ingested at high volumes (such as IoT)



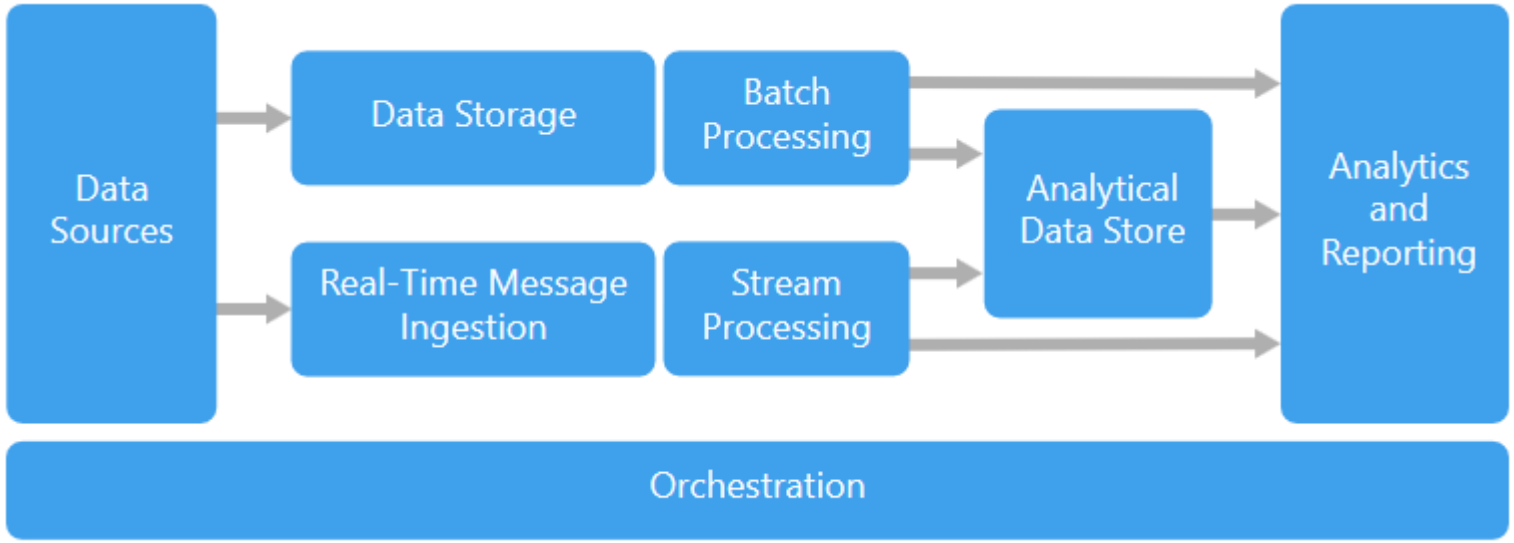
Big Data

Big data architecture

- Manage ingestion, processing, and analysis of data that's too large or complex for traditional database systems

Typical big data solutions

- Batch processing of big data sources at rest
- Real-time processing of big data in motion
- Interactive exploration of big data
- Predictive analytics and machine learning



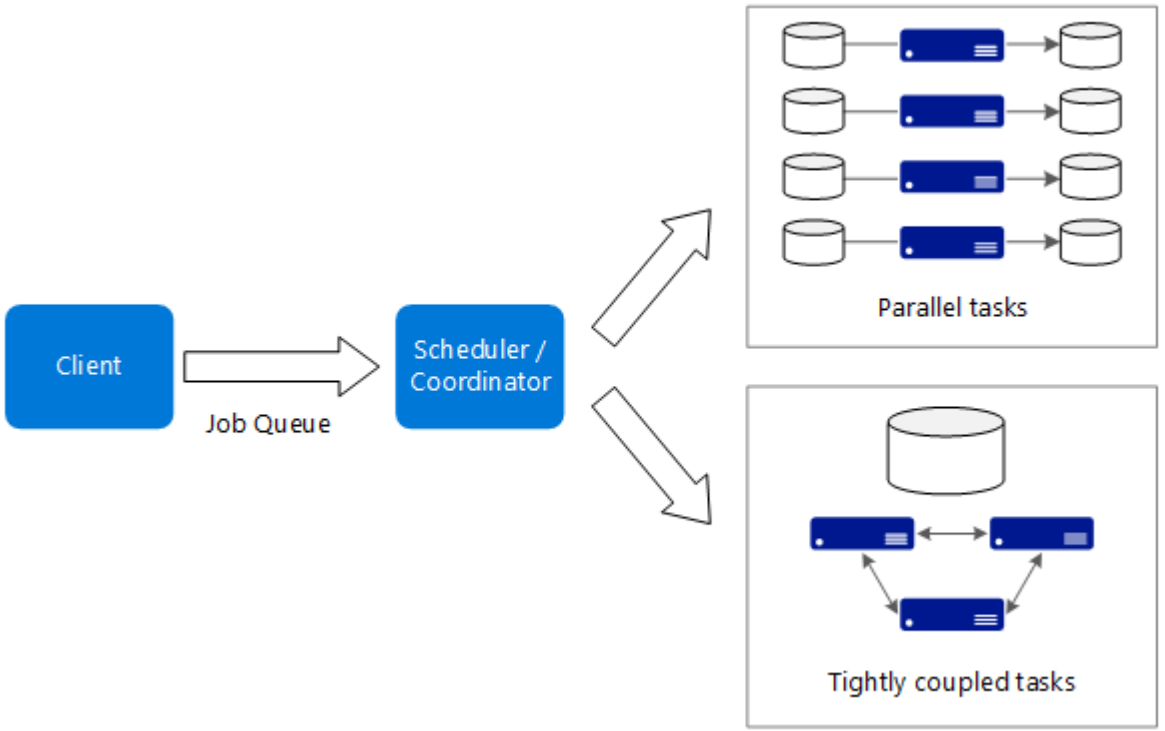
Big Compute

Big compute architecture

- Large scale workloads, requiring very large number of cores

When to use big compute

- Computationally intensive operations
- Simulations requiring too much memory for one computer
- Long computations that would take too long on a single computer
- Small computation that must be run hundreds or thousands of times



Lesson 07: Choosing a Compute Service

Lesson Overview

- Comparing IaaS, PaaS, and FaaS
- Azure compute options
- Choosing a compute service

Comparing IaaS, PaaS, and FaaS

IaaS

- You must provision the individual VMs with their associated network and storage components. Microsoft manages the physical infrastructure.

PaaS

- Azure provides a managed hosting environment for deploying applications
- Doesn’t require management of VMs or networking resources

FaaS

- Makes use of serverless architecture
- FaaS goes furthest in removing the need to worry about the hosting environment
- Simplicity, elastic scale, and paying only for time code is running

Azure compute options

IaaS:

- Azure virtual machines

PaaS:

- Azure App Service
- Azure Container Instances
- Azure Cloud Services

FaaS:

- Azure Functions
- Azure Batch

Modern native cloud apps, providing massive scale and distribution:

- Azure Service Fabric
- Azure Kubernetes Service

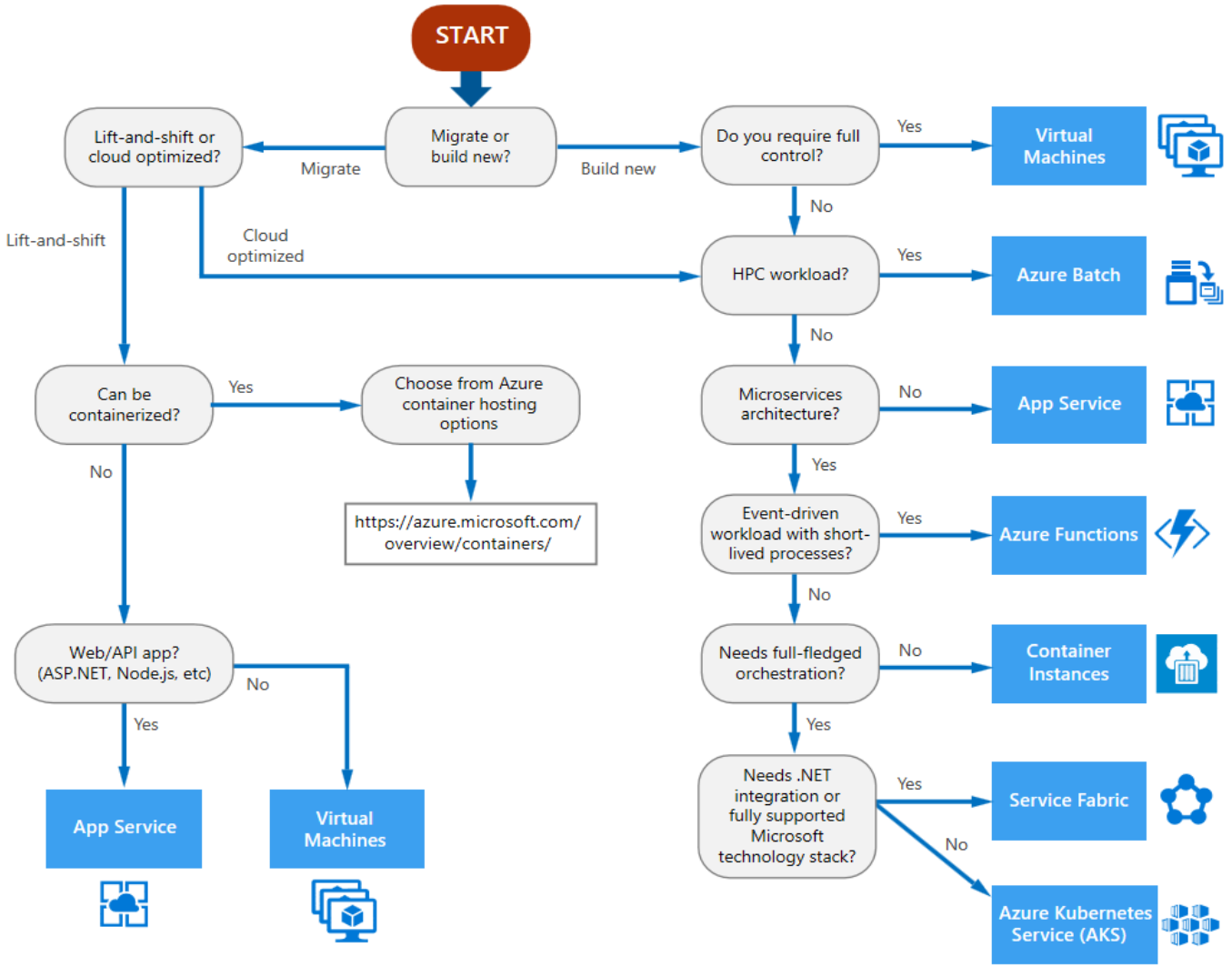
Choosing a compute service

Hosting model for your compute resources.

Evaluate each application

for its unique requirements

- Feature sets
- Service limits
- Cost
- SLA
- Regional availability
- Developer ecosystem and team skills
- Compute comparison tables



Microsoft

AZ-400T05 Module 04: Third Party and Open Source Tool Integration with Azure

Lesson 01: Chef with Azure

Lesson Overview

- What is Chef?
- Chef Automate
- Chef Cookbooks
- Chef Knife command

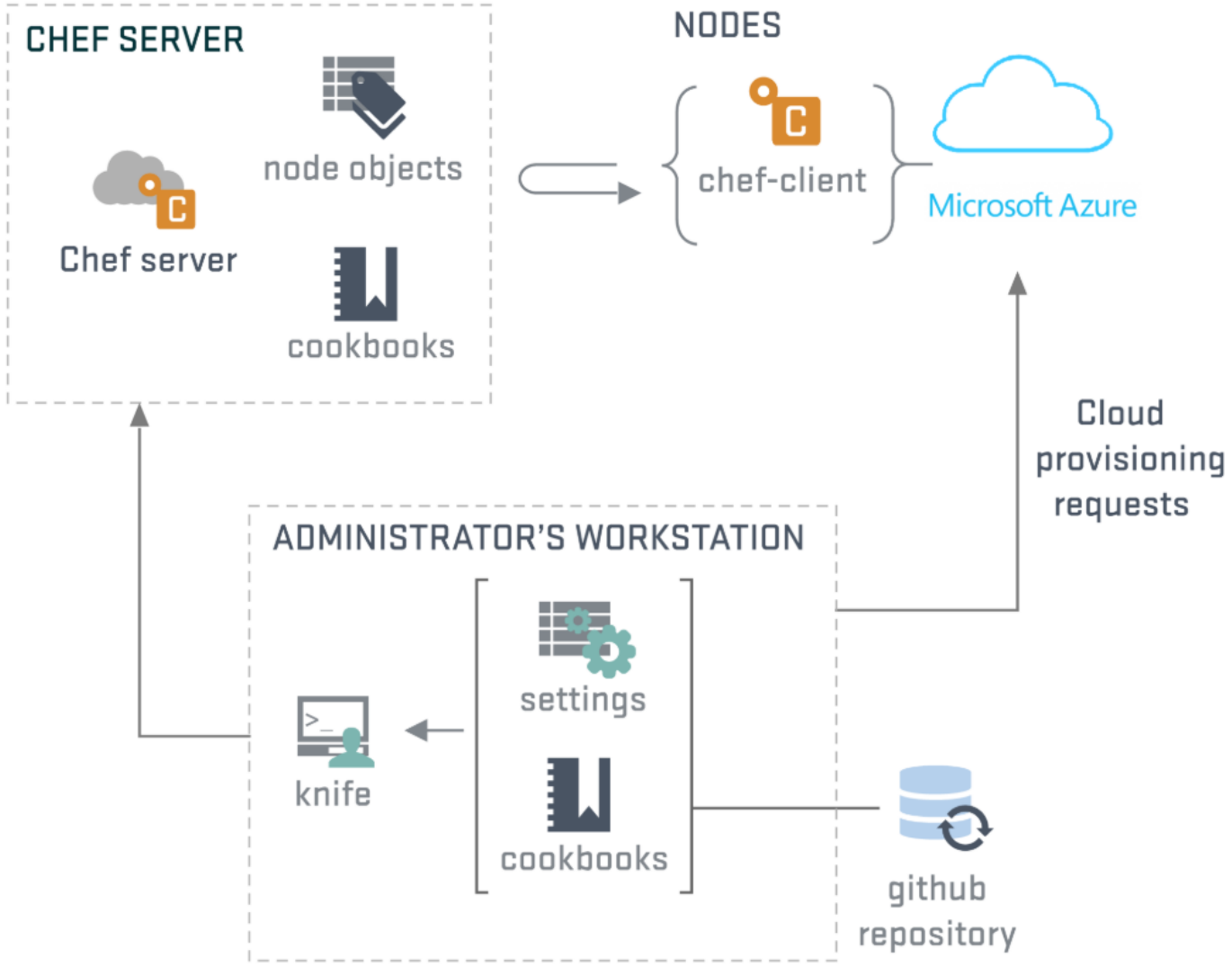
What is Chef?

- *Chef* is an infrastructure automation tool that is used for deploying, configuring, managing, and ensuring compliance of applications and infrastructure

Chef Components:

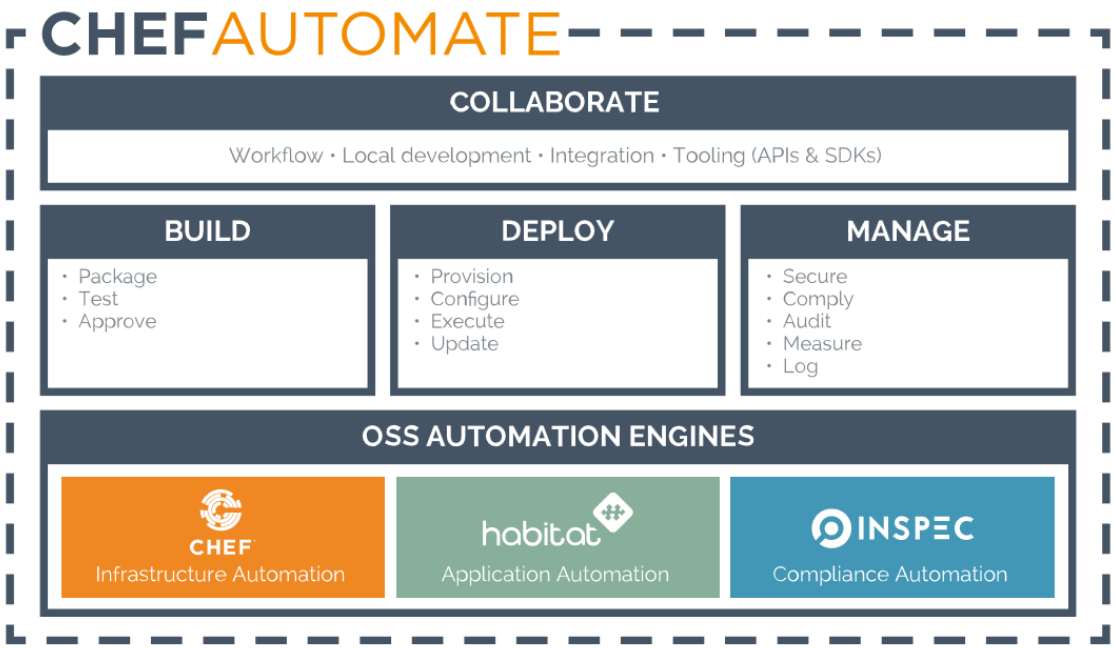
- **Chef Server -** This is the management point, which has two options for the Chef Server: a hosted solution, and an on-premises solution.
- **Chef Client (node) -** This is a Chef agent that resides on the servers you are managing.
- **Chef Workstation -** This is the Admin workstation where you create policies and execute management commands. You run the **knife** command from the Chef Workstation to manage your infrastructure.
- **Cookbooks** – Policy which defines the scenario we are applying, contains recipes, written in Ruby.
- **Recipes –** Basic configuration scripting unit, written in Ruby

Chef Architecture



Chef Automate

- *Chef Automate* is a Chef product that allows you to package and test your applications, and provision and update your infrastructure
- Available directly on Azure through the Marketplace
- Integrates with Chef Habitat and Chef InSpec
- Chef InSpec is pre-installed in Azure Cloud Shell



Chef Cookbooks

- *A cookbook* is a set of tasks that you use to configure an application or feature.
- Defines a scenario and everything required to support that scenario, and contain recipes.

Creating a Cookbook:

- Before creating a cookbook, you first configure your Chef Workstation by setting up the Chef Development Kit on your local workstation.
- You can download and install the Chef Development Kit from https://downloads.chef.io/chefdkt
- To create a cookbook, you can the ***chef generate cookbook*** command.
- After you create a cookbook, you can then create a Role. A *Role* defines a baseline set of cookbooks and attributes that you can apply to multiple servers

Chef Knife command

- *Knife* is a command that's made available as part of the Chef Development Kit installation
- Provides a wide range of management and configuration commands for cookbooks, clients, environments, roles, users and many more

Use the Knife command to perform tasks such as:

- Manage cookbooks and recipes using the *Knife Cookbook* command
- Create a role to define a baseline set of cookbooks and attributes that you can apply to multiple servers using the *Knife role* command
- Bootstrap a node or client and assign a role using the *Knife Bootstrap* command
- Manage clients using the *Knife client* command

Lesson 02: Puppet with Azure

Lesson Overview

- What is Puppet?
- Deploying Puppet in Azure
- Manifest files

What is Puppet

- *Puppet* is a deployment and configuration management toolset
- Puppet operates using a client server model

Puppet Components:

- **Puppet Master.** Responsible for compiling code to create agent catalogs.
- **Puppet Agent.** The machine (or machines) managed by the Puppet Master
- **Console Services**. Web-based user interface for managing systems.
- **Facts**. Metadata related to state.

Deploying Puppet in Azure

There are two options for deploying Puppet in Azure:

Azure Marketplace:

- Puppet Enterprise is available to install directly into Azure using the Azure Marketplace.
- The Puppet Enterprise image allows you to manage up to 10 Azure VMs for free, and is available to use immediately

Azure virtual machines:

- Install a Linux VM in Azure and deploy the Puppet Enterprise package manually

Manifest files

- Puppet uses a declarative file syntax to define state.
- State, is defined in manifest files known as *Puppet Program* files. These have the file extension **.pp**.

Puppet program files have the following elements:

- **Class -** A bucket that you put resources into. That class then becomes an entity that you can use to compose other workflows.
- **Resources -** A single elements of your configuration that you can specify parameters for.
- **Module -** The collection of all the classes, resources, and other elements of the Puppet program file in a single entity

Lesson 03: Ansible with Azure

Lesson Overview

- What is Ansible?
- Ansible components
- Installing Ansible
- Ansible on Azure
- Playbook structure
- Walkthrough-Run Ansible in Azure Cloud Shell
- Walkthrough-Run Ansible in Visual Studio Code

What is Ansible?

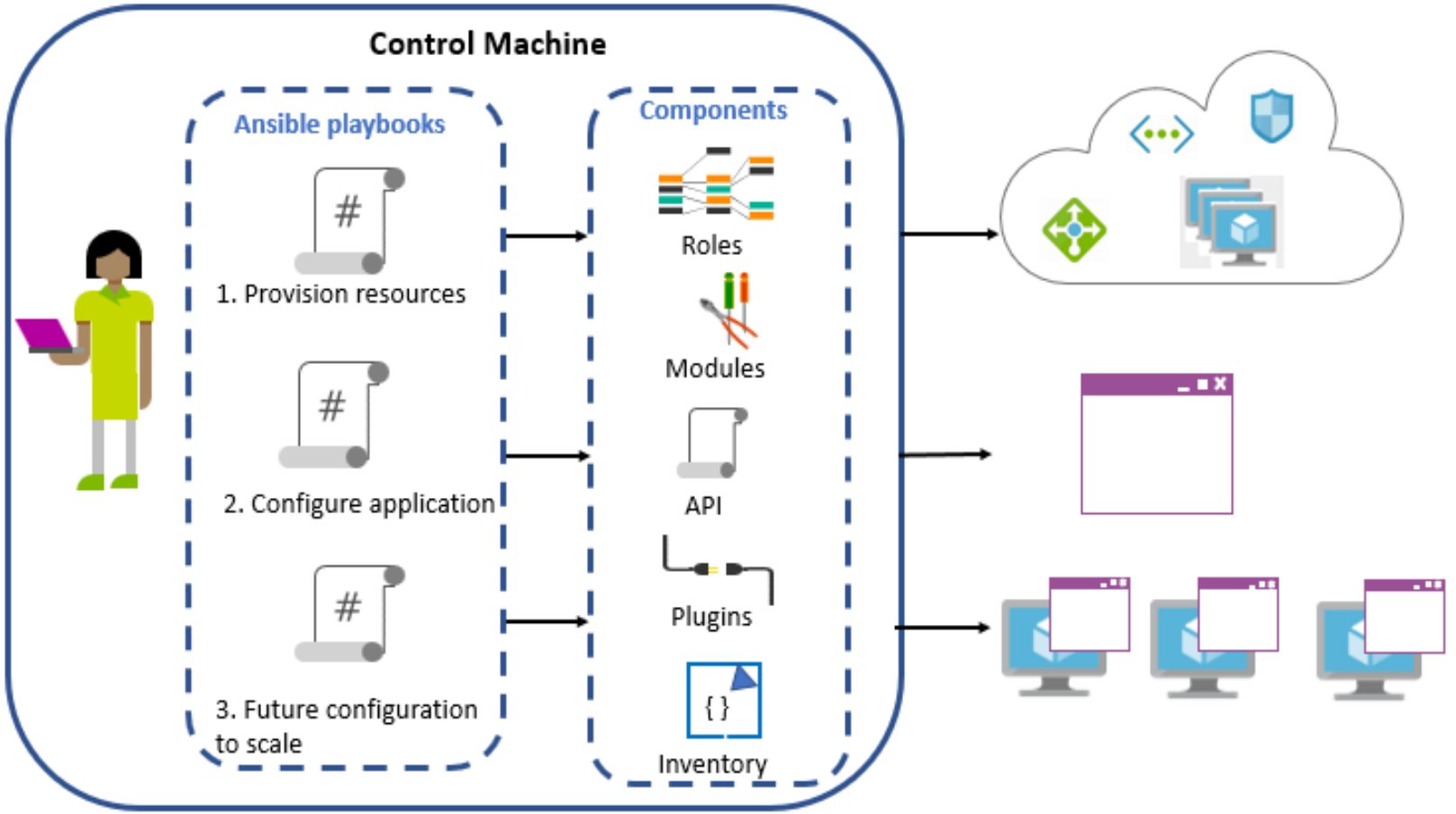
- *Ansible* is an open-source platform that automates cloud provisioning, configuration management, and application deployments.
- Allows you to automate deployment and configuration of resources in your environment such as virtual networks, storage, subnets, and resources groups
- Unlike Puppet or Chef, Ansible is *agentless*, so you do not have to install software on the managed machines.
- Ansible also models your IT infrastructure by describing how all of your systems interrelate, rather than managing just one system at a time

Ansible Components

- Control Machine. This is the machine from which the configurations are run.
- Managed Nodes. These are the devices and/or machines and environments that are being managed.
- Playbooks. Playbooks are ordered lists of tasks, written in YAML, that have been saved so you can run them in the same order repeatedly.
- Modules. Ansible works by connecting to your nodes and then pushing out to the nodes small programs (or *units of code*), called *modules*. *Modules* are the units of code that define the configuration. They are modular, and can be re-used across playbooks

Ansible Workflow

- The following workflow and component diagram outlines how playbooks can be run in different circumstances, one after another



1) Provision resources.
2) Configure the application.
3) Manage future configurations to scale

Installing Ansible

To enable a machine to act as the control machine from which to run playbooks, you need to install both Python and Ansible

Python:

- Must install either Python 2 (version 2.7), or Python 3 (versions 3.5 and higher).

Ansible:

- Only need to install Ansible on one machine, which could be workstation or a laptop—you can manage an entire fleet of remote machines from that central point. Can be Linux, macOS or Windows
- No database is installed as part of the Ansible setup.
- No daemons are required to start or keep running.

Ansible on Azure

There are a number of ways you can use Ansible in Azure.

Azure marketplace

- Must install either Python 2 (version 2.7), or Python 3 (versions 3.5 and higher).

Azure virtual machines:

- Only need to install Ansible on one machine, which could be workstation or a laptop—you can manage an entire fleet of remote machines from that central point. Can be Linux, macOS or Windows
- No database or daemons are installed as part of the Ansible setup.

Azure Cloud Shell:

- Ansible is installed by default in Azure Cloud Shell

Playbook structure

- Playbooks manage configurations of and deployments to remote machines
- structured with YAML (a data serialization language)
- Declarative
- Include detailed information regarding the number of machines to configure at a time

YAML structure:

- Yaml is based around the structure of key value pairs
- Indentations and new lines are used to separate key value pairs
- No definition on how to space indentation, but he indentations must be uniform throughout the file.

Walkthrough-Run Ansible in Azure Cloud Shell

- This walkthrough will create a resource group in Azure using Ansible in Azure Cloud Shell with bash. Azure Cloud Shell, has Ansible pre-installed, so you do not have to install or configure anything to be able or run Ansible.

- You can complete this walkthrough task by completing the steps in the course content, or you can simply read through them, depending on your available time

Walkthrough-Run Ansible in Visual Studio Code

- This walkthrough will install the Ansible extension in Visual Studio Code, create an ansible yaml file to crate a virtual machine in Azure, run the yaml file from within visual studio code, verify creation of the virtual machine in Azure.

- You can complete this walkthrough task by completing the steps outlined below, or you can simply read through them, depending on your available time

Lesson 04: Cloud-init with Azure

Lesson Overview

- What is cloud-init
- Cloud-init components
- Cloud-init on Azure
- Walkthrough-Configure a Linux VM using cloud-init and Azure Cloud Shell

What is Cloud-init?

- A widely used approach to customize a Linux VM as it boots for the first time
- Can install packages, write files, and configure users and security.
- Because cloud-init is called during the initial boot process, there are no additional steps or required agents to apply your configuration

- Cloud-init also works across Linux distributions. For example, you don't need to use *apt-get install* or *yum install* to install a package. Instead, you define a list of packages to install, and cloud-init automatically uses the native package management tool for the distribution you select.

Cloud-init Components

Cloud-init is run on Azure by using a configuration definition file, known as *cloud-config*. This file is in the form of a *.txt* file and uses the *.yml* file structure

The *.txt* cloud-config file is applied using the Azure Command-Line Interface (Azure CLI) *az* command using the *–custom-data* parameter, and then specifying the *.txt* cloud-config file.

```
#cloud-config
package_upgrade: true
packages:
   - httpd
```

```
az vm create \
   --resource-group myResourceGroup \
   --name centos74 \
   --image OpenLogic:CentOS:7-CI:latest \
   --custom-data cloud-init.txt \
   --generate-ssh-keys
```

Cloud-init on Azure

- If cloud-init is already installed in the Linux image, you need not do anything else to use cloud-init because it works as soon as it is installed.
- Microsoft is actively working with endorsed Linux distribution partners to have cloud-init enabled images available in the Azure marketplace

Current cloud-init enabled images available on the Azure platform:

- Ubuntu Server
- CoreOS
- CentOS
- RHEL

Walkthrough-Configure a Linux VM using Cloud-init and Azure Cloud Shell

- This walkthrough will create a Linux VM using Azure CLI, and configure it on boot up, using cloud-init.

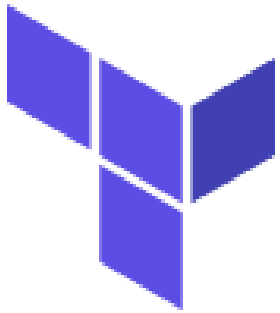
- You can complete this walkthrough task by completing the steps outlined below, or you can simply read through them, depending on your available time

Lesson 05: Terraform with Azure

Lesson Overview

- What is Terraform?
- Terraform components
- Terraform on Azure
- Installing Terraform
- Terraform config file structure
- Walkthrough-Run Terraform in Azure Cloud Shell
- Walkthrough-Run Terraform in Visual Studio Code

What is Terraform



- *HashiCorp Terraform* is an open-source tool that allows you to provision, manage, and version cloud infrastructure.

- Terraform's command-line interface (CLI) provides a simple mechanism to deploy and version the configuration files to Azure.
- Supports multi-cloud scenarios which enables developers to use the same tools and configuration files to manage infrastructure on multiple cloud providers.

Terraform components

Some of the Core components of Terraform include:

- **Configuration files -** Text-based configuration files allow you to define infrastructure and application configuration, and end in the *.tf* or *.tf.json* extension.
- **Terraform CLI -** *Terraform CLI* is a command-line interface from which you run configurations. You can run command such as **Terraform apply** and **Terraform plan**, along with many others.
- **Modules -** *Modules* in Terraform are self-contained packages of Terraform configurations that are managed as a group

Terraform on Azure

Some of the Core components of Terraform include:

- **Configuration files -** Text-based configuration files allow you to define infrastructure and application configuration, and end in the *.tf* or *.tf.json* extension.
- **Terraform CLI -** *Terraform CLI* is a command-line interface from which you run configurations. You can run command such as **Terraform apply** and **Terraform plan**, along with many others.
- **Modules -** *Modules* in Terraform are self-contained packages of Terraform configurations that are managed as a group

Terraform on Azure

Options for deploying Terraform in Azure include:

Azure Marketplace:

- Offers a fully-configured Linux image containing Terraform.

Azure virtual machines:

- Deploy a Linux or Windows VM in Azure VM's IaaS service, install Terraform and the relevant components, and then use that image.

Azure Cloud Shell:

- Terraform is installed by default in Azure Cloud Shell

Installing Terraform

Options for deploying Terraform in Azure include:

- You must install Terraform on the machine from which you are running the Terraform commands.
- Can be installed on Windows, Linux or macOS environments
- You can download appropriate Terraform install package from https://www.terraform.io/downloads.html

Authenticate Terraform with Azure using:

- The Azure CLI
- A Managed Service Identity (MSI)
- A service principal and a client certificate or secret

Terraform config file structure

```
# Configure the Microsoft Azure Provider
provider "azurerm" {
    subscription_id = "xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
    client_id       = "xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
    client_secret   = "xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
    tenant_id       = "xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
}

# Create a resource group if it doesn't exist
resource "azurerm_resource_group" "myterraformgroup" {
    name     = "myResourceGroup"
    location = "eastus"

    tags {
        environment = "Terraform Demo"
    }
}
```

Walkthrough-Run Terraform in Azure Cloud Shell

- This walkthrough will create a create a resource group in Azure using Terraform in Azure Cloud Shell.

- You can complete this walkthrough task by completing the steps outlined in the course, or you can simply read through them, depending on your available time

Walkthrough-Run Terraform in Visual Studio Code

- This walkthrough will create a VM in Visual Studio Code using Terraform

- You can complete this walkthrough task by completing the steps outlined in the course, or you can simply read through them, depending on your available time

Microsoft

AZ-400T05 Module 05: Compliance and Security

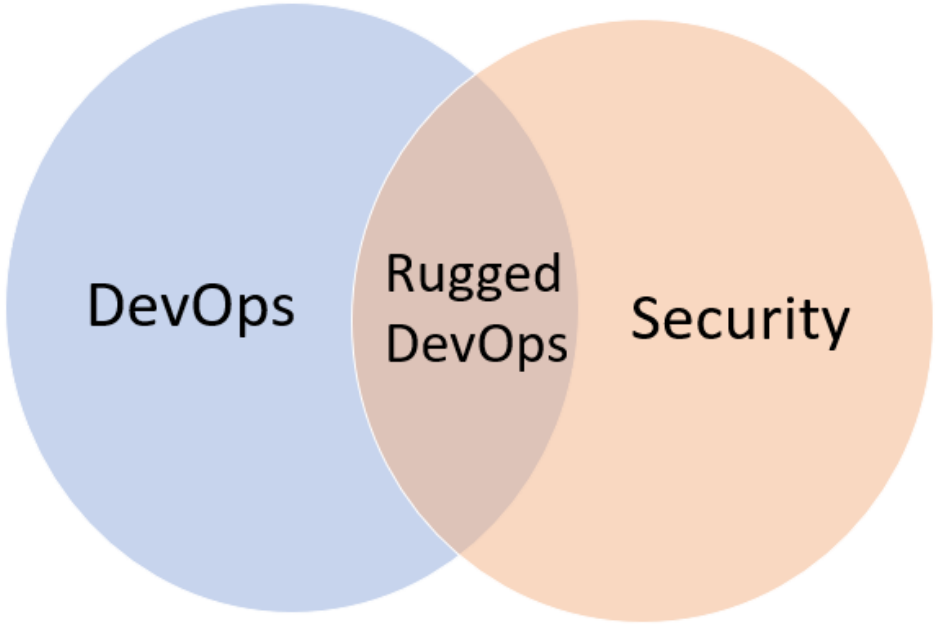
Lesson 01: Security and Compliance in the Pipeline

Lesson Overview

- What is Rugged DevOps?
- Rugged DevOps pipeline
- Software composition analysis (SCA)
- WhiteSource integration with Azure DevOps pipeline
- Micro Focus Fortify integration with Azure DevOps pipeline
- Checkmarx integration with Azure DevOps
- Veracode integration with Azure DevOps
- How to integrate SCA checks into pipelines
- Ops and pipeline security
- Secure DevOps Kit for Azure (AzSK)

What is Rugged DevOps

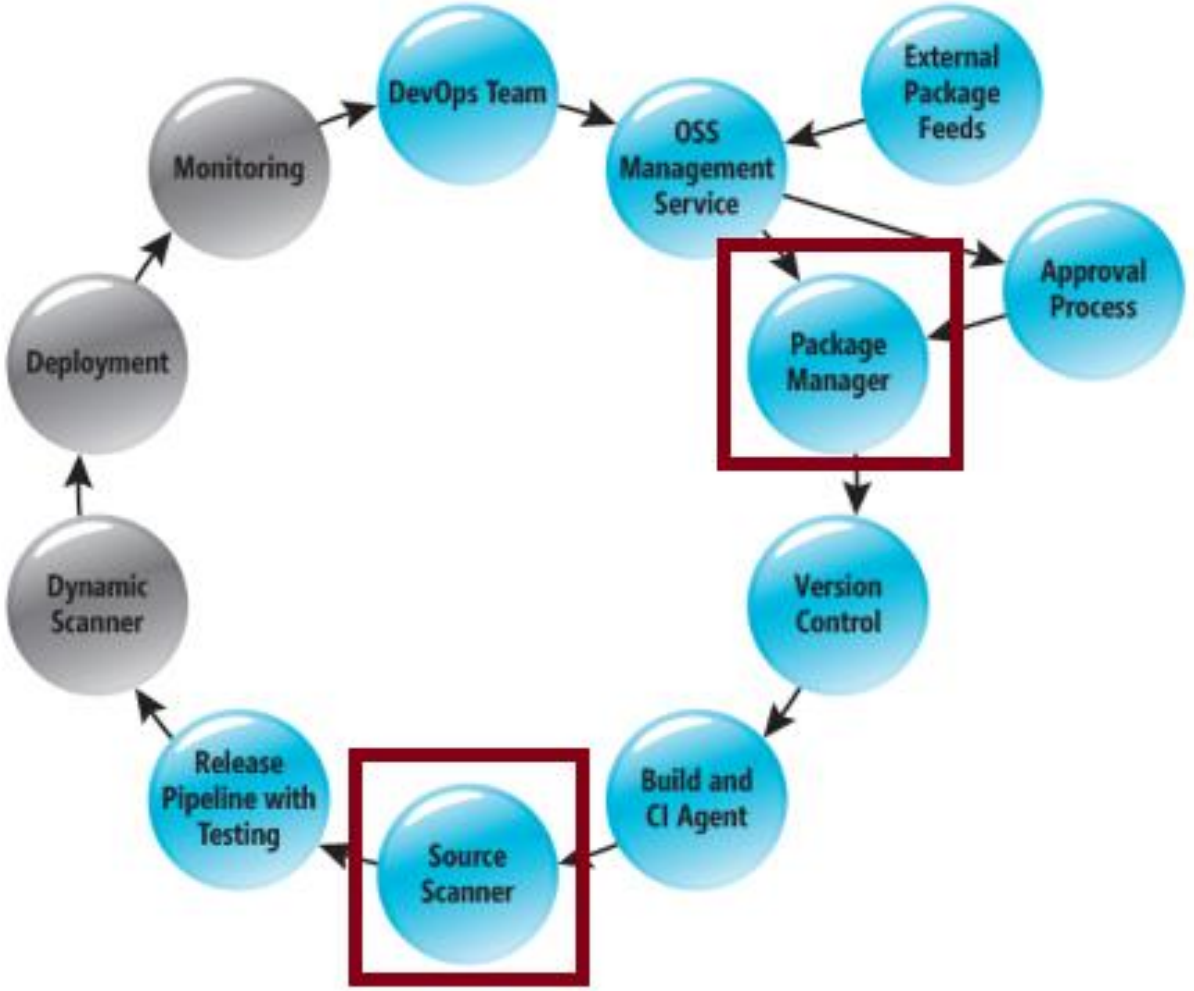
- Rugged DevOps is a set of practices designed integrate DevOps and security
- The goal is to enable development teams to work fast without introducing unwanted vulnerabilities
- Security strategy includes access control, environment hardening, perimeter protection, and more



Rugged DevOps is also sometimes referred to as *DevOpsSec*

Rugged DevOps pipeline

- Package Management, and the approval process associated with it, accounts for how software packages are added to the pipeline, and the approval process they need to go through
- Source Scanner is for performing a security scan to verify certain security vulnerabilities are not present in our application source code



Software composition analysis (SCA)

- Package Management
  - Unique source of binary components
  - Create a local cache of approved components
  - Use Azure Artifacts to share and organize your packages
  - Package types: NuGet, npm, Maven, Gradle, Universal, and Python
- Open Source Software (OSS) Components
  - Reused dependencies can have security vulnerabilities
  - Ensure you have the latest version
  - Check for the correct binaries
  - Promptly address vulnerabilities
  - Analyzing the software to determine its composition can help mitigate potential vulnerabilities

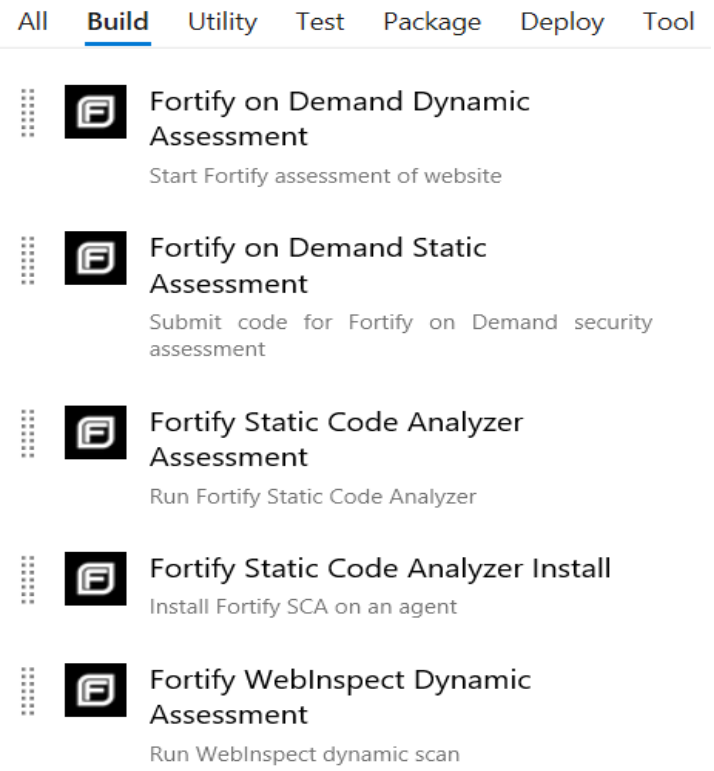
WhiteSource integration with Azure DevOps pipeline

- *Azure DevOps Marketplace* is an important site for addressing rugged DevOps issues. From here you can integrate specialist security products into your Azure DevOps pipeline.
- Azure DevOps Marketplace can be access at https://marketplace.visualstudio.com/azuredevops

- *WhiteSource* is one such example of an extension available on the Azure DevOps Marketplace.
- If consuming external packages, the *WhiteSource* extension specifically addresses the questions of open source security, quality, and license compliance. With *WhiteSource* you can:
  - Continuously detect all open-source components in your software
  - Receive alerts on open-source security vulnerabilities
  - Automatically enforce open-source security and license compliance

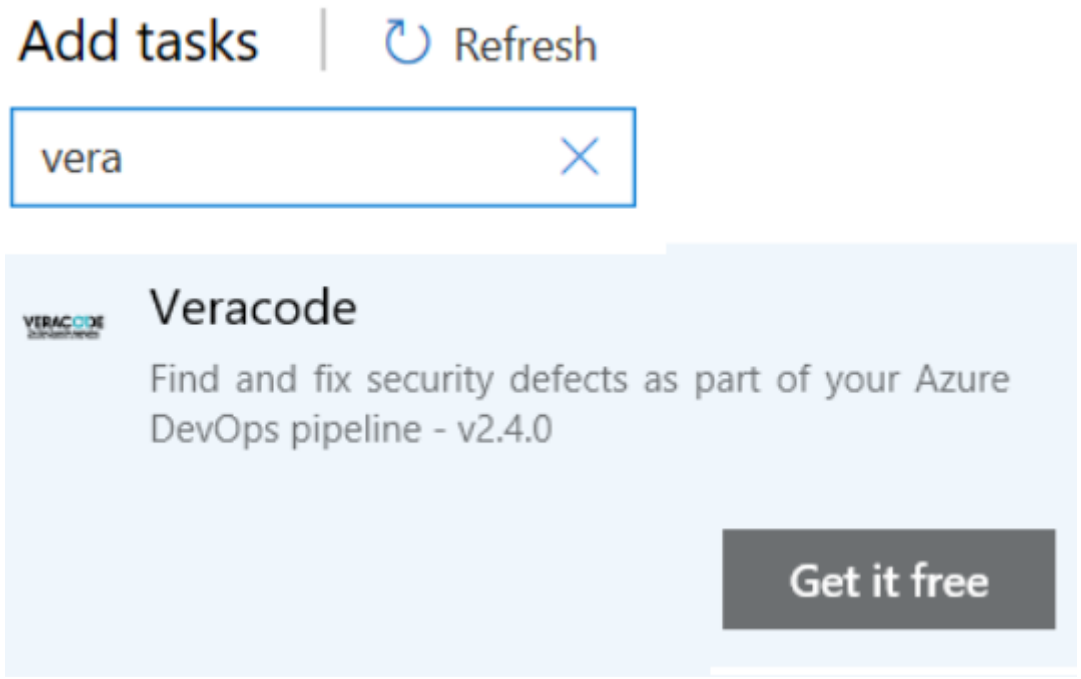
Micro Focus Fortify integration with Azure DevOps pipeline

- Can add build tasks to CI/CD pipeline to help identify vulnerabilities in your source code
- Provides a comprehensive set of software security analyzers
- **Fortify Static Code Analyzer:** Identifies root causes of software security vulnerabilities
- **Fortify on Demand** automatically submits static and dynamic scan requests



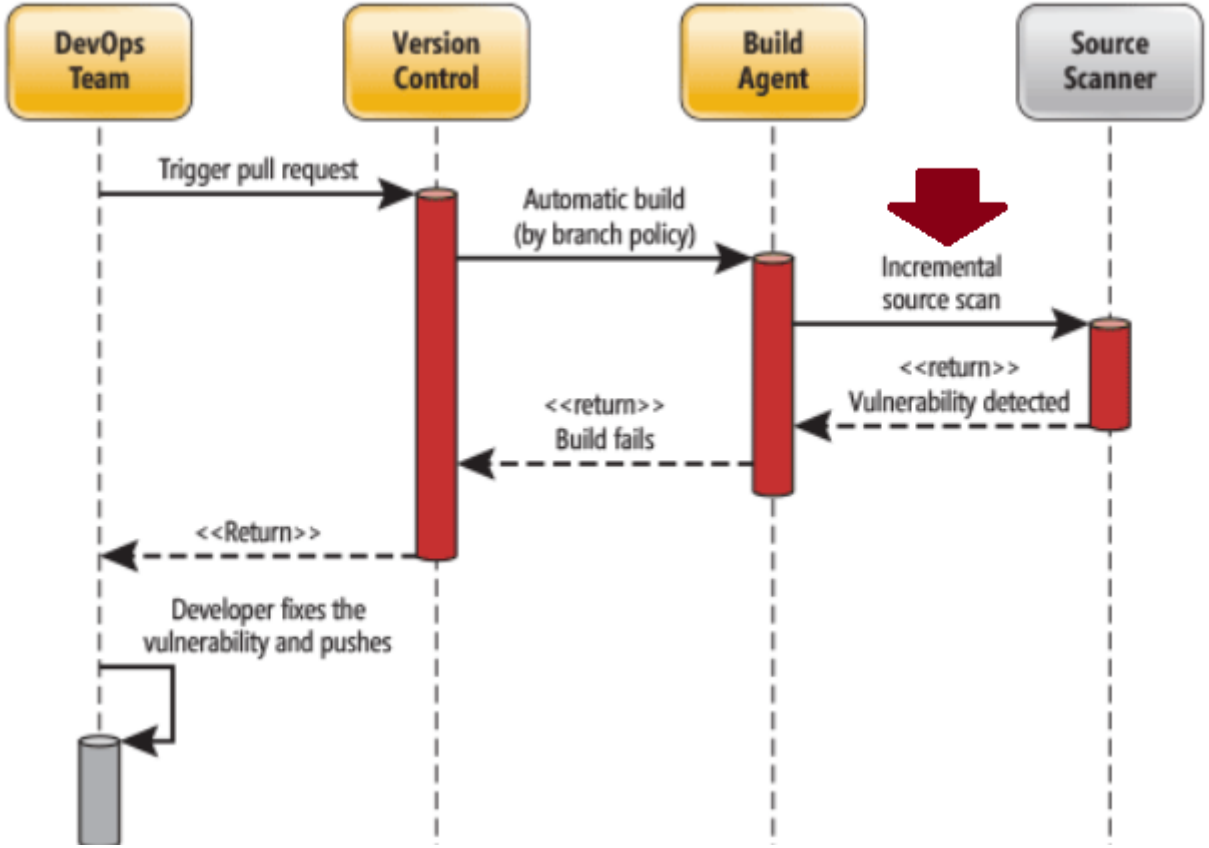
Veracode integration with Azure DevOps

- Integrate application security into your development tools
- Don't stop for false alarms
- Align your application security practices with your development practices
- Don't just find vulnerabilities, fix them
- Onboarding process allows for scanning on day one



How to integrate software composition analysis Checks into Pipelines

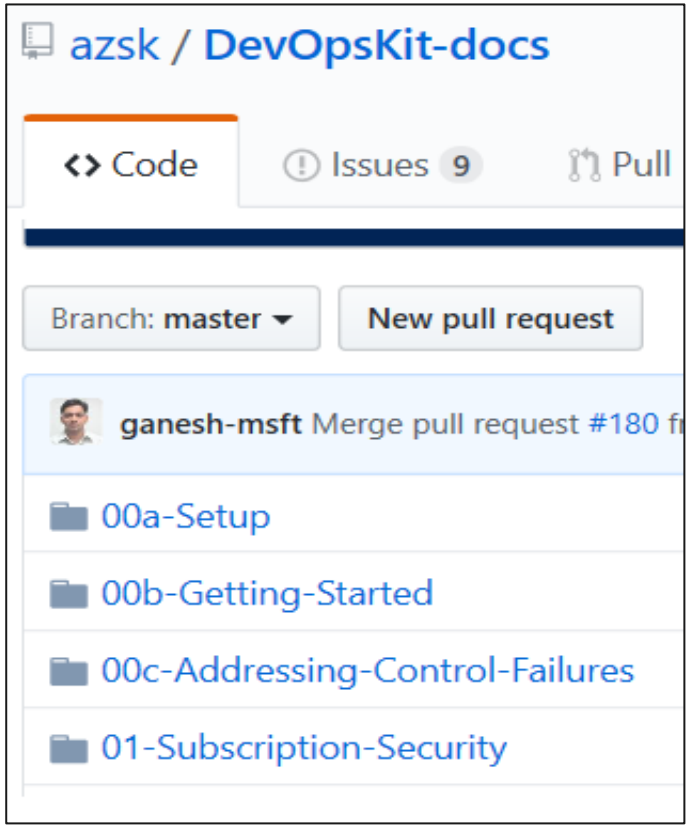
- *Pull requests* are the way DevOps teams submit changes
- WhiteSource, Checkmarx, Veracode, and Black Duck by Synopsis can facilitate incremental scans
- Integrate scanning into a team's workflow at multiple points along the path



Secure DevOps Kit for Azure (AzSK)

AzSK is a collection of scripts, tools, extensions, automations

- Helps secure the subscription
- Enables secure development
- Integrates security into CI/CD
- Provides continuous assurance
- Assists with alerts & monitoring
- Governing cloud risks



Available at: https://github.com/azsk/DevOpsKit-docs

Lesson 02: Azure Security and Compliance Tools and Services

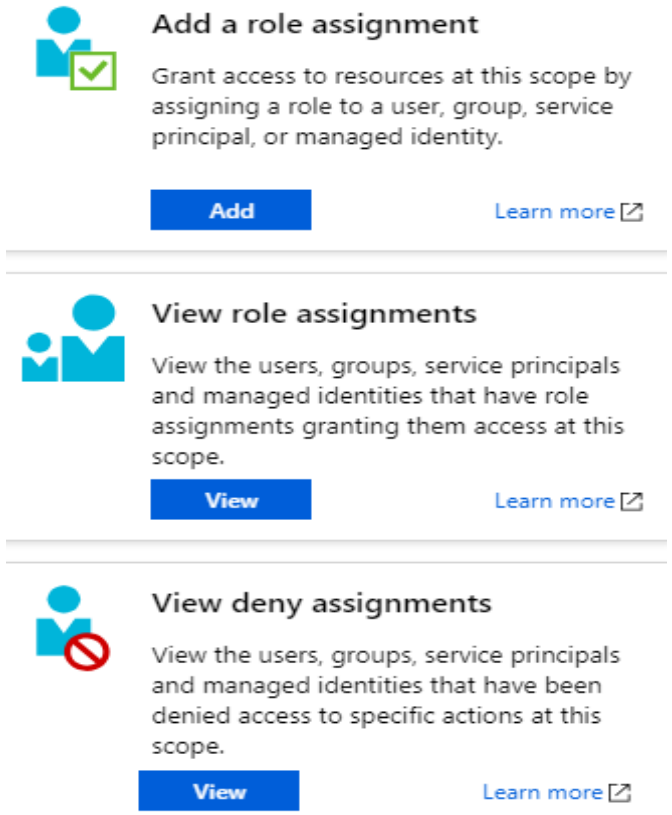
Lesson Overview

- Azure Security Center
- Azure Security Center usage scenarios
- Azure Policy
- Policies
- Initiatives
- Azure Key Vault
- Role-Based Access Control (RBAC)
- Locks
- Subscription governance
- Azure Blueprints
- Azure Advanced Threat Protection

Role-Based Access Control (RBAC)

Segregate duties within your team and grant only the amount of access to users that they need to perform their jobs

- Fine-grained access management for Azure resources
- Grant users the lowest privilege level that they need to do their work
- No additional cost to all Azure subscribers
- Built-in roles for an *allow model*



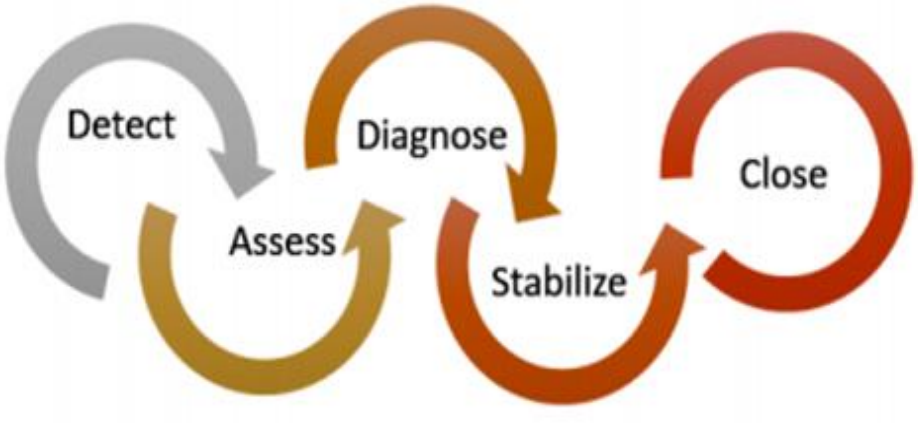
Azure Security Center

- Provide security recommendations
- Monitor security settings
- Monitor all your services
- Use Azure Machine Learning
- Analyze and identify potential attacks
- Provide just-in-time (JIT) access control
- Supports Windows and Linux operating systems
- Available in two pricing tiers

Azure Security Center Usage Scenarios

Scenario 1

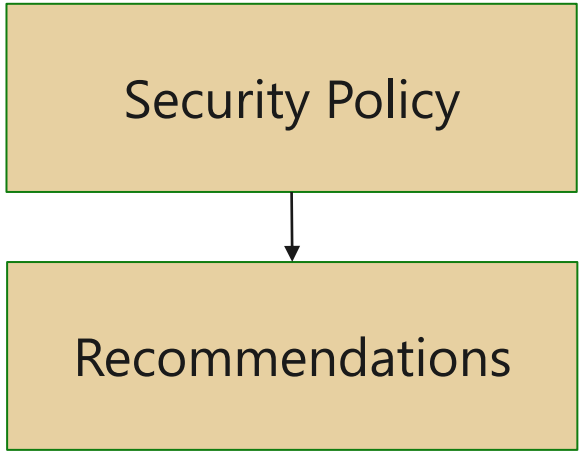
Use Security Center for an incident response



- **Detect** - Verify a high security alert was raised
- **Access -** Obtain information about the alert
- **Diagnose –** Follow the remediation steps

Scenario 2

Use Security Center recommendations to enhance security



- **Configure** a security policy
- **Implement** the recommendations for the security policy

Azure Policy

- Is a service in Azure that you use to create, assign, and, manage policies
- Provides enforcement by using policies and initiatives
- Runs evaluations on your resources and scans for those not compliant
- Comes with several built-in policy and initiative definitions
- Integrates with Azure DevOps by applying any continuous integration (CI) and continuous deployment (CD) pipeline policies

✔ An example of an Azure policy that you can integrate with your DevOps pipeline is the Check Gate task

Policies

- A *policy definition* expresses what to evaluate and what action to take
- Policies are defined in JSON
- You can assign policies in the Azure Portal, Azure CLI, or PowerShell
- Use remediation for non-compliant resources
- The screenshot is a section of a policy file defining allowed locations

```
        "displayName": "Allowed locations",
        "description": "This policy enables you to
restrict the locations your organization can specify
when deploying resources.",
        "policyRule": {
            "if": {
                "not": {
                    "field": "location",
                    "in": "
[parameters('allowedLocations')]"
                }
            },
            "then": {
                "effect": "deny"
            }
        }
    }
}
```

Initiatives

- An *initiative definition* is a set of policy definitions
- Helps track your compliance state for a larger goal
- Assigned to a specific scope
- Reduces the need for individual scope assignments

Initiative Example:

Create an Initiative named *"Enable Monitoring In Azure Security Center". This would p*rovide the following policies

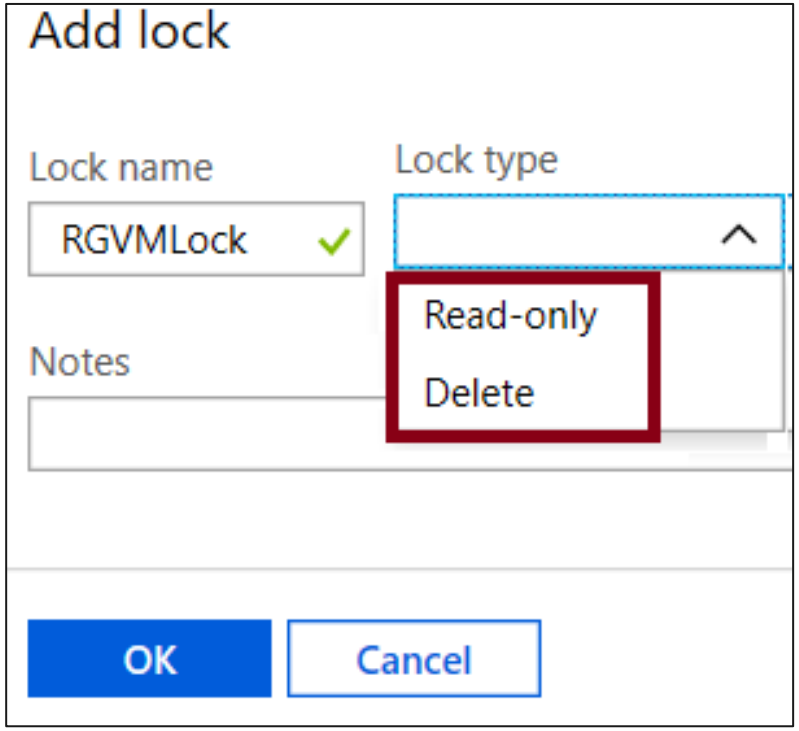
- *Monitor unencrypted SQL Database policy definition*
- *Monitor OS vulnerabilities policy definition*
- *Monitor missing Endpoint Protection policy definition*

✔ Even if you have a single policy, we recommend using initiatives if you anticipate increasing the number of policies over time

Locks

Prevent accidental deletion or modification of your Azure resources

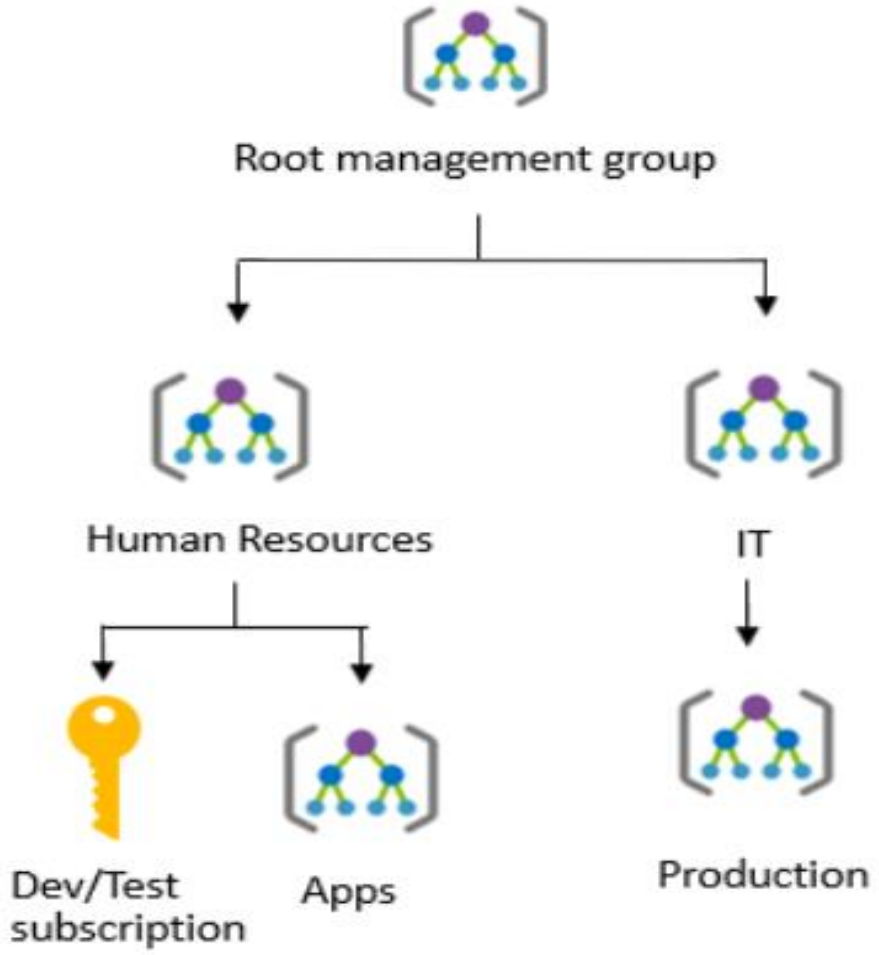
- **CanNotDelete -** means authorized users can still read and modify a resource, but they can't *delete* the resource
- **ReadOnly** - means authorized users can read a resource, but they can't delete or update it. Applying this lock is like restricting all authorized users to the permissions granted by the Reader role



In the Azure portal, the locks are called *Delete* and **Read-only** respectively

Subscription governance

- Considerations for creating and managing subscriptions:
  - Billing
  - Access Control
  - Subscription Limits
- Management groups manage access, policies, and compliance across multiple Azure subscriptions

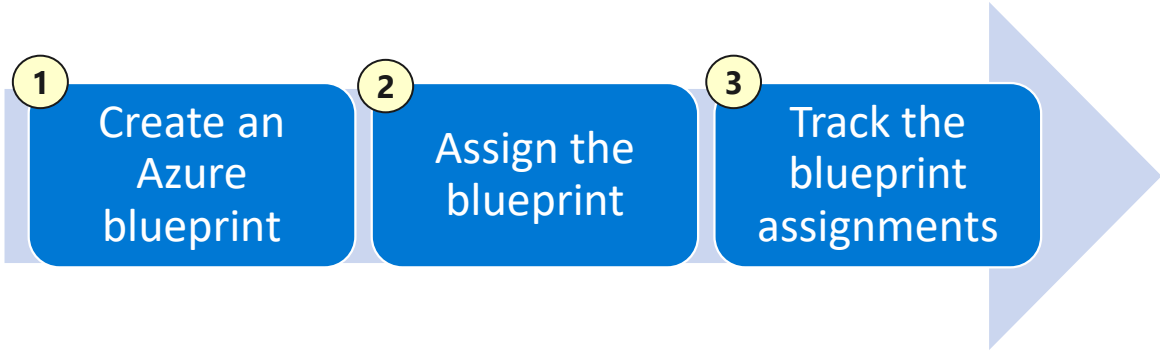


Azure Blueprints

Allows for the definition of a repeatable set of Azure resources that implement and adhere to an organization's standards, patterns, and requirements

A declarative way to orchestrate the deployment of various resource templates and other artifacts such as:

- Role Assignments
- Policy Assignments
- Azure Resource Manager templates
- Resource Groups



The blueprint definition (what should be deployed) and the blueprint assignment (what is deployed) supports improved deployment tracking and auditing

Azure Advanced Threat Protection (ATP)

Identifies, detects, and helps you investigate advanced threats, compromised identities, and malicious insider actions

- Azure ATP **portal** monitors and responds to suspicious activity
- Azure ATP **sensor** monitors domain controller traffic
- Azure ATP **cloud service** connects to Microsoft Intelligent Security Graph