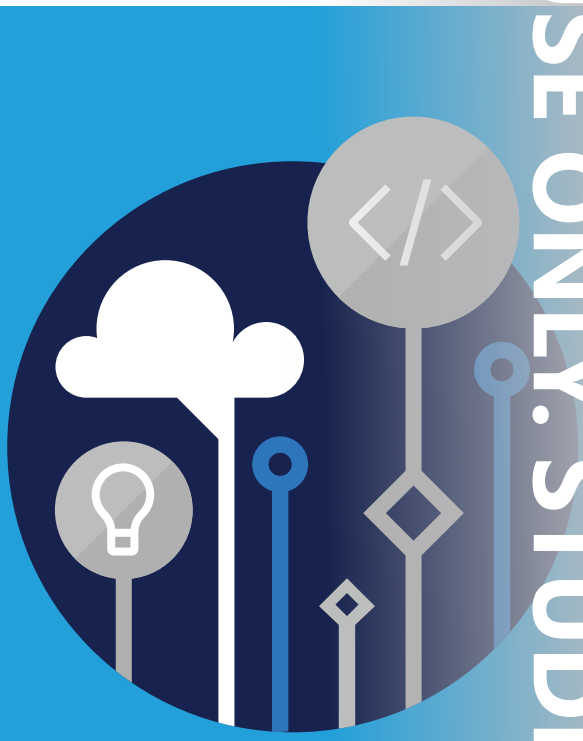
Microsoft

Microsoft Official Course



AZ-301T03

Designing for Deployment, Migration, and Integration

AZ-301T03

Designing for Deployment, Migration, and Integration

Contents

Module 0 Welcome

Start Here

Welcome to Designing for Deployment, Migration, and Integration

Welcome to Designing for Deployment, Migration, and Integration. This course is part of a series of four courses to help students prepare for Microsoft's Azure Solutions Architect technical certification exam AZ-301: Microsoft Azure Architect Design. These courses are designed for IT professionals and developers with experience and knowledge across various aspects of IT operations, including networking, virtualization, identity, security, business continuity, disaster recovery, data management, budgeting, and governance.

This course contains the following three modules:

Module 1 - Deploying Resources with Azure Resource Manager

This module establishes a basic understanding of Azure Resource Manager and the core concepts of deployments, resources, templates, resource groups, and tags. The module will dive deeply into the automated deployment of resources using ARM templates.

After completing this module, students will be able to:

- Create a resource group.
- Add resources to a resource group.
- Deploy an ARM template to a resource group.
- Filter resources using tags.
- Author a complex deployment using the Azure Building Blocks tools.

Module 2 - Creating Managed Server Applications in Azure

This module describes how solutions can leverage serverless application hosting services in Azure to host web applications, REST APIs, integration workflows and HPC workloads without the requirement to manage specific server resources. The module focuses on App Services-related components such as Web Apps, API Apps, Mobile Apps, Logic Apps, and Functions.

After completing this module, students will be able to:

- Select between hosting application code or containers in an App Service instance.
- Describe the differences between API, Mobile, and Web Apps.
- Integrate an API or Logic App with the API Management service.
- Design an App Service Plan or multi-region deployment for high performance and scale.

Module 3 - Authoring Serverless Applications in Azure

This module describes how solutions can leverage serverless application hosting services in Azure to host web applications, REST APIs, integration workflows and HPC workloads without the requirement to manage specific server resources. The module focuses on App Services-related components such as Web Apps, API Apps, Mobile Apps, Logic Apps, and Functions.

After completing this module, students will be able to:

- Select between hosting application code or containers in an App Service instance.
- Describe the differences between API, Mobile, and Web Apps.
- Integrate an API or Logic App with the API Management service.
- Design an App Service Plan or multi-region deployment for high performance and scale.
- Create a resource group.
- Add resources to a resource group.
- Deploy an ARM template to a resource group Integrate an API or Logic App with the API Management service.
- Design an App Service Plan or multi-region deployment for high performance and scale.
- Integrate an API or Logic App with the API Management service.
- Design an App Service Plan or multi-region deployment for high performance and scale.

Prerequisites

This course requires that students have the following knowledge and skills:

- Create resources and resource group in Azure.
- Manage users, groups, and subscriptions in an Azure Active Directory instance.
- Build an Azure Virtual Machine with related resources.
- Manage containers and blobs stored in an Azure Storage account.
- Create App Service Plans and manage apps related to the plan.
- Configure an Azure Virtual Network and enable S2S and P2S connectivity.
- Protect networked application components using Network Security Groups.
- Automate everyday Azure resource tasks using Azure CLI or Azure PowerShell.
- Deploy an Azure SQL, MySQL, Postgres or Cosmos database instance.
- Monitor existing Azure solutions using built-in metrics, Application Insights, or Operational Insights.

Module 1 Module Deploying Resources with Azure Resource Manager

ARM Templates

Azure Resource Manager

This module establishes a basic understanding of Azure Resource Manager and the core concepts of deployments, resources, templates, resource groups, and tags. The module will dive deeply into the automated deployment of resources using ARM templates.

After completing this module, students will be able to:

- Create a resource group
- Add resources to a resource group
- Deploy an ARM template to a resource group
- Filter resources using tags
- Author a complex deployment using the Azure Building Blocks tools

ARM Templates

Azure has developed a great deal over the last few years, and with new services being released on a regular basis, there was a need to create a way to manage and deploy resources in a componentized and reusable manner. Azure Resource Manager was designed to represent each service in Azure as a resource provider and each service instance in Azure as a modular resource. With Azure Resource Manager, JSON templates were used to deploy collections of resources using Infrastructure-as-Code concepts. Along with Azure Resource Manager, we saw that release of a new Azure Portal (**https://portal.azure.com**) that focused on the modular nature of resources. This lesson focuses on Azure Resource Manager (ARM) and how you can use JSON to deploy resources using ARM templates.

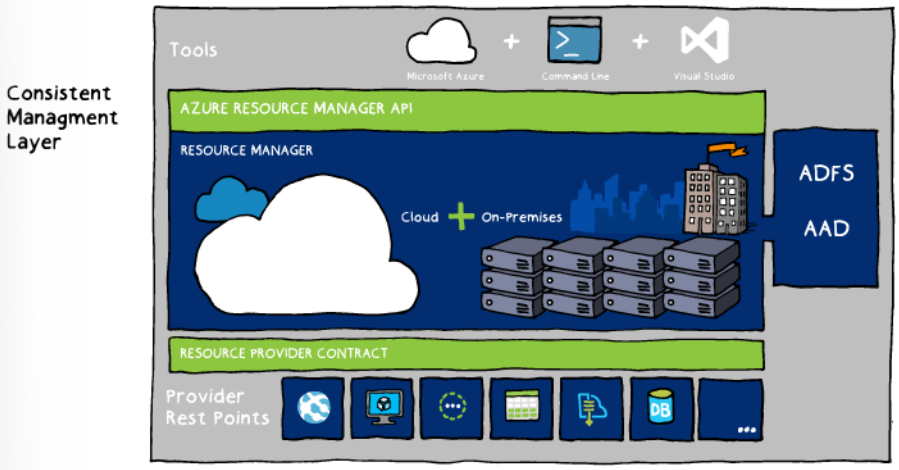
After completing this section you will be able to:

- Describe ARM Templates
- Understand the JSON format and how to author a JSON template in various ways

- Describe the format of a JSON file and where to obtain example templates for Azure deployments
- Deploy a resource using the Azure Quickstart templates on GitHub

Azure Resource Manager (ARM) is the latest way to conceptualize your Azure resources in your subscription. Service instances are now resources, which are grouped as resource groups. Resource groups provide a common lifecycle for the child resources. They can be created, managed, monitored, or deleted together. The Resource Manager also offers the concept of resource group templates which enable you to define a service unit in advance, and then use the template to create as many resource groups as you need.

Resource groups and resource group templates are ideal for developer operations scenarios where you need to quickly build out development, test, quality assurance, or production environments that are homogenous in nature and can be managed with a shared lifecycle. Developers can quickly delete their environment and create a new environment by using the shared template. The resource groups can be monitored to determine the billing rate or resource usage at a higher level than monitoring individual service instances.



Before using ARM, you need to consider resource providers and ensure your subscription contains registrations for each provider you wish to use. Deploying a complex Virtual Machine template will fail at the first hurdle if you are not registered for the **Microsoft.Compute** provider. This provider controls access to all Virtual Machine creation.

Azure Resource Manager Objects

When you envision your solution using ARM, you must start by designing and conceptualizing your entire solution considering all components that may compose your solution. Once you have designed your entire solution, you can then identify individual units of functionality and find resources available on Azure that can facilitate the specific functionalities.

You use a template—a resource model of the service—to create a resource group with the resources you specified above. After you author the template, you can manage and deploy that entire resource group as a single logical unit. There are three primary concepts in Resource Manager:

- **Resource**: A resource is merely a single service instance in Azure. Most services in Azure have a direct representation as a resource. For example, a Web App instance is a resource. An App Service Plan is also a resource. Even a SQL Database instance is a resource.

• **Resource Group**: A resource group is a group of resources in the logical sense. For example, a Resource Group composed of a Network Interface Card (NIC), a Virtual Machine compute allocation, a Virtual Network, and a Public IP Address creates what we would logically consider a "Virtual Machine."

• **Resource Group Template**: Every resource group deployment is completed using a JSON file known as the resource group template. This JSON file declaratively describes a set of resources. The deployment adds the new resources to a new or existing resource group. For example, a template could contain the configuration necessary to create 2 API App instances, a Mobile App instance and a Cosmos DB instance.

Interacting with Resource Manager

You can use Resource Manager in a variety of different ways including:

• **PowerShell**: There are PowerShell CmdLets already available to allow you to manage your services in the context of resources and resource groups.

• **Cross-Platform Command-Line Interface:** This CLI allows you to manage your Azure resources from many different operating systems.

• **Client Libraries**: There are client libraries already available for various programming frameworks/languages to create resources and resource groups in Azure.

• **Visual Studio**: Visual Studio 2015 ships with a Resource Manager project type that allows you to create a resource group template by either manually modifying JSON (with schema intellisense) or use scaffolding to update your JSON template automatically.

• **Portal template deployment**: In the portal, you can use the Template Deployment option in the Marketplace to deploy a Resource Group from a template.

• **REST API**: All the above options use the REST API to create your resources and resource groups. If you prefer to create resources without a library, you can always use the REST API directly.

Reference Link: **https://docs.microsoft.com/rest/api/resources/**

Azure Resource Manager Documentation: **https://docs.microsoft.com/en-us/azure/azure-resource-manager/**

ARM Templates

Azure supports the deployment of resources using Azure Resource Manager templates. In an Azure Resource Manager template, you specify the resource—for example, a virtual network, virtual machine, and so on—using a JSON object. Some or all of the properties of the resource can be parameterized so that you can customize your deployment by providing parameter values at deployment time.

- ARM Templates are deployed in a few ways. These depend on your aims, the result intended and your chosen method for development.
- A developer may choose to use Visual Studio to create and deploy ARM templates directly and to manage the lifecycle of the resources through Visual Studio.

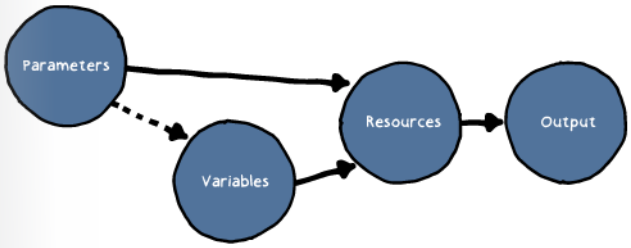
- An administrator may choose to use PowerShell or the Azure Command Line to deploy resources and amend them.
  An end user without command line or developer skills would choose to use the Azure Portal to deploy resources without realizing a template is involved. This deployment would typically be a less complicated group of resources unless the user chooses to deploy a marketplace offering.

Advantages Of Using Templates

Templates are preferred to manually deploying resources for the following:

- **Ensure idempotency**: If you deploy an identical template to multiple resource groups, they would functionally be the same.
- **Simplify orchestration**: only need to deploy the template to deploy all of your resources. Normally this would take multiple operations.
- **Configure multiple resources**: You can configure multiple resources simultaneously and use variables/parameters /functions to create dependencies between resources. For example you can require that a VM is created before a Web App because you need the VM's public IP address for one of the Web App's settings. Additionally, you can require a Storage account is created before a VM so that you can place the VHDs in that storage account.
- **Parameterize**: You can parameterize input and output values so they can be reused across many different scenarios. Templates can also be nested so you can reuse smaller templates as part of a larger orchestration.

Template Resources



JSON

What is JSON?

JavaScript Object Notification (JSON) is a lightweight formatted script designed for data transfer. Azure Resource Manager uses this as the foundation for every resource.

Empty ARM Template

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
  },
  "variables": {
  },
  "resources": [
  ],
  "outputs": {
```

```
  }
}
```

The format of JSON is flexible in that several elements are optional.

The script must have a schema and a content section as well as a resource section. The other three, parameters, variables and output are optional.
In day to day use, most Arm Template JSON contains all sections and even contains logic as well.

Resources can depend on other resources existing and even their names, locations and content. This resource dependency can lead to very complex nested JSON templates when building a sophisticated resource group, such as a SharePoint farm.

The Azure Quickstart templates on Github are an excellent resource for learning to structure and deploy your templates correctly. If you are just beginning on your JSON journey, then the Automation script element of every object in the Azure Portal allows you to learn the constructs and formatting of JSON.

Role-Based Access Control (RBAC)

Role-Based Access Control

Role-Based Access Control (RBAC)

Azure role-based access control (RBAC) leverages existing Azure AD user, groups, and services to create a relationship between those identities and components of your Azure subscription. Using RBAC, you can assign roles to existing Azure AD identities that grants them pre-determined levels of access to an Azure subscription, resource group or individual resource.

This lesson will discuss the best practices and possibilities of using Azure AD RBAC to manage access to your resources, resource groups, and application.

After completing this section, you will be able to:

• Describe Azure AD RBAC features.

• Decide how to provide access to your resources using RBAC.

• Allocate Roles to users and groups to provide access to resources.

• Deploy a custom role to an Azure resource group.

Azure role-based access control allows you to grant appropriate access to Azure AD users, groups, and services, by assigning roles to them on a subscription or resource group or individual resource level. The assigned role defines the level of access that the users, groups, or services have on the Azure resource.

Roles

A role is a collection of actions that can be performed on Azure resources. A user or a service is allowed to act on an Azure resource if they have been assigned a role that contains that action. There are built-in roles that include (but is not limited to):

ROLE NAME	DESCRIPTION
Contributor	Contributors can manage everything except access.
Owner	Owner can manage everything, including access.
Reader	Readers can view everything, but can't make changes.
User Access Administrator	Allows you to manage user access to Azure resources.
Virtual Machine Contributor	Allows you to manage virtual machines, but not access to them, and not the virtual network or storage account they are connectedto.

Role Assignment

A role assignment can be created that associates a security principal to a role. The role is further used to grant access to a resource scope. This decoupling allows you to specify that a specific role has access to a resource in your subscription and add/remove security principals from that role in a loosely connected manner. Roles can be assigned to the following types of Azure AD security principals:

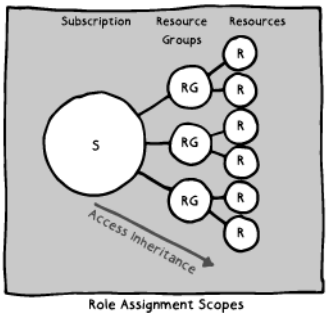
• **Users**: Roles can be assigned to organizational users that are in the Azure AD with which the Azure subscription is associated. Roles can also be assigned to external Microsoft accounts that exist in the same directory.

• **Groups**: Roles can be assigned to Azure AD security groups. A user is automatically granted access to a resource if the user becomes a member of a group that has access. The user also automatically loses access to the resource after getting removed from the group. Managing access via groups by assigning roles to groups and adding users to those groups is the best practice, instead of assigning roles directly to users.

• **Service principals**: Service identities are represented as service principals in the directory. They authenticate with Azure AD and securely communicate with one another. Services can be granted access to Azure resources by assigning roles via the Azure module for Windows PowerShell to the Azure AD service principal representing that service.

Resource Scope

Access does not need to be granted to the entire subscription. Roles can also be assigned to resource groups as well as for individual resources. In Azure RBAC, a resource inherits role assignments from its parent resources. So if a user, group, or service is granted access to only a resource group within a subscription, they will be able to access only that resource group and resources within it, and not the other resources groups within the subscription. As another example, a security group can be added to the Reader role for a resource group, but be added to the Contributor role for a database within that resource group.

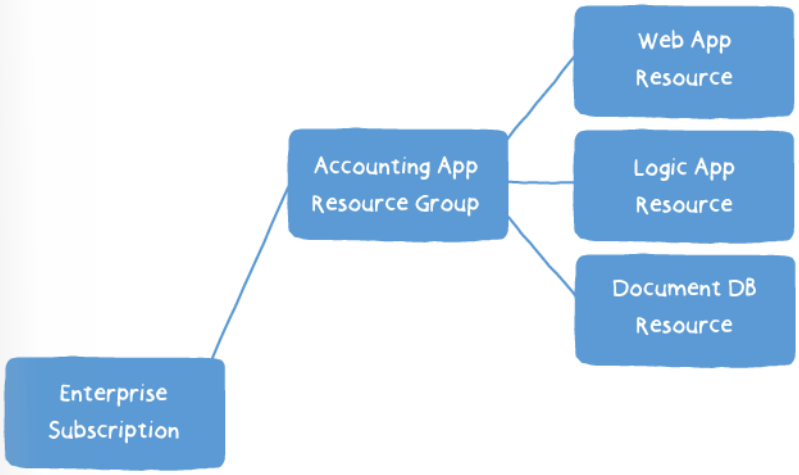


Role Assignment Scopes

Scoping to Resource Groups

While RBAC can easily be used to grant access to an individual resource, it is preferable to grant access to an entire Resource Group as opposed to individual resources. By scoping to a resource group, you can add/remove and modify resources quickly without having to recreate assignments and scopes. You can also give an individual owner or contributor access to a resource group so that they can create, recreate or destroy resources on their own without requiring involvement from the account administrator.

Let's explore a resource group for an actual application. In the example below, there is a resource group for an accounting application. this resource group contains multiple resources that are used in the application.



By granting an individual owner or contributor access to the resource group, they can configure resources, create new deployments, add new resources or create automation scripts at will without requiring additional administrator assistance or having access to resources in other resource groups.

Custom Roles

If there is no role suitable for your purposes or granular enough to suit your needs, it is possible to create a custom role and apply that. To create a custom role, you must use either Azure PowerShell or the Azure Command-Line Interface.

It is also possible to use the REST API to create roles programmatically. Each Azure AD tenant is limited to 2000 custom roles.

To create a new custom role you run the **New-AzureRmRoleDefinition** cmdlet; you can pass a JSON template to the cmdLet or use **PSRoleDefinitionObject**.

JSON required for a new Custom Role

```
{
  "Name": "New Role 1",
  "Id": null,
  "IsCustom": true,
  "Description": "Allows for read access to Azure storage and compute resources",
  "Actions": [
    "Microsoft.Compute/*/read",
    "Microsoft.Storage/*/read",

  ],
  "NotActions": [
  ],
  "AssignableScopes": [
    "/subscriptions/c489345-9cd4-44c9-99a7-4gh6575315336g"
 ]
}
```

If you save the JSON to a file such as C:\CustomRole\newrole1.json, you can use the following PowerShell to add the custom role to a subscription.

Command to add the Custom role to the Subscriptionin PowerShell

```
New-AzureRmRoleDefinition -InputFile "C:\CustomRole\newrole1.json"
```

Resource Policies

Azure Resource Policies

The use of RBAC controls users access to resources. Resource Policy is a preview service that controls which resources can be created and what for they take, such as naming conventions, locations, and sizes. This lesson will describe the features available and show the ways of defining, assigning and monitoring those policies.

After completing this section, you will be able to:

- Describe Azure Policy service.
- Decide when to apply policy definitions and to what scope.
- Author an Azure Policy definition.
- Assign and monitor an Azure resource policy.

Azure Resource Policies

Azure Policy is a new service designed to allow an organization to ensure that resources created in the cloud comply with corporate standards and service level agreements. As an example, preventing users from creating resources in specific locations and of specific types and sizes. If your organization has no need for a Virtual Machine with 32 CPU cores and 500GB of RAM in the more expensive UK South region, then this can be monitored and prevented with Azure Policy.

An Azure policy contains two elements, a policy definition, and a policy assignment. This design allows an organization to create a library of policy definitions to be assigned later. There are many pre-defined policy definitions built into Azure.

Policy vs. RBAC

There are differences between policy and role-based access control (ABC). RBAC controls user access, permissions, privileges, and actions at different scopes. As an example, you could add a user to the Owner role for a resource group as the desired scope. The Owner role gives you full control of that resource group.

Policy evaluates resource properties for already existing resources and during deployment. As an example, using policies, you can control which type of resource is available for deployment in your organization. Other policies can limit the locations in which you deploy resources or require your resources to follow naming conventions.

Azure policy is a default allow, and explicit deny system. This is not the same as RBAC.
To be able to create and assign Azure policies, you need to be assigned permissions using RBAC; the contributor role does not contain the necessary permissions.

The permissions required are:

- **Microsoft.Authorization/policydefinitions/write** permission to define a policy.
- **Microsoft.Authorization/policyassignments/write** permission to assign a policy.

Built-In Policies

Azure has a built-in library of Policies to allow organizations to control their resources. These include:

- Allowed locations
- Allowed resource types
- Allowed storage account SKUs
- Allowed virtual machine SKUs
- Apply tag and default value
- Enforce tag and value
- Not allowed resource types
- Require SQL Server version 12.0
- Require storage account encryption

The inclusion of these built-in policy definitions limits the number a user is required to create to manage their subscription efficiently.

Policy Definition

Every policy definition contains a minimum of two elements, first the conditions under which it is enforced. Secondly, it has an action that triggers if the conditions are met.

Policy definitions are created using JSON. A policy definition contains elements to define:

- **mode**
- **parameters**
- **display name**
- **description**
- **policy rule**
- **logical evaluation**
- **effect**

The following example shows a policy that defines not allowed resources (This is a built-in policy definition).

policy not allowed.json

```
{
  "if": {
    "field": 'type",
    "in": "[parameters('listOfResourceTypesNotAllowed')]"
  },
  "then": {
    "effect": "Deny"
  }
}
```

During assignment, the list of resource types not allowed parameter is populated and evaluated against all current resources in scope for the assignments. Any non-compliant resources will be identified. When creating new resources, the deployment will fail if they are non-compliant. The above is a straightforward policy definition. The JSON for policy definitions can be much more complicated.

The definition can contain multiple if, then blocks and combines logical operators, conditions, and fields to build out the policy. In addition, the policy can define alternative effects.

The available effects are:

- **Deny**
- **Audit**
- **Append**
- **AuditIfNotExists**
- **DeployIfNotExists**

You can assign any of these policy definitions through the Azure portal, PowerShell, or Azure CLI.

Policy Assignment

Once a policy definition has been created, this is known as a custom policy definition and can be assigned to take effect over a specific scope. This scope could range from several subscriptions within an enterprise (known as a management group) to a resource group. A scope is the list of all the resource groups, subscriptions, or management groups that the policy definition is assigned to.

If a policy is assigned to a resource group, the same policy applies to all the resources contained in that group. Policy assignments are inherited by all child resources. This can be altered or prevented using the exclusion option at the time of assignment. As an example, a policy assignment could prevent the creation of SQL databases within your subscription. An exclusion could allow this in a single Resource group, which you want to keep as your database Resource Group. RBAC is then used to restrict access to the SQL database resource group to the trusted group of users that have the necessary skills and requirements for these resources.

A policy definition will create parameters to be applied at the time of Policy assignment. This allows for one definition to be re-used across several resource groups, locations, and subscriptions.

The scope, exclusions, and parameters of the Not Allowed resource types are shown in the policy assignments blade when assigning a definition.
Azure policies can be assigned using the Azure Portal, Azure PowerShell, and the Azure Command Line Interface.

Initiative Definition

An initiative definition is a collection of policy definitions that are designed to achieve a design goal. Initiative definitions contain policy definitions and can be assigned as a group to simplify the process of achieving that goal. The definition appears as a single item rather than multiple definitions.

Policies for Naming Conventions

Azure policies can be used to maintain control on the naming conventions for all resources within your subscriptions.
A policy definition can be created to enforce naming conventions. These can use wildcards, patterns tags, and multiple patterns to apply restrictions to your Azure resource names.

The example below applies a pattern match asserting that is the resource name does not begin Contoso and have six characters as the rest of the name then it will be non-compliant.

Example Naming Pattern

```
{
  "if": {
    "not": {
      "field": "name",
      "match": "contoso??????"
    }
  },
  "then": {
    "effect": "deny"
  }
}
```

The effect of non-compliance is deployment failure on the creation of a resource or listing in the non-compliant reporting if the resource is already in existence.

Security

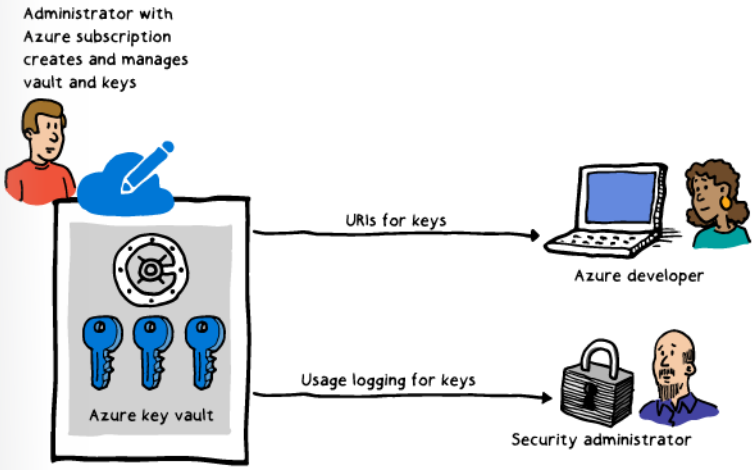
Azure Key Vault

When deploying resources using Arm templates and automating that deployment, it is best practice to use a Service Principal, the Azure equivalent of an Active Directory Service Account. This removes the risk whereby an administrator account is stored and used to deploy resources. To facilitate this in a secure manner, Azure Resource Manager can use the Azure Key Vault to store the service principal secrets. This lesson describes the Key Vault and its use in deploying Arm templates securely.

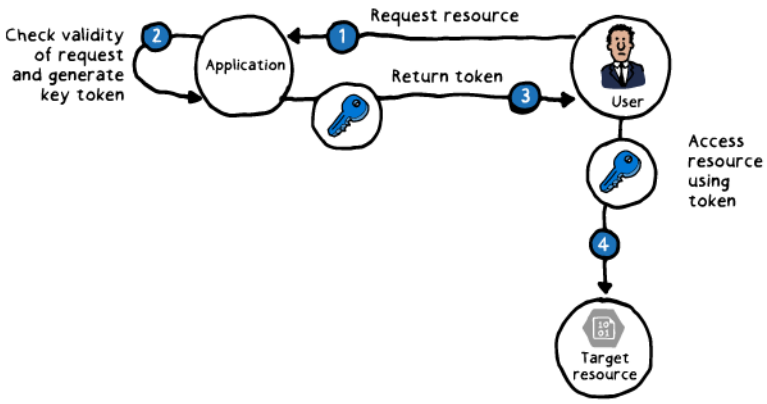
After completing this section you will be able to:

- Describe the Azure Key Vault service.
- Describe the secure deployment of templates using Service Principals and the Key Vault.
- Create a Service Principal.
- Deploy a resource ARM templates and the Azure Key Vault.

Azure Key Vault is a two-tier service that allows secure storage of cryptographic keys, certificates, and secrets. These can be used by applications, services, and users. The premium tier allows storage of these secrets in a Hardware Security Module, a physical device to contain all your secrets.



Key Vault makes the process of creation, importing, maintaining and recycling secrets much easier. The developer can easily create dev and test keys that can be migrated to production use at deployment. Security administrators can manage all the secrets in the Key Vault with ease and grant and revoke access when required. Key Vault can be used for several scenarios in Azure.



App developers want to use keys for signing and encryption but want all keys outside of the application. App developers do not want to manage keys but wants their customers to bring their own keys and certificates.

Security admins want to ensure that company applications comply with regulatory bodies requirements such as FIPS. Key Vaults can be created in any Azure subscription by any contributor or Owner. Key Vaults can be used to create, manage and import secrets, keys, and certificates for applications, services and users.

Key Vault Use in ARM Templates

When deploying resources into Azure using ARM Templates, a user credential is often required to allow the resources to be created. Best practices dictate that embedding credentials and passwords inside a template are unwise. Key Svault can be used to pass a secure value as a parameter during deployment. The secure value, in this instance a password can be retrieved from the Key Vault, retrieval is accomplished by referencing both the key vault id and the secret in the parameter file for the template. The benefit is that the value of this secret is never exposed. To further secure the deployment, it is advised to create an Azure Service Principal, which is the equivalent of an Active Directory Service Account. This reduces risk by allowing you to limit the privileges of the Service Principal to only what is needed.

The value is never exposed because you only reference its key vault ID. At deployment time you are not required to enter credentials. There are several steps required to allow a Key Vault to be used during template deployment. First, it is essential to set the **enabledForTemplateDeployment** property to **true** when you create the Key Vault. This setting allows access to the key Vault from Resource manager at deployment time.

Next, you must create a secret using the Azure Portal, PowerShell or Azure CLI. Having created a secret for use by the template you must give the template access to the vault and the secret. This is achieved by ensuring the service principal, user or template has the **Microsoft.KeyVault/vaults/deploy/action** permission for the correct Key Vault. The Contributor built-in role already has this permission. The last step is to reference the secret using a static ID in the template parameter file. Sometimes the Key Vault secret will change from deployment to deployment; this requires the use of a dynamic ID reference which cannot go in the parameter file and calls for a nested template to achieve this.

Building Blocks

Azure Building Blocks

While authoring ARM templates can give you the most flexibility when automating deployments, ARM templates can become very complex in a short amount of time. Additionally, it may be difficult to ensure that Microsoft best practices for Azure infrastructure are reflected in every template authored by your team.

This section introduces the Azure Building Blocks, an open-source tool and repository to help simplify the authoring and deployment of ARM Templates. This tool includes ARM templates that already reflect best practices as prescribed by the Patterns & Practices team at Microsoft.

After completing this section you will be able to:

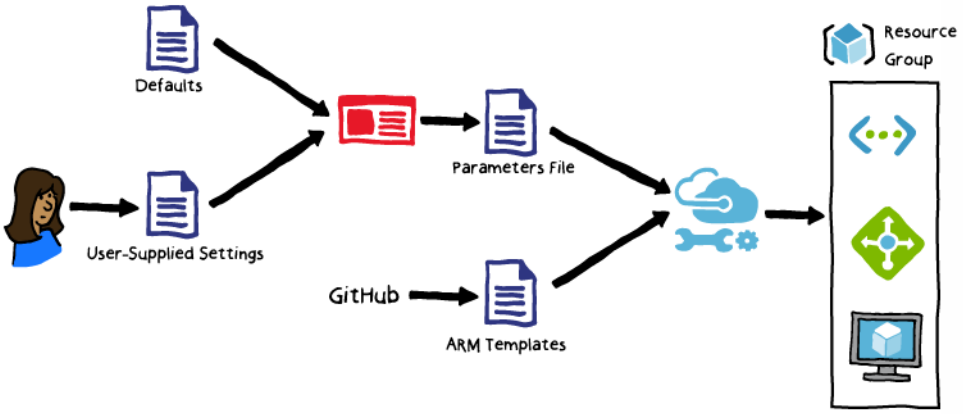
- Describe the Azure Building Blocks and how they fit into a deployment automation solution.
- Decide when to author an ARM template manually or when to use the Azure Building Blocks.
- Author an Azure Building Blocks parameters file.
- Deploy a resource using the Azure Building Blocks command-line tool.

Azure supports the deployment of resources using Azure Resource Manager templates. In an Azure Resource Manager template, you specify the resource—for example, a virtual network, virtual machine, and so on—using a JSON object. Some or all of the the properties for the resource can be parameterized so that you can customize your deployment by providing parameter values at deployment time.

If you have spent time developing and maintaining Azure Resource Manager templates, you may have noticed that development of your templates follows a pattern. First you specify the JSON object for a virtual network, then you specify the JSON object for virtual machines that are deployed into the virtual network. Then you specify a JSON object for a network security group to secure the virtual network, and you might specify the JSON object for a load balancer to distribute incoming requests to the virtual machines. Perhaps you have parameterized some of the property values so you can customize the deployment.

While Azure Resource Manager templates are very powerful and allow you to deploy very large and complex architectures to Azure, they require a great deal of knowledge about Azure Resource Manager and the resources themselves. This leads to difficulty maintaining your templates because any modification can lead to unforeseen issues.

The Azure Building Blocks project solves this problem by providing a command line tool and set of Azure Resource Manager templates designed to simplify deployment of Azure resources. You specify settings for Azure resources using the Building Blocks JSON schema, and the command line tool merges these settings with best practice defaults to produce a set of parameter files. The command line tool deploys these parameter files using a set of pre-built Azure Resource Manager templates. Below is an example of Azure Building Blocks workflow:



The Building Blocks JSON schema is designed to be flexible. You can either specify your resources settings in one large file or several small files.

The Template Building Blocks currently support the following resource types:

- Virtual Networks
- Virtual Machines (including load balancers)
- Virtual Machine Extensions
- Route Tables
- Network Security Groups
- Virtual Network Gateways
- Virtual Network Connection

Deploying Resources Using Building Blocks

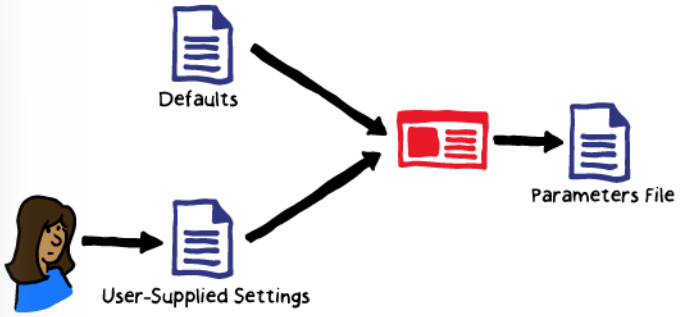
To demonstrate how to use the Azure Building Blocks, we will use the tool to deploy a simple virtual network. This example assumes the **azbb** command line tool and the Azure CLI are both installed on your local machine.

To get started with the building blocks, you must first create a JSON file with a **buildingBlocks** parameter containing an array of resources that you wish to deploy. Each resource within that JSON array can have certain options configured.

For example, a Virtual Network resource would have options such as **addressPrefix**, subnets and name. Any options you do not specify are set using the default options specified n the building blocks repository.

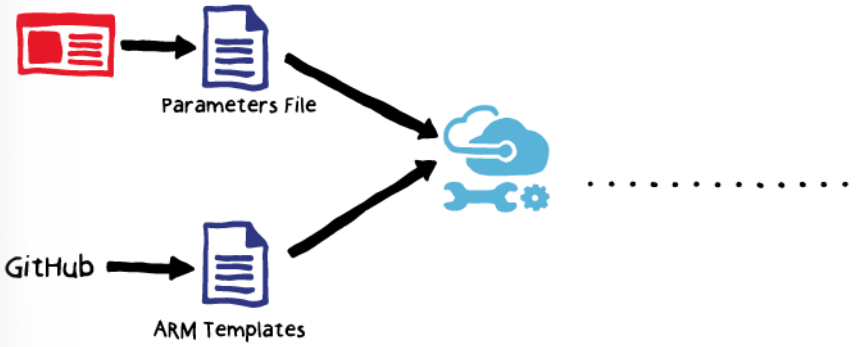
These default options are commonly Microsoft recommended best practices. Once the user-supplied settings file is created, run the **azbb** command line tool to parse your settings and combine it with the default settings from the building blocks repository. The **azbb** command line tool will output an ARM template parameters file that will be used by the tool later in deployment.

We'll begin our architecture with a simple virtual network, where we create a parameters file:

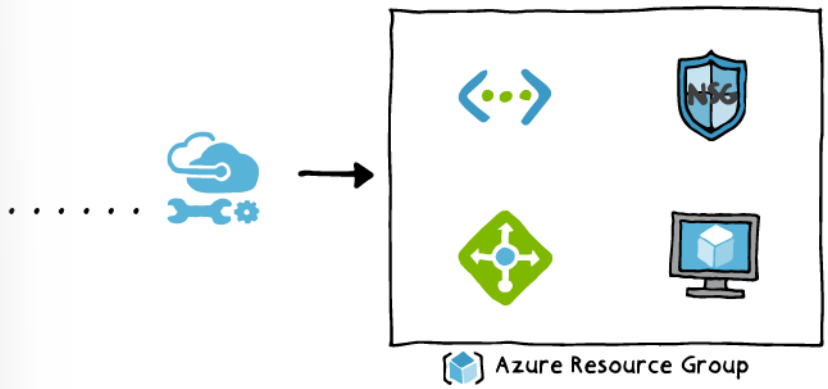


Once the azbb tool has created an ARM template parameters file, the **azbb** tool will use the Azure CLI (version 2.0) to deploy the master ARM template from the building blocks repository using the parameters file that was generated.

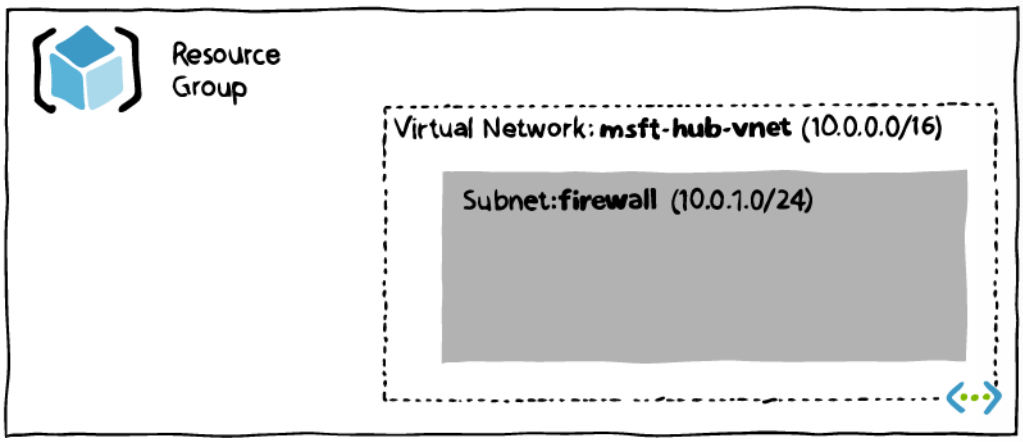
The template and parameters are sent to Azure and a deployment is created.



The Azure deployment will create a collection of resources within an Azure Resource Group. These resource group will have historical deployment information so you can go back and view the template and parameters used by the building blocks command line tool to deploy your resources.



SAMPLE RESOURCE GROUP



The architecture includes the following:

• A VNet named msft-hub-vnet with an address space of **10.0.0.0/16**.

• A subnet within msft-hub-vnet named firewall with an address space of **10.0.1.0/24**.

Building Block Resource

Every block can be represented as a JSON object. For our example, we will use the following JSON object to represent our simple Virtual Network.

The type property is used by the Azure Building Blocks to identify the type of building block. We're going to deploy a VNet, so we have to specify a VirtualNetwork building block type. This property is required for each of the building blocks.

Every Azure Building block also requires a settings object where the properties for the building block are specified.

Let's look at each property for a simple VNet:

• **Name**: In Azure, each resource requires a name to uniquely identify the resource within a resource group. In Azure Building Blocks, you specify a name property for each resource type to provide this unique name. When we deploy this settings file using the command line tool, this is the name that we'll see in the Azure portal user interface. In this settings file we've named the VNet msft-hub-vnet because this in future tutorials this will become the central "hub" VNet for our complex architecture.

• **addressPrefixes**: Next, we specify the address space for our virtual network using the **addressPrefixes** property. The address space is specified using CIDR notation. In our example settings file, we've specified the address space to be 10.0.0.0/16. This means Azure Resource Manager allocates 65536 IP addresses beginning at 10.0.0.0 and ending at 10.0.255.255.

Notice that the field for specifying the virtual network address space is an array. The reason for this is because we can specify multiple address ranges. For example, in addition to 10.0.0.0/16 we could have also specified 11.0.0.0/16 to specify everything between 11.0.0.0 and 11.0.255.255 as well:

```
"addressPrefixes": [
"10.0.0.0/16",
"11.0.0.0/16"
]
```

• **subnets**: Now that we have specified the address space for our virtual network, we can begin to create named network segments known as subnets. Subnets are used to manage security, routing, and user

access for each subnet independently of the entire VNet. Subnets are also used to segment VMs into back-end pools for load balancers and application gateways.

As you can see from our settings file, we've specified a single subnet named firewall with an address space of 10.0.1.0/24. Note that the subnets property is also an array—we can specify up to 1,000 subnets for each VNet.

Settings File

To deploy this virtual network, we will need to create a settings file for the azbb command line tool. An empty settings file looks like this:

```
{
  "$schema": "https://raw.githubusercontent.com/mspnp/template-building-blocks/master/schemas/
buildingBlocks.json",
  "contentVersion": "1.0.0.0",
  "parameters" : {
    "buildingBlocks": {
      "value": [
        {}
      ]
    }
  }
}
```

In the example above, the **buildingBlocks** array is empty. For our deployment, we will add the simple Virtual Network building block that we discussed earlier in this topic. The settings file for our deployment would look like this:

```
{
  "$schema": "https://raw.githubusercontent.com/mspnp/template-building-blocks/master/schemas/
buildingBlocks.json",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "buildingBlocks": {
      "value": [
        {
          "type": "VirtualNetwork",
          "settings": [
            {
              "name": "msft-hub-vnet",
              "addressPrefixes": [
                "10.0.0.0/16"
              ],
              "subnets": [
                {
                  "name": "firewall",
                  "addressPrefix": "10.0.1.0/24"
                }
              ]
            }
```

```
        ]
      }
    ]
  }
 }
}
```

Deploy a Settings File

To deploy your settings file, you will need to ensure that you are logged in to the Azure CLI. Then, you'll need a few things before you can deploy the VNet using the settings file:

• You need your Azure subscription ID. You can find your subscription ID by using the Azure CLI command **az account list**, or, by going to the Azure Portal and opening the subscriptions blade.

• You'll need to consider the resource group to which the VNet will be deployed. You can deploy to either an existing or new resource group. The Azure Building Blocks command line tool determines if the resource group name you pass with the **-g** option exists or not. If the resource group exists, the command line tool deploys the VNet to the existing resource group. If it doesn't exist, the command line tool creates the resource group for you and then deploys the VNet to the new resource group.

• You'll also need to consider the Azure region where the VNet will be deployed.

Once you have your settings file and the information listed above, you can create a deployment using the following command:

azbb -g <new or existing resource group> -s <subscription ID> -l <region> -p <path to your settings file> --deploy

The command line tool will parse your settings file and deploy it to Azure using Azure Resource Manager. To verify that the VNet was deployed, visit the Azure Portal, click on **Resource Groups** in the left-hand pane to open the **Resource Groups** blade, then click on the name of the resource group you specified above. The blade for that resource group will open, and you should see the **msft-hub-vnet** in the list of resources.

Online Lab - Getting Started with Azure Resource Manager Templates and Azure Building Blocks

Lab Steps

Online Lab: Getting Started with Azure Resource Manager Templates and Azure Building Blocks

NOTE: For the most recent version of this online lab, see: **https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign**

Before we start

1. Ensure that you are logged in to your Windows 10 lab virtual machine using the following credentials:
   - Username: **Admin**
   - Password: **Pa55w.rd**
2. Review Taskbar located at the bottom of your Windows 10 desktop. The Taskbar contains the icons for the common applications you will use in the labs:
   - Microsoft Edge
   - File Explorer
   - **Visual Studio Code**[1]
   - **Microsoft Azure Storage Explorer**[2]
   - Bash on Ubuntu on Windows
   - Windows PowerShell
3. **Note**: You can also find shortcuts to these applications in the **Start Menu**.

1 https://code.visualstudio.com/
2 https://azure.microsoft.com/features/storage-explorer/

3

Exercise 1: Deploy core Azure resources by using an Azure Resource Manager Template from the Azure portal

4

Task 1: Open the Azure Portal

1. On the Taskbar, click the **Microsoft Edge** icon.
2. In the open browser window, navigate to the **Azure Portal** (**https://portal.azure.com**).
3. If prompted, authenticate with the user account account that has the owner role in the Azure subscription you will be using in this lab.

5

Task 2: Deploy an Azure virtual network from the Azure portal by using an Azure Resource Manager template

1. In the upper left corner of the Azure portal, click **Create a resource**.
2. At the top of the **New** blade, in the **Search the Marketplace** text box, type **Template Deployment** and press **Enter**.
3. On the **Everything** blade, in the search results, click **Template deployment**.
4. On the **Template deployment** blade, click the **Create** button.
5. On the **Custom deployment** blade, click the **Build your own template in the editor** link.
6. On the **Edit template** blade, click **Load file**.
7. In the **Choose File to Upload** dialog box, navigate to the **\allfiles\AZ-301T03\Module_01\Labfiles\Starter** folder, select the **vnet-simple-template.json** file, and click **Open**. This will load the following content into the template editor pane:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/
deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "vnetNamePrefix": {
            "type": "string",
            "defaultValue": "vnet-",
```

3 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod01_Getting%20Started%20with%20Azure%20Resource%20Manager%20Templates.md#exercise-1-deploy-core-azure-resources-by-using-an-azure-resource-manager-template-from-the-azure-portal

4 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod01_Getting%20Started%20with%20Azure%20Resource%20Manager%20Templates.md#task-1-open-the-azure-portal

5 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod01_Getting%20Started%20with%20Azure%20Resource%20Manager%20Templates.md#task-2-deploy-an-azure-virtual-network-from-the-azure-portal-by-using-an-azure-resource-manager-template

```
            "metadata": {
                "description": "Name prefix of the vnet"
             }
        },
        "vnetIPPrefix": {
            "type": "string",
            "defaultValue": "10.2.0.0/16",
            "metadata": {
                "description": "IP address prefix of the vnet"
            }
        },
        "subnetNamePrefix": {
            "type": "string",
            "defaultValue": "subnet-",
            "metadata": {
                "description": "Name prefix of the subnets"
            }
        },
        "subnetIPPrefix": {
            "type": "string",
            "defaultValue": "10.2.0.0/24",
            "metadata": {
                "description": "IP address prefix of the first subnet"
            }
        }
    },
    "variables": {
        "vnetName": "[concat(parameters('vnetNamePrefix'), resourceGroup().
name)]",
        "subnetNameSuffix": "0"
    },
    "resources": [
    {
        "apiVersion": "2018-02-01",
        "name": "[variables('vnetName')]",
        "type": "Microsoft.Network/virtualNetworks",
        "location": "[resourceGroup().location]",
        "scale": null,
        "properties": {
            "addressSpace": {
                "addressPrefixes": [
                    "[parameters('vnetIPPrefix')]"
                ]
            },
            "subnets": [
                {
                    "name": "[concat(parameters('subnetNamePrefix'), varia-
bles('subnetNameSuffix'))]",
                    "properties": {
                    "addressPrefix": "[parameters('subnetIPPrefix')]"
                    }
```

```
                    }
                ],
                "virtualNetworkPeerings": [],
                "enableDdosProtection": false,
                "enableVmProtection": false
            },
            "dependsOn": []
            }
        ]
}
```

8. Click the **Save** button to persist the template.
9. Back on the **Custom deployment** blade, perform the following tasks:
   - Leave the **Subscription** drop-down list entry set to its default value.
   - In the **Resource group** section, select the **Create new** option and, in the text box, type **AADesign-Lab0201-RG**.
   - In the **Location** drop-down list, select the Azure region to which you want to deploy resources in this lab.
   - Leave the **vnetNamePrefix** text box set to its default value.
   - Leave the **vnetIPPrefix** text box set to its default value.
   - Leave the **subnetNamePrefix** text box set to its default value.
   - Leave the **subnetIPPrefix** text box set to its default value.
   - In the **Terms and Conditions** section, select the **I agree to the terms and conditions stated above** checkbox.
   - Click the **Purchase** button.
10. Wait for the deployment to complete before you proceed to the next task.

6

Task 3: View deployment metadata

1. In the hub menu of the Azure portal, click **Resource groups**.
2. On the **Resource groups** blade, click the entry representing the resource group to which you deployed the template in the previous task.
3. With the **Overview** selection active, on the resource group blade, click the **Deployments** link.
4. On the resulting blade, click the latest deployment to view its metadata in a new blade.
5. Within the deployment blade, observe the information displayed in the **Operation details** section.

Review: In this exercise, you deployed an Azure virtual network by using an Azure Resource Manager template from the Azure portal

6 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod01_Getting%20Started%20with%20Azure%20Resource%20Manager%20Templates.md#task-3-view-deployment-metadata

7

Exercise 2: Deploy core Azure resources by using Azure Building Blocks from the Azure Cloud Shell

8

Task 1: Open Cloud Shell

1. At the top of the portal, click the **Cloud Shell** icon to open a new shell instance.
2. **Note**: The **Cloud Shell** icon is a symbol that is constructed of the combination of the *greater than* and *underscore* characters.
3. If this is your first time opening the **Cloud Shell** using your subscription, you will see a wizard to configure **Cloud Shell** for first-time usage. When prompted, in the **Welcome to Azure Cloud Shell** pane, click **Bash (Linux)**.
4. **Note**: If you do not see the configuration options for **Cloud Shell**, this is most likely because you are using an existing subscription with this course's labs. If so, proceed directly to the next task.
5. In the **You have no storage mounted** pane, click **Show advanced settings**, perform the following tasks:
   - Leave the **Subscription** drop-down list entry set to its default value.
   - In the **Cloud Shell region** drop-down list, select the Azure region matching or near the location where you deployed resources in this lab
   - Resource group: ensure that the **Create new** option is selected and, in the text box, type **AADesignLab0202-RG**.
   - In the **Storage account** section, ensure that the **Create new** option is selected and then, in the text box below, type a unique name consisting of a combination of between 3 and 24 characters and digits.
   - In the **File share** section, ensure that the **Create new** option is selected and then, in the text box below, type **cloudshell**.
   - Click the **Create storage** button.
6. Wait for the **Cloud Shell** to finish its first-time setup procedures before you continue to the next task.

7 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod01_Getting%20Started%20with%20Azure%20Resource%20Manager%20Templates.md#exercise-2-deploy-core-azure-resources-by-using-azure-building-blocks-from-the-azure-cloud-shell

8 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod01_Getting%20Started%20with%20Azure%20Resource%20Manager%20Templates.md#task-1-open-cloud-shell

9

Task 2: Install the Azure Building Blocks npm package in Azure Cloud Shell

1. At the **Cloud Shell** command prompt at the bottom of the portal, type in the following command and press **Enter** to create a local directory to install the Azure Building Blocks npm package:

   ```
   mkdir ~/.npm-global
   ```

2. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to update the npm configuration to include the new local directory:

   ```
   npm config set prefix '~/.npm-global'
   ```

3. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to open the ~./bashrc configuration file for editing:

   ```
   vi ~/.bashrc
   ```

4. At the **Cloud Shell** command prompt, in the vi editor interface, scroll down to the bottom of the file (or type **G**), scroll to the right to the right-most character on the last line (or type **$**), type **a** to enter the **INSERT** mode, press **Enter** to start a new line, and then type the following to add the newly created directory to the system path:

   ```
   export PATH="$HOME/.npm-global/bin:$PATH"
   ```

5. At the **Cloud Shell** command prompt, in the vi editor interface, to save your changes and close the file, press **Esc**, press **:**, type **wq!** and press **Enter**.
6. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to install the Azure Building Blocks npm package:

   ```
   npm install -g @mspnp/azure-building-blocks
   ```

7. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to exit the shell:

   ```
   exit
   ```

8. In the **Cloud Shell timed out** pane, click **Reconnect**.
9. **Note**: You need to restart Cloud Shell for the installation of the Buliding Blocks npm package to take effect.

9 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod01_Getting%20Started%20with%20Azure%20Resource%20Manager%20Templates.md#task-2-install-the-azure-building-blocks-npm-package-in-azure-cloud-shell

10

Task 3: Deploy an Azure virtual network from Cloud Shell by using Azure Building Blocks

1. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to download the GitHub repository containing the Azure Building Blocks templates:

```
git clone https://github.com/mspnp/template-building-blocks.git
```

2. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to view the content of the Azure Building Block parameter file you will use for this deployment:

```
cat ./template-building-blocks/scenarios/vnet/vnet-simple.json
```

3. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a variable which value designates the name of your Azure subscription:

```
SUBSCRIPTION_ID=$(az account list --query "[0].id" | tr -d '"')
```

4. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a variable which value designates the name of the resource group you created earlier in this exercise:

```
RESOURCE_GROUP='AADesignLab0202-RG'
```

5. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a variable which value designates the Azure region you will use for the deployment:

```
LOCATION=$(az group list --query "[?name == 'AADesignLab0201-RG'].location"
--output tsv)
```

6. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to deploy a virtual network by using the Azure Building Blocks:

```
azbb -g $RESOURCE_GROUP -s $SUBSCRIPTION_ID -l $LOCATION -p ./tem-
plate-building-blocks/scenarios/vnet/vnet-simple.json --deploy
```

7. Wait for the deployment to complete before you proceed to the next task.

11

Task 4: View deployment metadata

1. On the left side of the portal, click the **Resource groups** link.

10 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod01_Getting%20Started%20with%20Azure%20Resource%20Manager%20Templates.md#task-3-deploy-an-azure-virtual-network-from-cloud-shell-by-using-azure-building-blocks

11 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod01_Getting%20Started%20with%20Azure%20Resource%20Manager%20Templates.md#task-4-view-deployment-metadata

2. On the **Resource groups** blade, click the entry representing the resource group you created earlier in this exercise.
3. With the **Overview** selection active, on the resource group blade, click the **Deployments** link.
4. On the resulting blade, click the latest deployment to view its metadata in a new blade.
5. Within the deployment blade, observe the information displayed in the **Operation details** section.
6. Close the **Cloud Shell** pane.

Review: In this exercise, you deployed an Azure virtual network by using an Azure Resource Manager template from the Azure portal

12

Exercise 3: Remove lab resources

13

Task 1: Open Cloud Shell

1. At the top of the portal, click the **Cloud Shell** icon to open the Cloud Shell pane.
2. At the **Cloud Shell** command prompt at the bottom of the portal, type in the following command and press **Enter** to list all resource groups you created in this lab:

```
az group list --query "[?starts_with(name,'AADesignLab02')]".name --output tsv
```

3. Verify that the output contains only the resource groups you created in this lab. These groups will be deleted in the next task.

14

Task 2: Delete resource groups

1. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to delete the resource groups you created in this lab

```
az group list --query "[?starts_with(name,'AADesignLab02')]".name --output tsv | xargs -L1 bash -c 'az group delete --name $0 --no-wait --yes'
```

2. Close the **Cloud Shell** prompt at the bottom of the portal.

Review: In this exercise, you removed the resources used in this lab.

12 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod01_Getting%20Started%20with%20Azure%20Resource%20Manager%20Templates.md#exercise-3-remove-lab-resources

13 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod01_Getting%20Started%20with%20Azure%20Resource%20Manager%20Templates.md#task-1-open-cloud-shell-1

14 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod01_Getting%20Started%20with%20Azure%20Resource%20Manager%20Templates.md#task-2-delete-resource-groups

Review Questions

Module 1 Review Questions

Manage resources

You are designing a solution to migrate your on-premises architecture to Microsoft Azure.
You need to ensure resources are deployed in a componentized and reusable manner.

What Azure service is essential to the deployment of module resources? What is used to manage resources with a shared lifecycle?

Suggested Answer ↓

Azure Resource Manager is designed to represent each service in Azure as a resource provider and each service instance as a modular resource. Resource Groups provide a common lifecycle for child resources. You can add multiple resources to a single resource group. You can manage all resources in a resource group as a single unit.

Manage resource access
You are designing a solution to migrate your on-premises architecture to Microsoft Azure.
You need to manage access to Azure resources by team members in your company.
How is access to resources managed in Azure? To which security principals can access be assigned?

Suggested Answer ↓

Azure AD identities grant predetermined levels of access to an Azure subscription, resource group, or individual resource. Role-Based access control is used to assign roles to existing Azure AD identities. You can assign roles to users, groups, and service principals.

Manage security objects
You are designing a solution for Microsoft Azure.
What should you use to manage certificates and keys? What should you do to secure resource deployments?

Suggested Answer ↓

Azure Key Vault allows the storage of cryptographic keys, certificates, and secrets that can be used by applications, services, and users. You can secure resource deployments by retrieving passwords from Key Vault when you deploy an ARM Template.

Module 2 Module Creating Managed Server Applications in Azure

Infrastructure-Backed Platform-as-a-Service (PaaS)

Infrastructure-Backed PaaS

This module describes services that use infrastructure but manage the infrastructure on behalf of the user instead of obfuscating the infrastructure resources. The module focuses on infrastructure-backed PaaS options such as Azure Service Fabric, Container Service, and App Service Environments. The module will explore how to deploy custom workloads to these services such as an HPC batch processing task.

After completing this module, students will be able to:

- Describe the differences between App Service Environments, Service Fabric and Container Service.
- Use Azure Batch to manage an HPC workload.
- Migrate to an Infrastructure-backed PaaS service from another IaaS service or a legacy Cloud Service.

Infrastructure-Backed Platform-as-a-Service (PaaS)

The need to deploy highly scalable, isolated applications with secure network access and high memory utilization is one that is growing with increased cloud adoption. This lesson will describe the options within Microsoft Azure to deploy App Service Environments, Azure Service Fabric and Azure Container Service applications. The growth of containerized applications and microservice based applications requires the use of these services to deploy open source solutions to deploy, manage and orchestrate at low cost and high flexibility.

After completing this section, you will be able to:

- Describe App Service Environments and distinguish between v1 and v2.
- Describe Azure Service Fabric and distinguish between Microservices and Containers.
- Describe Azure Container Services and the variants, Docker, Mesosphere and Kubernetes.
- Deploy an App Service Environment with a web app.

Infrastructure-Backed PaaS

Every Microsoft Azure PaaS service is already hosted on Azure IaaS. Azure provides several options for architecting and hosting your own PaaS applications at scale in the same secure and isolated infrastructure within the Azure Datacenters. To be able to provide highly scalable applications requiring isolation with secure network access and high memory utilization, there are three services available. App Service Environments provides a dedication scalable home for Azure Web Apps. Azure Service Fabric provides a cluster of VMs to host containers and microservices for those applications. Finally, there is the Azure Container Service this provides open source tools to orchestrate containers based on Docker Swarm, Mesosphere and Kubernetes clusters. The benefit of all these technologies is that Azure lets you concentrate on the application architecture, and code whilst it looks after the infrastructure and the orchestration of the containers. In this lesson, we will detail these services and their intended uses.

App Service Environments

App Services are useful because they separate many of the hosting and management concerns for your web application and allow you to focus on your application's functionality and configuration. There are some scenarios, however, where you require a bit more control.

For example, your organization may require you to make sure that all of the virtual machines hosting your web applications do not allow any outbound requests. This is a typical scenario when implementing a web solution that must be PCI compliant. With App Services, you can't access or modify the configuration of the virtual machines hosting your applications. You do not have any mechanism to implement this requirement.

To implement scenarios where you require more control, you can use the App Service Environment (ASE) service in Azure. ASE allows you to configure network access and isolation for your applications. ASE also allows you to scale using pools of instances far beyond the limits of a regular App Service plan instance. Finally, ASE instances are dedicated to your application alone. You retain much of the convenience of using App Services such as automatic scaling, instance management, and load balancing when using ASE but you gain more control.
Networking in ASE

With ASE, you can configure network access using the same concepts and paradigms that you use in Virtual Machines. Environments are created within a subnet in an Azure Virtual Network. You can use Network Security Groups to restrict network communications to the subnet where the Environment resides. You can also use a protected connection to connect your Virtual Network to corporate resources so that your ASE instance can securely use the resources.

ASE is available in v1 and v2. The ASE v1 is useful since it can take advantage of both classic and Resource Manager Virtual Networks whilst v2 can only use Resource Manager resources. ASE v2 also automates most of the scaling, creation, and maintenance which v1 requires to be carried out manually.

Azure Service Fabric

Azure Service Fabric provides a distributed systems solution making it simple to package, deploy, and manage scalable and reliable containers and microservices. Developers can use Service Fabric to assist with the developing and managing cloud native applications.

The use of Service Fabric can avoid complex infrastructure problems and leave operations and development staff to concentrate on implementing mission-critical workloads that are easy to manage, scale and result in a reliable solution. Service Fabric is intended for container-based applications running on enterprise-class, cloud-scale environments.

Microservice based applications

Service Fabric creates a cluster of VMs that allow you to run high density, massively scalable applications composed of microservices. These applications can be stateful or stateless microservices running in containers which Azure Service Fabric will assist with provisioning, deploying, monitoring and managing in fast and efficient, automated fashion.

Service Fabric powers many Microsoft services today, including Azure SQL Database, Azure Cosmos DB, Cortana, Microsoft Power BI, Microsoft Intune, Azure Event Hubs, Azure IoT Hub, Dynamics 365, Skype for Business, and many core Azure services. Azure Service Fabric is ideal for hosting cloud-native applications at high scale often growing to hundreds of thousands of machines.

A microservice could be a user profile, a customer's shopping cart, queues, caches, or any other component of an application. Service Fabric is a platform built to handle microservices by placing each one in a uniquely named container.

Service Fabric provides comprehensive runtime and lifecycle management capabilities to applications that are composed of these microservices. It hosts microservices inside containers that are deployed and activated across the Service Fabric cluster. The use of microservices adds the ability to increase the density of the containers used by a service.

As an example, a single cluster for Azure SQL Database contains hundreds of machines running tens of thousands of containers hosting a total of hundreds of thousands of databases. Each Azure SQL database is a Service Fabric state-full microservice.

Application lifecycle management

Azure Service Fabric supports the application lifecycle and CI/CD of cloud applications including containers.

Service Fabric empowers operators to use simple workflows to provision, deploy, patch, and monitor applications. This dramatically reduces the workload for the operators. Service Fabric is integrated with CI/CD tools such as Visual Studio Team Services, Jenkins, and Octopus Deploy.

Azure Container Service

Azure Container Service (ACS) provides several alternative ways to simplify the creation, configuration, and management of a cluster of virtual machines that are configured to run your applications in a container. Using open-source orchestration and scheduling tools provides a large community of experience and expertise to assist in deploying and managing your containerized applications.

ACS provides three distinct container orchestration technologies:

- **Docker Swarm**
- **Mesosphere DC/OS**
- **Kubernetes**

ACS uses Docker images to provide container portability. The use of the orchestration technologies enables the ability to scale applications to tens of thousands of containers. ACS makes no charges for the service or orchestration since all the tools provided are open source software. Charges are made for resource utilization only. ACS is available for Standard A, D, DS, G, GS, F, and FS series Linux virtual machines. You are only charged for these VMs, Storage and networking resources actually used.

The latest service available is Kubernetes clusters. This is the AKS service. AKS passes the cluster management functions to Azure and removes it from the operator or administrator. Azure manages health, maintenance, and monitoring. AKS also provides easy scaling, automatic version upgrades, and self-healing master nodes.

Since Azure provides all the usual management functions, there is no SSH access to the AKS cluster.

Access is provided to the Kubernetes API endpoints. This allows any software capable of communicating with Kubernetes endpoints to be used. Examples could include kubectl, helm, or draft. Whilst Azure handles the infrastructure management, Kubernetes itself handles your containerized applications.

The list of Kubernetes management features includes:

- **Automatic binpacking**
- **Self-healing**
- **Horizontal scaling**
- **Service discovery and load balancing**
- **Automated rollouts and rollbacks**
- **Secret and configuration management**
- **Storage orchestration**
- **Batch execution**

High-Performance Compute (HPC)

High Performance Computing

High Performance Computing (HPC)

Azure provides solutions to allow for massively scalable parallel processing of memory-intensive workloads, such as 3D rendering and tasks that process, transform, and analyze large volumes of data. Azure Batch provides this service for short-term massively scalable infrastructure. For other cloud-based and even hybrid solutions, Azure provides access to both Windows and Linux HPC clusters. This lesson will describe these solutions and their uses.

After completing this lesson, you will be able to:

- Describe Azure HPC Pack features.
- Describe the Azure Batch service.
- Understand the use of the HPC pack in Cloud only and hybrid scenarios.
- Deploy an Azure HPC pack for Windows.

High Performance Computing (HPC)

Traditionally, complex processing was something saved for universities and major research firms. A combination of cost, complexity, and accessibility served to keep many from pursuing potential gains for their organization from processing large and complex simulations or models. Cloud platforms have democratized hardware so much that massive computing tasks are within reach of hobbyist developers and small to medium-sized enterprises.

High-Performance Computing (HPC) typically describes the aggregation of complex processes across many different machines thereby maximizing the computing power of all of the machines. Through HPC in the cloud, one could create enough compute instances to create a model or perform a calculation and then destroy the instances immediately afterward. Advancements in the HPC field have led to improvements in the way that machines can share memory or communicate with each other in a low latency manner.

Remote Direct Memory Access

Remote Direct Memory Access, or RDMA, is a technology that provides a low-latency network connection between processing running on two servers, or virtual machines in Azure. This technology is essential for engineering simulations, and other compute applications that are too large to fit in the memory of a single machine. From a developer perspective, RDMA is implemented in a way to make it seem that the machines are "sharing memory." RDMA is efficient because it copies data from the network adapter directly to memory and avoids wasting CPU cycles.

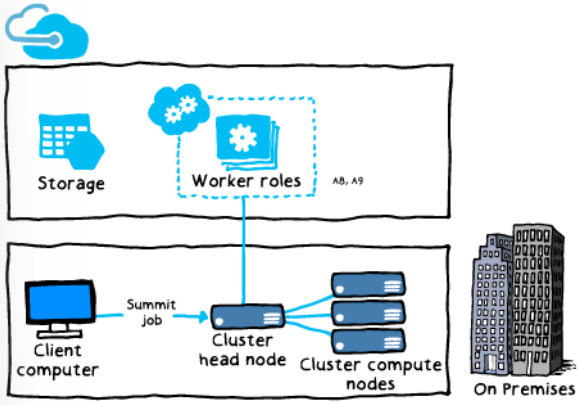
The A8 and A9 VM size in Azure use the InfiniBand network to provide RDMA virtualized through Hyper-V with near "bare metal" performance of less than 3-microsecond latency and greater than 3.5 Gbps bandwidth.

HPC Pack using Azure Virtual Machines

HPC Pack is Microsoft's HPC cluster and job management solution for Windows. The HPC Pack can be installed on a server that functions as the "head node," and the server can be used to manage compute nodes in an HPC cluster.

HPC Pack is not required for you to use the A8, A9, A10, and A11 instances with Windows Server, but it is a recommended tool to create Windows HPC clusters in Azure. In the case of A8 and A9 instances, HPC Pack is the most efficient way to run Windows MPI applications that access the RDMA network in Azure. HPC Pack includes a runtime environment for the Microsoft implementation of the Message Passing Interface for Windows.

HPC Pack can also be used in hybrid scenarios where you want to "burst to Azure" with A8 or A9 instances to obtain more processing power.



RDMA on Linux Virtual Machines in Azure

Starting with HPC Pack 2012 R2 Update 2, HPC Pack supports several Linux distributions to run on compute nodes deployed in Azure VMs, managed by a Windows Server head node. With the latest release of HPC Pack, you can deploy a Linux-based cluster that can run MPI applications that access the RDMA network in Azure. Using HPC Pack, you can create a cluster of virtual machines using either Windows or Linux that use the Intel Message Passing Library (MPI) to spread the workload of simulations and computations among compute nodes of many virtual machines.

Azure Batch

Microsoft Azure Batch is a fully-managed cloud service that provides job scheduling and compute resource management for developers in organizations, independent software vendors, and cloud service providers. Both new and existing high-performance computing (HPC) applications running on workstations and clusters today can be readily enabled to run in Azure at scale, and with no on-premises infrastructure required. Common application workloads include image and video rendering, media transcoding, engineering simulations, Monte Carlo simulations, and software test execution, among others; all highly parallel, computationally intensive workloads that can be broken into individual tasks for execution. With Azure Batch, you can scale from a few VMs, up to tens of thousands of VMs, and run the most massive, most resource-intensive workloads.

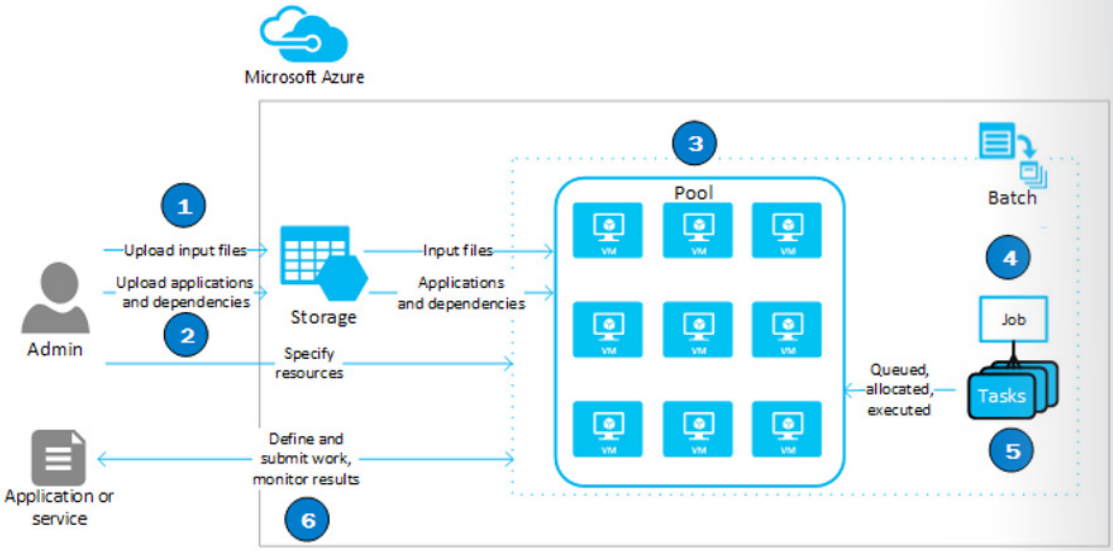
Azure Batch

Azure Batch is a service that manages Virtual Machines for large-scale parallel and high-performance computing (HPC) applications. Batch is a Platform as a Service (PaaS) offering that manages the VMs necessary for your compute jobs for you instead of forcing you to manage an HPC cluster or job schedule. Batch provides auto-scaling functionality and job scheduling functionality in addition to managing compute nodes.

Batch computing is a familiar pattern for organizations that process, transform, and analyze large amounts of data, either on a schedule or on-demand. It includes end-of-cycle processing such as a bank's daily risk reporting or a payroll that must be done on schedule. It also includes large-scale business, science, and engineering applications that typically need the tools and resources of a compute cluster or grid. Applications include traditional HPC applications such as fluid dynamics simulations as well as specialized workloads in fields ranging from digital content creation to financial services to life sciences research. Batch works well with intrinsically parallel (sometimes called "embarrassingly parallel") applications or workloads, which lend themselves to running as parallel tasks on multiple computers.

Scaling out Parallel Workloads

The Batch API can handle scale out of an intrinsically parallel workload such as image rendering on a pool of up to thousands of compute cores. Instead of having to set up a compute cluster or write code to queue and schedule your jobs and move the necessary input and output data, you automate the scheduling of large compute jobs and scale a pool of compute VMs up and down to run them. You can write client apps or front-ends to run jobs and tasks on demand, on a schedule, or as part of a larger workflow managed by tools such as Azure Data Factory.



You can also use the Batch Apps API to wrap an existing application, so it runs as a service on a pool of compute nodes that Batch manages in the background. The application might be one that runs today on client workstations or a compute cluster. You can develop the service to let users offload peak work to the cloud, or run their work entirely in the cloud. The Batch Apps framework handles the movement of input and output files, the splitting of jobs into tasks, job and task processing, and data persistence.

Stateless Component Workloads

In addition to the formal HPC services above, Azure provides non-managed solutions and related services to provide a degree of HPC
using IaaS.

These include:

• Virtual Machines

- VM scale Sets
- Azure Container Services
- HDInsight
- Machine Learning

And plenty more. The inbuilt massively scalable Azure fabric provides a perfect platform to deploy parallel compute tasks on several services. The use of Azure Resource manager to automate deployment and auto scale based on various criteria enables the developer and architect to be creative with their application solutions.

As an example, deploying VMSS to manage an HPC workload by varying the number of VMs based on the queue length would combine auto-scale based on a custom metric with HPC Pack and templated infrastructure deployment. You do not have to rely on the pre defined HPC services.

Migration

On-Premises Lift and Shift

Migrating from on-premises workloads into Azure can be achieved in several ways that depend upon the individual workloads and the level of rewriting that the user is prepared or able to achieve. This lesson covers the Azure Migrate preview service and strategies for migrating from on-premises to Azure IaaS, and PaaS solutions. The ability to convert cloud service roles into Azure Service Fabric microservices is also discussed.

After completing this lesson, you will be able to:

- Describe Azure Migrate.
- Understand the process to migrate from IaaS to PaaS.
- Describe the best way to migrate apps to Azure.
- Use Azure Migrate to plan a single VM migration.

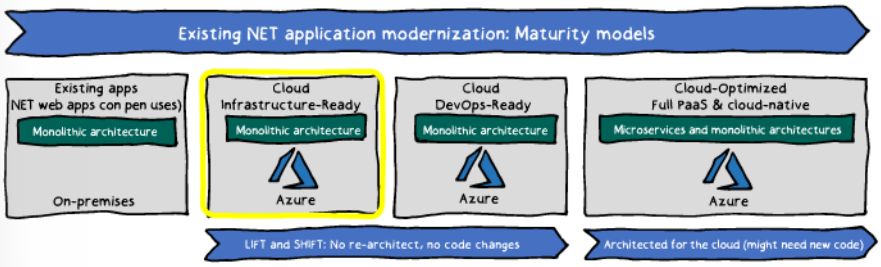
Migration

The challenge of migrating workloads and applications to a cloud-based or hybrid platform or even from converting from cloud-based IaaS to a PaaS solution is one that causes much concern when adopting a public cloud strategy. There are several options for beginning the "journey to the cloud." For moving workloads, the "lift and shift" approach provides all the benefits of pay as you go computing without the potential headaches of rewriting application code or re-architecting the application to fit a specific cloud application pattern.

On-Premises Lift and Shift

A new service from Microsoft, the Azure Migrate service provides a discovery and assessment tool. Azure Migrate assesses suitability for migration and ensures that sizing is correct for the performance of the VM. An estimate of the cost of running your VM in Azure is also available; this service is designed for those considering a lift and shift migration or beginning to plan their migration to the cloud. Migrate provides the ability to visualize dependencies of a specific VM or for all VMs in a group. The service currently only provides assessment for VMWare VMs but soon will also include Hyper-V VMs.

The lift and shift approach is an ideal method to begin the migration of workloads to modernization. Commencing with lift and shift, moving through the phases shown below to cloud-optimized taking advantage of the PaaS services and cloud native applications. The latter often requires the applications to be rewritten.



Migration from Classic IaaS

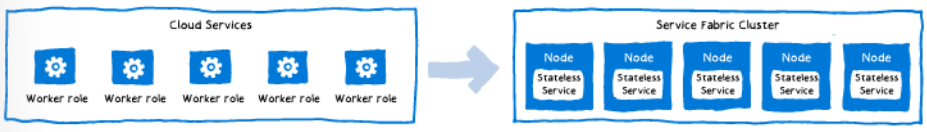
If there are resources in an Azure subscription based on the classic or Azure Service Manager model, it is now possible to migrate them to an Azure Resource Manager (ARM) deployment. The following resources can be migrated to ARM from ASM:

- Virtual Machines
- Availability Sets
- Cloud Services
- Storage Accounts
- Virtual Networks
- VPN Gateways
- Express Route gateways
- Network Security Groups
- Route Tables
- Reserved IPs

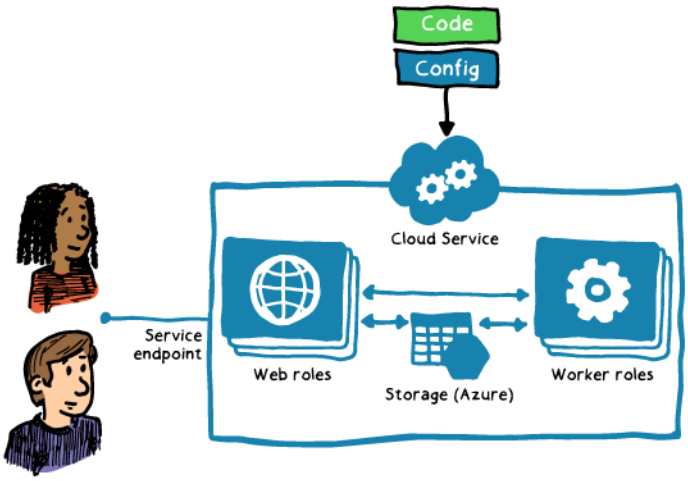
Migration from Cloud Services

When migrating cloud services to a PaaS solution, it is necessary to consider the difference between VMs, workloads, and applications in each model.

A cloud service deploys applications as VMs; code is connected to a VM instance which might be a Web role or a worker role. To scale the application, more VMs are deployed.



The deployment package contains the web role and worker role definition and specifies the instance count for each role; an instance is a VM hosting that role.

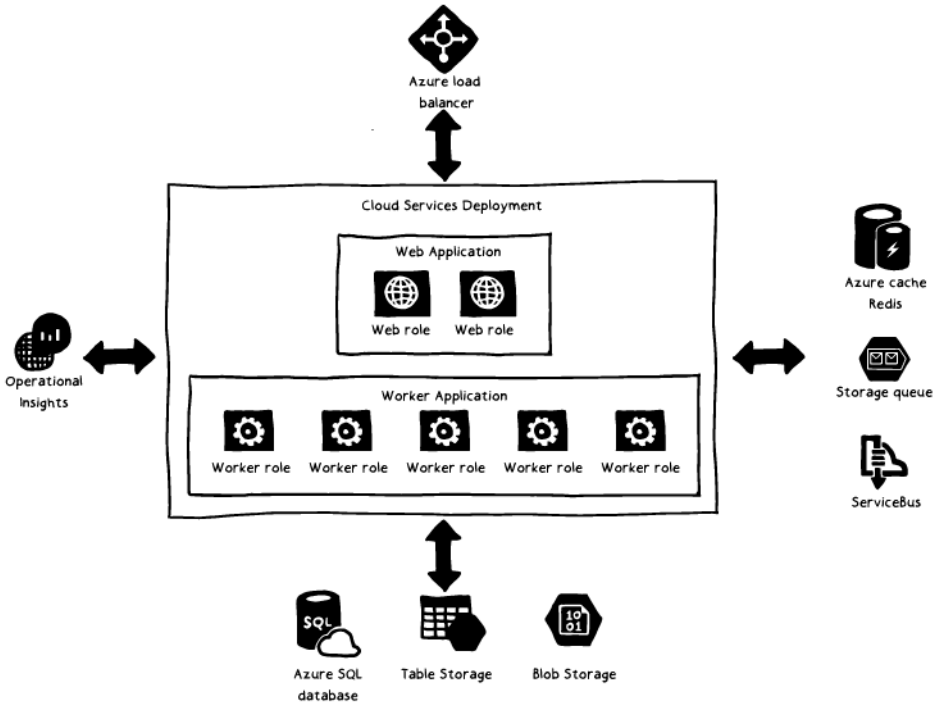


Deployment Package

Migrating a cloud service to Service Fabric switches to deploying applications to VMs that are already running Service Fabric either on Windows or Linux. The applications or services that are deployed are entirely unrelated to the VM infrastructure. The service fabric application platform hides the VM layer from the application. This also allows multiple applications to be deployed to a service fabric cluster.

The architecture of applications running on the two services is fundamentally different.

A cloud service application will typically have external dependencies which manage the data and state of an application and the method of communicating between web and worker roles.



CLOUD SERVICE SOLUTION

A Service fabric application can also rely on the same external service dependencies. The quickest and easiest way to migrate a Cloud Service application to service fabric is to merely convert the Web roles and worker roles to stateless services whilst keeping the architecture the same.

If the aim is to remove the external dependencies and take full advantage of the ability to unify deployment, management and upgrade models, then state-full services would be required which will mean full code and application rewrites.

Migration both to the cloud in general as a lift and shift and to modern cloud-native applications is a complicated and time-consuming project.

Online Lab - Deploying Managed Containerized Workloads to Azure

Lab Steps

Deploying Managed Containerized Workloads to Azure

NOTE: For the most recent version of this online lab, see: **https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesig**n

Before we start

1. Ensure that you are logged in to your Windows 10 lab virtual machine using the following credentials:
   - Username: **Admin**
   - Password: **Pa55w.rd**
2. Review Taskbar located at the bottom of your Windows 10 desktop. The Taskbar contains the icons for the common applications you will use in the labs:
   - Microsoft Edge
   - File Explorer
   - **Visual Studio Code**[1]
   - **Microsoft Azure Storage Explorer**[2]
   - Bash on Ubuntu on Windows
   - Windows PowerShell
3. **Note**: You can also find shortcuts to these applications in the **Start Menu**.

3

Exercise 1: Create Azure Kubernetes Service (AKS) cluster

4

Task 1: Open the Azure Portal

1. On the Taskbar, click the **Microsoft Edge** icon.
2. In the open browser window, navigate to the **Azure Portal** (**https://portal.azure.com**).

1 https://code.visualstudio.com/
2 https://azure.microsoft.com/features/storage-explorer/
3 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod02_Deploying%20Managed%20Containerized%20Workloads%20to%20Azure.md#exercise-1-create-azure-kubernetes-service-aks-cluster
4 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod02_Deploying%20Managed%20Containerized%20Workloads%20to%20Azure.md#task-1-open-the-azure-portal

3. If prompted, authenticate with the user account account that has the owner role in the Azure subscription you will be using in this lab.

5

Task 2: Open Cloud Shell

1. At the top of the portal, click the **Cloud Shell** icon to open a new shell instance.
2. **Note**: The **Cloud Shell** icon is a symbol that is constructed of the combination of the *greater than* and *underscore* characters.
3. If this is your first time opening the **Cloud Shell** using your subscription, you will see a wizard to configure **Cloud Shell** for first-time usage. When prompted, in the **Welcome to Azure Cloud Shell** pane, click **Bash (Linux)**.
4. **Note**: If you do not see the configuration options for **Cloud Shell**, this is most likely because you are using an existing subscription with this course's labs. If so, proceed directly to the next task.
5. In the **You have no storage mounted** pane, click **Show advanced settings**, perform the following tasks:
   - Leave the **Subscription** drop-down list entry set to its default value.
   - In the **Cloud Shell region** drop-down list, select the Azure region matching or near the location where you intend to deploy resources in this lab.
   - In the **Resource group** section, ensure that the **Create new** option is selected and then, in the text box, type **AADesignLab0401-RG**.
   - In the **Storage account** section, ensure that the **Create new** option is selected and then, in the text box below, type a unique name consisting of a combination of between 3 and 24 characters and digits.
   - In the **File share** section, ensure that the **Create new** option is selected and then, in the text box below, type **cloudshell**.
   - Click the **Create storage** button.
6. Wait for the **Cloud Shell** to finish its first-time setup procedures before you proceed to the next task.

6

Task 3: Create an AKS cluster by using Cloud Shell

1. At the **Cloud Shell** command prompt at the bottom of the portal, type in the following command and press **Enter** to create a variable which value designates the name of the resource group you will use in this task:

```
RESOURCE_GROUP='AADesignLab0402-RG'
```

5 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod02_Deploying%20Managed%20Containerized%20Workloads%20to%20Azure.md#task-2-open-cloud-shell

6 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod02_Deploying%20Managed%20Containerized%20Workloads%20to%20Azure.md#task-3-create-an-aks-cluster-by-using-cloud-shell

2. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a variable which value designates the Azure region you will use for the deployment (replace the placeholder `<Azure region>` with the name of the Azure region to which you intend to deploy resources in this lab):

```
LOCATION='<Azure region>'
```

3. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a new resource group:

```
az group create --name $RESOURCE_GROUP --location $LOCATION
```

4. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a new AKS cluster:

```
az aks create --resource-group $RESOURCE_GROUP --name aad0402-akscluster
--node-count 1 --node-vm-size Standard_DS1_v2 --generate-ssh-keys
```

5. **Note**: If you receive an error message regarding availability of the VM size which value is represented by the `--node-vm-size` parameter, review the message and try other suggested VM sizes.
6. **Note**: Alternatively, you can identify VM sizes available in your subscription in a given region by running the following command and reviewing the values in the **Restriction** column (make sure to replace the `<region>` placeholder with the name of the target region):

```
Get-AzComputeResourceSku | where {$_.Locations -icontains "region"} |
Where-Object {($_.ResourceType -ilike "virtualMachines")}.
```

7. **Note**: The **Restriction** column will contain the value **NotAvailableForSubscription** for VM sizes that are not available in your subscription.
8. **Note**: As of 2/21/2019, VM Size **Standard_DS2_V2** was available in **westeurope**
9. Wait for the deployment to complete before you proceed to the next task.
10. **Note**: This operation can take up to 10 minutes.

7

Task 4: Connect to the AKS cluster.

1. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to retrieve the credentials to access the AKS cluster:

```
az aks get-credentials --resource-group $RESOURCE_GROUP --name aad0402-aks-
cluster
```

2. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to verify connectivity to the AKS cluster:

7 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod02_Deploying%20Managed%20Containerized%20Workloads%20to%20Azure.md#task-4-connect-to-the-aks-cluster

```
kubectl get nodes
```

3. At the **Cloud Shell** command prompt, review the output and verify that the node is reporting the **Ready** status. Rerun the command until the correct status is shown.

Result: After you complete this exercise, you should have successfully deployed a new AKS cluster.

8

Exercise 2: Managing an AKS cluster and its containerized workloads.

9

Task 1: Deploy a containerized application to an AKS cluster

1. In the Microsoft Edge window, in the Azure portal, at the **Cloud Shell** prompt, type the following command and press **Enter** in order to deploy the **nginx** image from the Docker Hub:

```
kubectl run aad0402-akscluster --image=nginx --replicas=1 --port=80
```

1. **Note**: Make sure to use lower case letters when typing the name of the deployment.
2. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to verify that a Kubernetes pod has been created:

```
kubectl get pods
```

3. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to identify the state of the deployment:

```
kubectl get deployment
```

4. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to make the pod available from Internet:

```
kubectl expose deployment aad0402-akscluster --port=80 --type=LoadBalancer
```

5. **Note**: Make sure to use lower case letters when typing the name of the deployment.
6. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to identify whether the public IP address has been provisioned:

```
kubectl get service --watch
```

8 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod02_Deploying%20Managed%20Containerized%20Workloads%20to%20Azure.md#exercise-2-managing-an-aks-cluster-and-its-containerized-workloads

9 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod02_Deploying%20Managed%20Containerized%20Workloads%20to%20Azure.md#task-1-deploy-a-containerized-application-to-an-aks-cluster

7. Wait until the value in the **EXTERNAL-IP** column for the **aad0402-akscluster** entry changes from to a public IP address, then press **Ctrl-C** key combination. Note the public IP address in the **EXTERNAL-IP** column for **aad0402-akscluster**.
8. Start Microsoft Edge and browse to the IP address you obtained in the previous step. Verify that Microsoft Edge displays a web page with the **Welcome to nginx!** message.

10

Task 2: Scaling containerized applications and AKS cluster nodes

1. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to scale the deployment:

```
kubectl scale --replicas=2 deployment/aad0402-akscluster
```

2. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to verify the outcome of scaling the deployment:

```
kubectl get pods
```

3. **Note**: Review the output of the command and verify that the number of pods increased to 2.
4. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to scale out the number of cluster nodes:

```
az aks scale --resource-group $RESOURCE_GROUP --name aad0402-akscluster
--node-count 2
```

5. Wait for the provisioning of the additional node to complete.
6. **Note**: This operation can take up to 10 minutes. If it fails, rerun the `az aks scale` command.
7. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to verify the outcome of scaling the cluster:

```
kubectl get nodes
```

8. **Note**: Review the output of the command and verify that the number of nodes increased to 2.
9. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to scale the deployment:

```
kubectl scale --replicas=10 deployment/aad0402-akscluster
```

10. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to verify the outcome of scaling the deployment:

```
kubectl get pods
```

10 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod02_Deploying%20Managed%20Containerized%20Workloads%20to%20Azure.md#task-2-scaling-containerized-applications-and-aks-cluster-nodes

11. **Note**: Review the output of the command and verify that the number of pods increased to 10.
12. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to review the pods distribution across cluster nodes:

```
kubectl get pod -o=custom-columns=NODE:.spec.nodeName,POD:.metadata.name
```

13. **Note**: Review the output of the command and verify that the pods are distributed across both nodes.
14. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to delete the deployment:

```
kubectl delete deployment aad0402-akscluster
```

11

Exercise 3: Autoscaling pods in an AKS cluster

12

Task 1: Deploy a Kubernetes pod by using a **.yaml** file.

1. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to download a sample containerized application:

```
git clone https://github.com/Azure-Samples/azure-voting-app-redis.git
```

2. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to navigate to the location of the downloaded app:

```
cd azure-voting-app-redis
```

3. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to list the content of the application **.yaml** file:

```
cat azure-vote-all-in-one-redis.yaml
```

4. Review the output of the command and verify that the pod defninition includes requests and limits in the followng format:

```
resources:
  requests:
     cpu: 250m
  limits:
     cpu: 500m
```

11 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod02_Deploying%20Managed%20Containerized%20Workloads%20to%20Azure.md#exercise-3-autoscaling-pods-in-an-aks-cluster

12 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod02_Deploying%20Managed%20Containerized%20Workloads%20to%20Azure.md#task-1-deploy-a-kubernetes-pod-by-using-a-yaml-file

5. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to deploy the application based on the **.yaml** file:

```
kubectl apply -f azure-vote-all-in-one-redis.yaml
```

6. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to to verify that a Kubernetes pod has been created:

```
kubectl get pods
```

7. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to identify whether the public IP address for the containerized application has been provisioned:

```
kubectl get service azure-vote-front --watch
```

8. Wait until the value in the **EXTERNAL-IP** column for the **azure-vote-front** entry changes from to a public IP address, then press **Ctrl-C** key combination. Note the public IP address in the **EXTERNAL-IP** column for **azure-vote-front**.
9. Start Microsoft Edge and browse to the IP address you obtained in the previous step. Verify that Microsoft Edge displays a web page with the **Azure Voting App** message.

13

Task 2: Autoscale Kubernetes pods.

1. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to download a sample containerized application:

```
git clone https://github.com/kubernetes-incubator/metrics-server.git
```

2. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to install **Metrics Server**:

```
kubectl create -f ~/metrics-server/deploy/1.8+/
```

3. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to configure autoscaling for the **azure-vote-front** deployment:

```
kubectl autoscale deployment azure-vote-front --cpu-percent=50 --min=3
--max=10
```

4. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to view the status of autoscaling:

```
kubectl get hpa
```

5. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to view the pods:

13 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod02_Deploying%20Managed%20Containerized%20Workloads%20to%20Azure.md#task-2-autoscale-kubernetes-pods

```
kubectl get pods
```

6. **Note**: Verify that the number of replicas incrased to 3. If that is not the case, wait one minute and rerun the two previous steps.
7. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to delete the deployment:

```
kubectl delete deployment azure-vote-front
```

8. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to delete the deployment:

```
kubectl delete deployment azure-vote-back
```

9. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to verify that the commands you ran in the previous steps completed successfully:

```
kubectl get pods
```

10. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to delete the AKS cluster:

```
az aks delete --resource-group AADesignLab0402-RG --name aad0402-akscluster
--yes --no-wait
```

11. Close the **Cloud Shell** pane.

Review: In this exercise, you implemented autoscaling of pods in an AKS cluster

14

Exercise 4: Implement DevOps with AKS

15

Task 1: Deploy DevOps with AKS

```
> **Note**: This solution is based on the DevOps with Containers solution
described at https://docs.microsoft.com/en-us/azure/architecture/exam-
ple-scenario/apps/devops-with-aks.
```

1. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to generate the SSH key pair that will be used to authenticate when accessing the Linux VMs running the Jenkins instance and Grafana console:

14 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod02_Deploying%20Managed%20Containerized%20Workloads%20to%20Azure.md#exercise-4-implement-devops-with-aks

15 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod02_Deploying%20Managed%20Containerized%20Workloads%20to%20Azure.md#task-1-deploy-devops-with-aks

```
ssh-keygen -t rsa -b 2048
```

- When prompted to enter the file in which to save the key, press **Enter** to accept the default value **(~/.ssh/id_rsa)**.
- When prompted to enter passphrase, press **Enter** twice.

2. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a variable which value designates the public key of the newly generated key pair:

```
PUBLIC_KEY=$(cat ~/.ssh/id_rsa.pub)
```

3. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a variable which value designates the public key of the newly generated key pair and which takes into account any special character the public key might include:

```
PUBLIC_KEY_REGEX="$(echo $PUBLIC_KEY | sed -e 's/\\/\\\\/g; s/\//\\\//g;
s/&/\\\&/g')"
```

4. **Note**: This is necessary because you will use the **sed** utility to insert this string into the Azure Resource Manager template parameters file. Alternatively, you could simply open the file and enter the public key string directly into the file.
5. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a variable which value designates the name of the resource group you will use for the deployment:

```
RESOURCE_GROUP='AADesignLab0403-RG'
```

6. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a variable which value designates the Azure region you will use for the deployment:

```
LOCATION=$(az group list --query "[?name == 'AADesignLab0402-RG'].location"
--output tsv)
```

7. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a new resource group:

```
az group create --name $RESOURCE_GROUP --location $LOCATION
```

8. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create an Azure Active Directory service principal for the authentication of services and resources within the sample solution:

```
SERVICE_PRINCIPAL=$(az ad sp create-for-rbac --name AADesignLab0403-SP)
```

9. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to retrieve the **appId** attribute of the newly created service principal:

```
APP_ID=$(echo $SERVICE_PRINCIPAL | jq .appId | tr -d '"')
```

10. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to retrieve the **password** attribute of the newly created service principal:

```
PASSWORD=$(echo $SERVICE_PRINCIPAL | jq .password | tr -d '"')
```

11. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create the parameters file you will use for deployment of the sample solution and open it in the vi interface:

```
vi ~/parameters.json
```

12. At the **Cloud Shell** command prompt, in the vi editor interface, add the content of the sample parameters file (**\allfiles\AZ-301T03\Module_02\Labfiles\Starter\parameters.json**):

```
{
  "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploy-
mentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "spClientId": {
      "value": "$APP_ID"
    },
    "spClientSecret": {
      "value": "$PASSWORD"
    },
    "linuxAdminUsername": {
      "value": "Student"
    },
    "linuxAdminPassword": {
      "value": "Pa55w.rd1234"
    },
    "linuxSSHPublicKey": {
      "value": "$PUBLIC_KEY_REGEX"
    }
  }
}
```

13. At the **Cloud Shell** command prompt, in the vi editor interface, to save your changes and close the file, press **Esc**, press **:**, type **wq!** and press **Enter**.

14. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to replace the placeholder for the **appId** attibute with the value of the **$APP_ID** variable in the parameters file:

```
sed -i.bak1 's/"$APP_ID"/"'"$APP_ID"'"/' ~/parameters.json
```

15. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to replace the placeholder for the **password** attribute with the value of the **$PASSWORD** variable in the parameters file:

```
sed -i.bak2 's/"$PASSWORD"/"'"$PASSWORD"'"/' ~/parameters.json
```

16. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to replace the placeholder for the **sshPublicKey** parameter with the value of the **$PUBLIC_KEY_REGEX** variable in the parameters file:

```
sed -i.bak3 's/"$PUBLIC_KEY_REGEX"/"'"$PUBLIC_KEY_REGEX"'"/' ~/parameters.
json
```

17. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to verify that the placeholders were successfully replaced in the parameters file:

```
cat ~/parameters.json
```

18. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to deploy the sample solution by using its Azure Resource Manager template residing in a GitHub repository:

```
az group deployment create --resource-group $RESOURCE_GROUP --template-uri
https://raw.githubusercontent.com/mspnp/solution-architectures/master/apps/
devops-with-aks/azuredeploy.json --parameters @parameters.json
```

19. Wait for the deployment to complete before you proceed to the next task.
20. **Note**: The deployment can take up to 15 minutes.

16

Task 2: Review the DevOps with AKS architecture

1. In the hub menu in the Azure portal, click **Resource groups**.
2. On the **Resource groups** blade, click the entry representing the **AADesignLab0403-RG** resource group.
3. On the **AADesignLab0403-RG** resource group blade, review the list of resources and compare them with the information available at **https://docs.microsoft.com/en-us/azure/architecture/example-scenario/apps/devops-with-aks**

Review: In this exercise, you deployed Azure VMs running Windows Server 2016 Datacenter and Linux from Cloud Shell by using Azure Building Blocks.

17

Exercise 5: Remove lab resources

18

Task 1: Open Cloud Shell

1. At the top of the portal, click the **Cloud Shell** icon to open the Cloud Shell pane.

---

16 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod02_Deploying%20Managed%20Containerized%20Workloads%20to%20Azure.md#task-2-review-the-devops-with-aks-architecture

17 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod02_Deploying%20Managed%20Containerized%20Workloads%20to%20Azure.md#exercise-5-remove-lab-resources

18 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod02_Deploying%20Managed%20Containerized%20Workloads%20to%20Azure.md#task-1-open-cloud-shell

2. At the **Cloud Shell** command prompt at the bottom of the portal, type in the following command and press **Enter** to list all resource groups you created in this lab:

```
az group list --query "[?starts_with(name,'AADesignLab04')]".name --output
tsv
```

3. Verify that the output contains only the resource groups you created in this lab. These groups will be deleted in the next task.

19

Task 2: Delete resource groups

1. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to delete the resource groups you created in this lab

```
az group list --query "[?starts_with(name,'AADesignLab04')]".name --output
tsv | xargs -L1 bash -c 'az group delete --name $0 --no-wait --yes'
```

2. Close the **Cloud Shell** prompt at the bottom of the portal.

Review: In this exercise, you removed the resources used in this lab.

19 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod02_Deploying%20Managed%20Containerized%20Workloads%20to%20Azure.md#task-2-delete-resource-groups

Review Questions

Module 2 Review Questions

Infrastructure as a Service (IaaS)

You are designing a solution for your company.

You need to automate management of the infrastructure for the solution.

What Azure architecture type provides automatic management of infrastructure? What Azure services are built on this principle?

Suggested Answer ↓

Azure provides an Infrastructure-Backed Platform-as-a-Service (IaaS) architecture that can be used when automatic management of infrastructure is required. There are several services that Azure offers built on this architecture including Web Apps, Service Fabric Apps, and Container Services.

High performance computing

You are designing a solution for a company. The solution requires high performance and low latency. Which Azure services support High Performance Computing?

Suggested Answer ↓

he Microsoft Azure platform has a large number of features that support HPC, including HPC Pack, Azure Batch, and Stateless Component Workloads.

Migration strategies

You develop a solution to migrate your company architecture to Azure Resource Manager (ARM).

You need to move all on-premises and classic Azure resources.

What types of migration does Azure support? What service does Azure offer for virtual machines (VMs)?

Suggested Answer ↓

Microsoft Azure supports multiple types of migration strategies including on-premises lift and shift, classic (Azure Service Manager) migration to ARM, and cloud to Platform-as-a-Service (PaaS). The Azure Migrate service assesses suitability of workloads and environments for migration, and ensures that VM sizing is optimized.

Module 3 Module Authoring Serverless Applications in Azure

Azure Web Apps

Web Apps

This module describes how solutions can leverage serverless application hosting services in Azure to host web applications, REST APIs, integration workflows and HPC workloads without the requirement to manage specific server resources. The module focuses on App Services-related components such as Web Apps, API Apps, Mobile Apps, Logic Apps, and Functions.

After completing this module, students will be able to:

- Select between hosting application code or containers in an App Service instance.
- Describe the differences between API, Mobile, and Web Apps.
- Integrate an API or Logic App with the API Management service.
- Design an App Service Plan or multi-region deployment for high performance and scale.

Azure Web Apps

This lesson describes the Web Apps service. In many scenarios, it is preferable to use a quick and easy way to deploy web applications to the cloud rather than to reengineer the web applications as cloud projects. Web Apps allow you to quickly create a new Web App and iterate changes to the Web App in an agile manner.

After completing this section you will be able to:

- Describe the Web Apps service.
- List the different tiers for a Web App.

Web Apps

Web Apps is a low friction Platform-as-a-Service (PaaS) offering to host your web applications in the Azure platform. The service is fully managed and you can easily configure advanced features such as AlwaysOn, custom domains, and autoscale by using either portal.

Flexibility

You can use a variety of integrated development environments (IDEs) and frameworks, such as .NET, Java, PHP, Node.js, or Python, to develop your web applications that are eventually deployed to Azure Web Apps. You can use Git and Kudu to deploy Node.js or PHP web applications. You also can deploy web applications that are developed in Microsoft Visual Studio to Web Apps by using the File Transfer Protocol (FTP) or the Web Deploy protocol.

Scalability

Because Web Apps is a fully managed service implementation, you can focus on developing your application and solving business problems instead of the hosting implementation and hardware scaling or specifics. You can easily scale up a stateless web application by configuring autoscale in the portal. Autoscale creates multiple instances of your Web App that are automatically load balanced so that your application can meet potential spikes in demand.

Web App Containers

The Web Apps service is offered in both a Windows and Linux variant. The Linux variant specifically offers the ability to host Docker containers directly using a Web App. The docker containers can be sourced from Docker Hub, Azure Container Registry or GitHub. Containers can be deployed manually, as part of the Web App deployment or deployed in a streamlined continuous integration process using Docker Hub or GitHub.

API Apps

API and Mobile Apps are specialized version of Web Apps designed to server different purposes in an all-up solution on Azure.

API Apps provide specific support for developing, hosting and securing your custom API's in the context of App. API Apps can run either custom code or can run dozens of pre-built software to connect to existing popular Software-as-a-Service solutions. API Apps share a lot of the same features and support as Web Apps.

Mobile Apps

Azure Mobile Apps is a component of Azure App Services offering designed to make it easy to create highly-functional mobile apps using Azure. Mobile Apps brings together a set of Azure services that enable backend capabilities for your apps. Mobile Apps provides the following backend capabilities in Azure to support your apps:

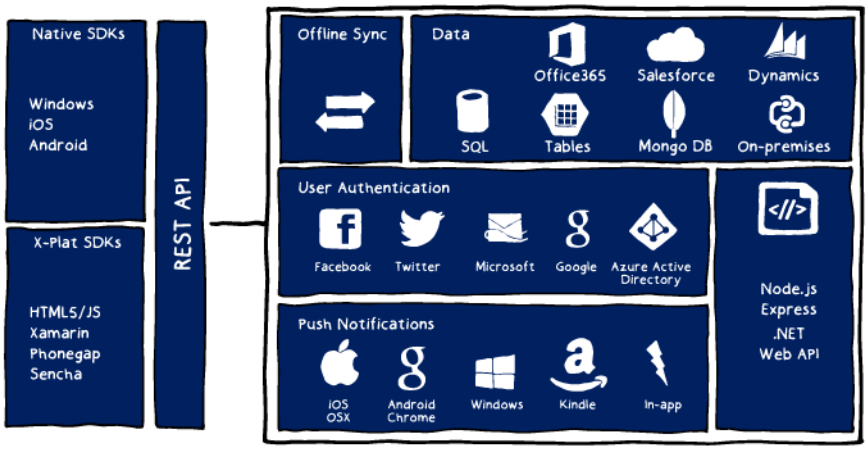
• **Single Sign On:** Select from an ever-growing list of identity providers including Azure Active Directory, Facebook, Google, Twitter, and Microsoft Account, and leverage Mobile Apps to add authentication to your app in minutes.

• **Offline Sync:** Mobile Apps makes it easy for you to build robust and responsive apps that allow employees to work offline when connectivity is not available, and synchronize with your enterprise backend systems when devices comes back online. Offline sync capability is supported on all client platforms and works with any data source including SQL, Table Storage, Mongo, or Document DB, and any SaaS API including Office 365, Salesforce, Dynamics, or on-premises databases.

• **Push Notifications:** Mobile Apps offers a massively scalable mobile push notification engine, Notification Hubs, capable of sending millions of personalized push notifications to dynamic segments of audience using iOS, Android, Windows, or Kindle devices within seconds. You can easily hook Notification Hubs to any existing app backend, whether that backend is hosted on-premises or in the cloud.

• **Auto Scaling:** App Service enables you to quickly scale-up or out to handle any incoming customer load. Manually select the number and size of VMs or set up auto-scaling to scale your mobile app backend based on load or schedule.

Mobile App endpoints are, at their simplest, REST APIs and can be used on a wide variety of platforms and with a wide variety of devices. Client SDKs are available, however, if you like to connect your mobile application to a Mobile App instance for its backend data.



Mobile client SDKs are available for the following platforms:

• **Xamarin Android/IOS**

• **Android Native**

• **IOS Native**

• **Windows Store**

• **Windows Phone**

• **.NET**

• **HTML**

Azure Functions

Serverless Processing

This section briefly introduces Azure Functions and the concept of Serverless processing.

After completing this lesson, you will be able to:

• Explain how serverless compute can potentially save costs.

• Describe the Azure Functions service.

Serverless Processing

Azure Functions are the newest kind of App available in App Services. Function Apps are designed to make it faster to process events
and distribute output to bound services.

Note: Function Apps were implemented using the existing code and base functionality for Azure WebJobs. To this end, you can go to GitHub and view the Azure WebJobs repositories to see the open-source implementation of Function Apps.

Function Apps can be created using one of two models:

• **App Service**: The App Service model allows you to use the already familiar App Service Plan paradigm with Function Apps. In this model, Function Apps exist as simply another kind of app within an App Service Plan.

• **Consumption:** The consumption model allows you to pay for Function Apps based on execution time as opposed to having a dedicated App Service Plan. Function Apps are billed at a rate specific to Gigabyte Seconds after a specific free monthly allocation threshold has been met.
Function Apps share a lot of functionality with other App types such as Web Apps. You can configure App Settings, Connection Strings, AlwaysOn, Continous Deployment and many other settings that are configurable in a Web App instance.

Note: In order to guarantee that your Function App will respond to requests as quickly as possible, you should enable the AlwaysOn feature for your Function App whenever possible.

Event-Based Triggers

Azure Functions must be initiated in some way before they can begin running code or processing data. A Trigger is anything that invokes an Azure Function.

Azure Functions have triggers for many different scenarios including:

• Creation of Blobs

- Changes to data in Cosmos DB
- External File or Table changes
- HTTP requests
- OneDrive or Excel files
- E-mail messages
- Mobile Apps
- SendGrid e-mails
- Twilio Text Messages

Additionally, Azure Functions can also be triggered by existing messaging services including:

- Service Bus
- Storage Queues
- Logic Apps
- Event Grid
- Event Hubs
- Webhooks

Azure Functions is unique in the sense that all of these triggers can be configured with minimal or no code. The developer team can focus on authoring the application.

Integration

API Management

This section introduces API Management and Logic Apps as integration solutions available on the Azure platform.

After completing this section, you will be able to:

• Describe API Management in the context of a REST API B2B solution.

• Use Logic Apps for sophisticated integration among application components.

API Management

Azure API Management helps organizations publish APIs to external, partner and internal developers to unlock the potential of their data and services. Businesses everywhere are looking to extend their operations as a digital platform, creating new channels, finding new customers and driving deeper engagement with existing ones. API Management provides the core competencies to ensure a successful API program through developer engagement, business insights, analytics, security and protection.

To use API Management, administrators create APIs. Each API consists of one or more operations, and each API can be added to one or more products. To use an API, developers subscribe to a product that contains that API, and then they can call the API's operation, subject to any usage policies that may be in effect.

Logic Apps

Logic Apps are workflows, designed using JSON, that can connect various components of your application together using minimal or no-code. Logic Apps can integrate with existing services using built-in connectors. There are connectors available for popular services such as:

• SQL Server

• OneDrive

• Dropbox

• Twilio

• SendGrid

• Cosmos DB

• Event Grid

• Event Hubs

• Storage Blobs, Queues and Tables

• Mobile Apps

Logic Apps can also integrate with custom APIs that are deployed as API Apps in your subscription.

The main components of a Logic App are as follows:

• **Workflow**: The business process described as a series of steps, in an acyclic graph (loops are not supported).

• **Triggers**: The step that starts a new workflow instance.

• **Actions**: A step in a workflow, typically a Connector or a custom API App.

• **Connector**: A special case of API App ready made to integrate a specific service (e.g., Twitter) or data source (e.g., SQL Server).

You can think of Logic Apps as being like toy bricks for integration- you select your bricks and snap together an app.
For example, you could use the Twitter Connector to read tweets of interest and write them to SQL Database using the Microsoft SQL Connector.

Additionally, you could use the Azure Service Bus Connector to queue processing that you will run on HDInsight against some big data, and then send the result via an SMS message using the Twilio connector.

All of this is performed by configuring the connectors, declaratively instead of by writing code.

High Performance Hosting

Best Practices

Azure provides the capability to create scalable and performance aware application using several PaaS Service together. Using a serverless model we can create global scale and high performance web applications.

This section outlines how the App Service and related services can work together to create scalable and performance web applications.

After completing this section you will be able to:

• Best Practices in creating performance aware web applications.

• How to scale your web application.

• Understand how multi-region apps can be built in Azure.

• How to create a business continuity plan.

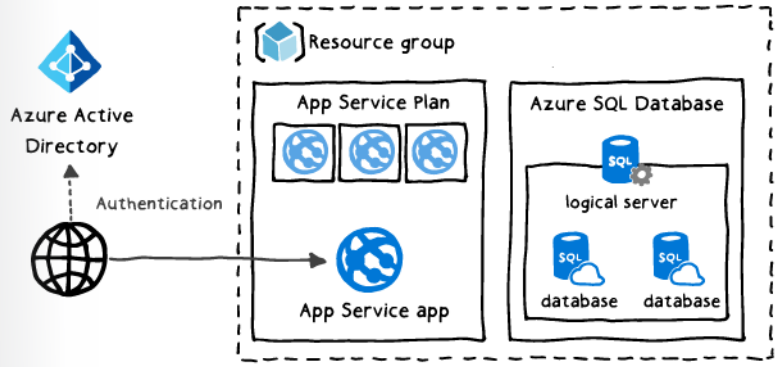
Best Practices

Azure allows you to create state of the art deployments for your web applications. Both Front-end and Back-end. Using the PaaS service App Service, the challenge of creating complex, global high-scalable solutions is simplified. Azure provides a set of services that will simplify the creation of such solutions. These services range from App Service, Traffic Manager, Redis Cache, Queues, Cosmo DB, etc.

Basic Web Application

In this architecture, we start by using the App Service that can have more than one application running in the same compute. We also have
a logic SQL Server to store our data. It's recommended to use the Standard or Premium tiers in order to support autoscale and SSL.

It's also a recommendation to host both the app services as the SQL database in the same Azure region in order to minimize network latency.



Scaling

In Azure App Service, when an administrator needs to increase the performance he can do it by changing the number of instances running in the App Service Plan (scale out) or by changing to a higher pricing tier (scale up). Scaling up provides more CPU, memory, disk space and features, like autoscale, deployment slots, etc. Scaling out increases the number of instances running the application. This scaling can be manual, if you're using the Basic Tier, or automatically, if you're using the Standard or upper service tiers. Standard tier allows to scale up to 10 instances and if you still need more instances you can go to the Isolated tier where you can scale up to 100 instances. Scaling an application does not require you to change your application code.

In this architecture we had separated our two applications tiers (Web app and an REST API) into different App Service Plans. This way we can scale each tier individually. We had created a Web Job in order to process long-running task in the background, this is also accomplished by using a Message Queue to control the work to be done and already processed.

Traffic Manager

Using a multi-region architecture provides a higher availability of your applications. If there is an outage in the primary datacenter, using the Traffic Manager service is possible to redirect the traffic to a secondary data center that is available, this way the application will not suffer an outage, reinforcing the overall application uptime.

In a multi-region model, we have three main components:

- Active and Standby regions
- Traffic Manager
- SQL Database Geo-replication

In a multi-region model, we have higher global availability of our solutions. This is accomplished by having our solution replicated in two regions and having the Traffic Manager routing incoming traffic to the active region, but in case of failure of this region, this service will failover to the standby region. The SQL Database active geo-replication feature also plays an essential role in this architecture. It creates a replica in the standby region and if the primary database fails or needs to be taken offline, we can failover to one of the up to four replicas.

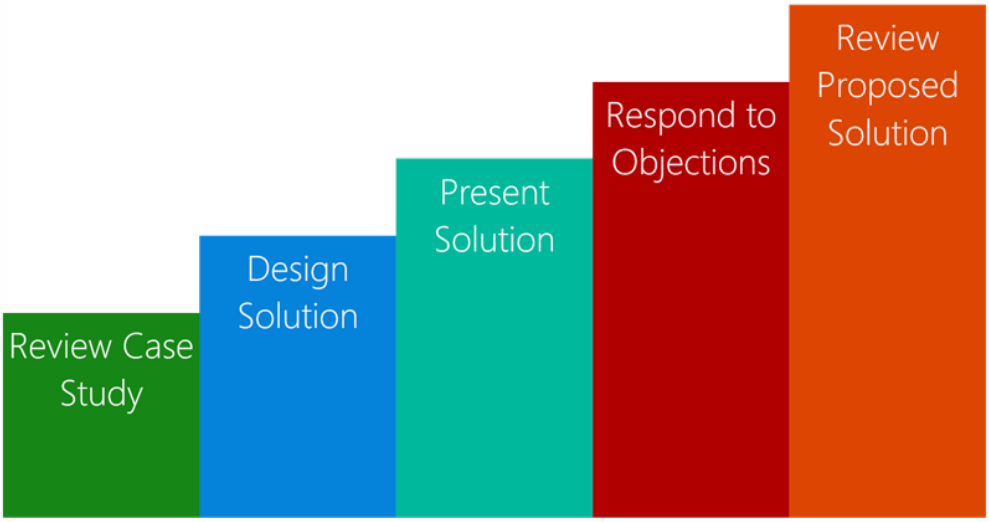
Design - Mobile Apps Case Study

Case Study Overview

In this case study, we will look at a customer problem that requires an architectural recommendation.

After this case study, you should:

- Identify customer problems as they are related to networking.
- Design a solution that will meet the customer's objectives.
- Ensure your designed solution accounts for customer objections.



Who Is the Customer?

With Corporate Headquarters in Phoenix Arizona, the Crazy Taxi Cab Company has quickly risen to be the premier provider of private low-cost transportation in Arizona. Bucking industry norms, Crazy Taxi Cab Co drivers are company employees who work as a team, rather than independent contractors who essentially compete with each other for fares. The founding partners believed this would allow its drivers to focus on providing a great customer experience as opposed to simply "racing to the finish line." Crazy Taxi has developed a reputation for having fast, reliable, and friendly service, due largely in part to their extensive network of drivers, and their well-executed, albeit radio based dispatching approach.
Crazy Taxi is drowning in success. While dispatchers are reaching out to drivers to pick up customers who have called in, new callers often find themselves on hold, waiting for their calls to be answered. The executives at Crazy Taxi realize that they need to modernize their operation or risk losing business in the future. Their Chief Operating Officer Christopher Giovanni states that "while we function like a well-oiled machine, we're still running on 20th century equipment and we are already seeing signs that this is eroding our advantage over the competition…we need to bring our operation into the 21st century and that starts with FastRide."

What Does the Customer Already Have?

Crazy Taxi does not want to manage a more complex of a system than absolutely necessary. They already have three web and app developers, who have built their marketing pages as well as few proof of

concept apps for iOS and Android, and one of them is quite savvy with .NET and brings some backend experience to the table. However, the executives all agree that this one developer cannot deliver the entire backend alone, so they are looking for a solution that provides them a "back end in a box" as they cannot afford to hire any more developers. Additionally, headquarters wants their IT team to have easy visibility into the health of the system, and more importantly that the system can take care of itself. Specifically, they are familiar with the notion of pay-as-you-go billing, and would love a solution that operates at nominal cost during their average daytime load, but on Friday night and Saturday night automatically handles the increased demand, albeit at an additional cost.

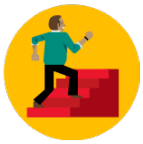
What Is the Customer's Goal?

FastRide is a software solution that the company has been planning to build that uses the GPS of 4G enabled mobile devices in each Crazy Taxi to monitor the location of each vehicle in the Crazy Taxi fleet. By being able to visualize how their fleet is moving throughout the city in real time, FastRide will allow Crazy Taxi Cab Co. to optimize their driver coverage throughout the city. When a new customer calls in, the telephone dispatcher enters their pickup location and the FastRide system identifies the closest available driver, instead of requiring dispatch to manually poll all of the drivers over the radio. While stationary, a driver accepts a fare using the FastRide app on his mobile device. FastRide will provide the driver with the pick-up details such as location, passenger name and callback phone (if available). Upon arrival the destination, the device will compute the fare automatically based on time in transit and distance driven, all with intent of minimizing driver distraction while driving due to interacting with the device. The fare information should be collected on the device and uploaded to a central database in near real-time. Should the driver enter a pocket without cellular connectivity, the device should store the data offline and sync it as soon as connectivity is restored.

The fare data is associated with each driver, who logs in in thru his or her app using his or her own Microsoft Account, Google, Twitter or Facebook credentials. It is the intent that driver fare data will be used to track the driver's progress against established company goals, which both the company and driver access in the form of reports. Because these reports may grow to involve more complex computations (such as comparing the driver's selected route to similar routes taken by other drivers), these data powering the reports will be computed nightly and made available the following day.

While initially notifications would be sent only to drivers about potential fares they can choose to accept, Crazy Taxi is looking forward to when they can offer their patrons a mobile app that can receive notifications about when the driver is in route, ETA updates and any messaging from the driver to the customer. They would like to be certain that the system they build on today has the capacity to meet those increased demands in the future.

Who are the business decision makers and stakeholders?



What customer business needs do you need to address with your solution?

Diagram your proposed solution

What Does the Customer Need?

• A back-end as a service that provides both data storage, API support and push notifications.

• A solution that is quick to implement with a small team that has limited back-end development experience and capacity.

• A back-end solution that can be implemented using .NET.

• A solution that is easy to monitor for health and telemetry data.

What Things Worry the Customer?

• Doesn't Azure Mobile Apps only work on Windows devices?

• Our development team doesn't know node.js. We had heard mention of Mobile Apps, but thought it only supported JavaScript backends.

• Our development team seems to think implementing push notifications using Apple and Android apps directly is easy, but we (as the executives) aren't so sure. How difficult can it be?

• Can't we just build all of our backend using Azure Web Apps?

Case Study Solution

Preferred Target Audience

Operations Management Team led by Christopher Giovanni, Chief Operating Officer at Crazy Taxi Cab Co.

Preferred Solution

Crazy Taxi Cab Co. liked the idea of a highly scalable, agile solution that they could both execute on and manage with a small IT team with limited back-end development experience in .NET. The company's Microsoft representative introduced them to Azure Mobile Apps, a cloud based mobile platform that provides backend CRUD data services, offline data sync for mobile devices, custom REST API services,

push notifications and services for login via a social identity provider. This was exactly the "back-end in a box" that they were looking for.

Crazy Taxi Cab Co. leveraged the many features of Mobile Apps available to them with the Standard tier in order to minimize their backend development burden and accelerate solution delivery.

The implementation of the proposed Azure Mobile Apps solution would create a strategic partnership that will help Crazy Taxi Cab Co. to overcome its challenges with:

- Minimizing system downtime.
- Sending multiple user specific notifications to mobile devices.
- Managing user accounts leveraging social identity providers, such as Microsoft Account, Facebook, and Twitter.
- 24/7 (secure) data accessibility throughout the Crazy Taxi Cab Co. network.
- Scalability in software solutions in an agile marketplace.

As the FastRide system continues to be improved upon, Mobile Apps will help to inject velocity into the development cycle by providing a mobile back end to the application. Mobile Apps offers cross platform compatible components, which gives Crazy Taxi Cab Co. the flexibility to change their mobile platform as the needs of their business dictate. This "back-end as a service" approach will allow Crazy Taxi to focus on building an app that merges the right functionality with a great user experience for each market the operate in.

By utilizing push notifications, Crazy Taxi Cab Co. can optimize their customer pickup messages through the FastRide app. This allows for faster, and more streamlined in-app communication. When a new fare's pickup address is entered into FastRide by a dispatcher, or the customer facing mobile app, Mobile Apps will enable FastRide to automatically send a notification to the closest available driver—eliminating the need for manual notifications. Since push notifications can be managed, each base will have control over the messages sent to its drivers.

To propose a more complete solution and ensure deployment success, it would be helpful to know:

- Type and operating system of tablets being used.
- Expected product life of current tablet choice.
- Current number of dashboard tablets.
- Projected number of tablets after the planned expansion this year and next year.
- Current average number of fares completed per day, week, month, year.
- Projected average number of fares completed per day, week, month, year after the expansion into new markets.
- Rate of growth across bases (customer and driver).
- Other software products used to operate the company.

Understanding these details and decisions will help identify the current and future software, hardware, and infrastructure needs of Crazy Taxi Cab Co., and to provide solutions that are consistent with their short and long-term business goals.

High-Level Architecture

Crazy Taxi Cab Co. leveraged the many features of Mobile Apps available to them with the Standard tier in order to minimize their backend development burden and accelerate solution delivery.

• **Authentication:** Drivers login to the FastRide app on their device using their Microsoft, Google, Twitter or Facebook credentials, the federated login process being handled by Mobile Apps in conjunction with the aforementioned identity providers.

• **Notifications**: Once logged in, the app registers with Mobile Apps, associating the driver's identity with the Notification Hub (associated with the Mobile App). In this way, Crazy Taxi dispatch can send broadcast notifications to all drivers, but still be able to send targeted Fare Alert messages to a particular driver.

By having Mobile Apps in place with Push Notifications, Crazy Taxi Cab Co. is well positioned in the future to light up the ability to deliver a customer-oriented app that deliver push notifications to customers informing them of events relevant to their pickup.

• **Offline Data:** For the driver's iOS and Android devices, in the construction of the FastRide app, they leveraged the native client Offline Data Sync functionality for synchronizing Fare Data when temporarily disconnected from the cellular network. This Fare Data is stored using Tables in Mobile Apps, which ultimately rests as relational tables in a SQL Database. This SQL Database also stores the driver profiles (that associate social credentials with driver profile information).

• **Back End Custom Services**: When a driver accepts or declines a fare received via a push notification, that message is sent using a Custom REST API hosted by Mobile Apps and built using ASP.NET Web API.

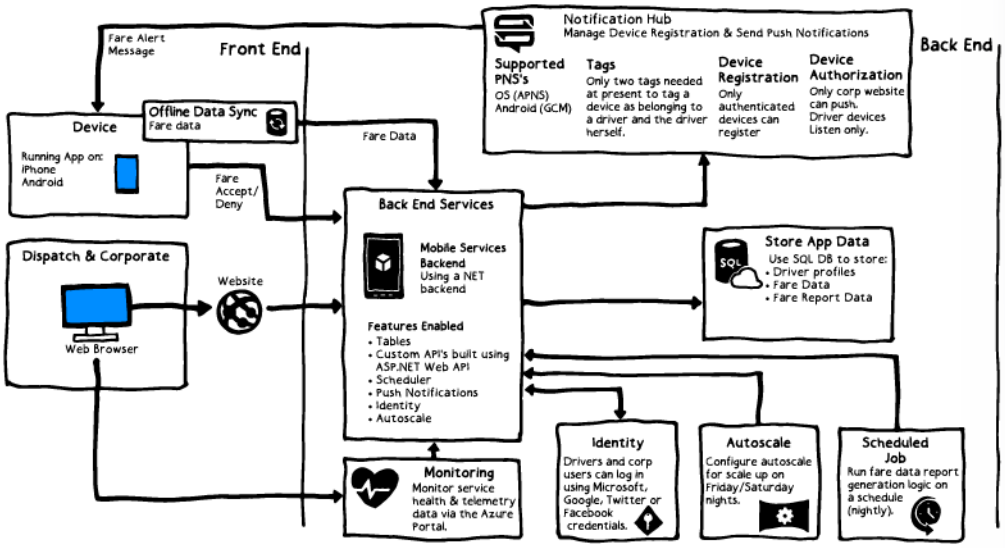
• **Front-End Website:** Dispatch uses a website hosted in Azure Websites to manage the taxi cab dispatch process. The Notification Hub is configured so that only the dispatch website is allowed to send push notifications
to the drivers (the FastRide app for drivers is Listen only).

• **Monitoring**: Crazy Taxi corporate IT monitors the health of the solution using the Dashboard for the Mobile App or Website in the Azure Portal. To assist the IT team with visibility into the health of the system, they should configure monitoring endpoints, again using the Azure Portal, on their website and Mobile Apps and enable e-mail alerts should the Response Time and Uptime metrics for those fall below values they deem acceptable.

• **Scaling Configuration**: They have configured Autoscale on their Mobile App, via the Scale tab in the portal, to double the number of instances it uses on Friday and Saturday night in order to handle the increased load, then return back to their normal instance count for the rest of the week.

• **Backend Jobs**: They have also created a Mobile Apps Scheduled Job that processes the Fare Data on nightly basis to generate the data sets that power the Fare Reports. This data is stored in the same SQL Database that stores all the other data used by the solution.

Preferred Solution Diagram



Checklist of Preferred Objection Handled

• Doesn't Azure Mobile Apps only work on Windows devices?

Azure Mobile Apps provides native clients for iOS, Android, Xamarin, PhoneGap, Sencha and Appcelerator in addition to Windows universal C#, Windows universal JavaScript and Windows Phone. In addition, Azure platform services offer REST APIs that extend the reach to platforms for which there is not a native API, but are capable of making REST style requests.

• Our development team doesn't know node.js. We had heard mention of Mobile Apps, but thought it only supported JavaScript backends.

Mobiles Services supports using .NET for the backend logic, and node.js (or JavaScript logic) does not have to be used anywhere in the backend code.

• Our development team seems to think implementing push notifications using Apple and Android apps directly is easy, but we (as the executives) aren't so sure. How difficult can it be?

While using the Push Notification System of a particular device platform directly is typically made fairly simple by the provider of that platform (e.g., iOS apps have a straightforward API for leveraging Apple's Push Notification System), this simplicity is only true for the individual device and does not address the complete solution that requires at minimum a backend managing device registrations at scale and sending push notifications cross platforms in a timely fashion. Azure Mobile Apps provides that backend functionality, which can be easily scaled to meet demand.

• Can't we just build all of our backend using Azure Web Apps?

Azure Web Apps is effectively a superset of Mobile Apps and so can be used to implement the backend for Mobile Applications. However, Web Apps do not deliver the services tailored for the mobile application scenario, requiring the developers to write their own logic to integrate with Notification Hubs, SQL Database, Identity services and WebJobs. Additionally, Mobile Apps is prescriptive in the patterns used for developing custom API's, and so speeds the development of such API's by allowing the development efforts to focus on the business logic instead of dealing with structural and hosting decisions. These become important factors to consider when taking into account the development team size, capabilities and timeframe.

Proof Of Concept (POC) Checklist

The primary items a Proof of Concept for this solution could demonstrate include:

• Scalability / Scheduled Autoscale.

• Mobile Apps ease of integration (e.g., the backend in a box).

• Streamlined communication with Push notifications.

• Integration of social identity platforms to aid in customer authentication and profile management.

• Device offline data storage and synchronization.

• Monitoring solution health.

The Proof of Concept will be considered successful if the Crazy Taxi Operations Management Team believes they can realize value in:

• Speeding up the delivery of the overall solution.

• Push notifications to streamline communication and send fare updates to tablet devices.

• Authenticating users via social media platforms and future benefits of successfully leveraging social media integration.

• Minimizing system downtime by keeping app data in the cloud.

• Scalability in a mobile cloud solution.

The personnel resources you would leverage to build the PoC, may include:

• Partner Resources in the Region or MCS to help assist with migration design and implementation.

• Microsoft Azure CAT for level 300 expertise requests with Azure.

Online Lab - Deploying Serverless Workloads to Azure

Lab Steps

Online Lab - Deploying Serverless Workloads to Azure

NOTE: For the most recent version of this online lab, see: **https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign**

Before we start

1. Ensure that you are logged in to your Windows 10 lab virtual machine using the following credentials:
   - Username: **Admin**
   - Password: **Pa55w.rd**
2. Review Taskbar located at the bottom of your Windows 10 desktop. The Taskbar contains the icons for the common applications you will use in the labs:
   - Microsoft Edge
   - File Explorer
   - **Visual Studio Code**[1]
   - **Microsoft Azure Storage Explorer**[2]
   - Bash on Ubuntu on Windows
   - Windows PowerShell
3. **Note**: You can also find shortcuts to these applications in the **Start Menu**.

3

Exercise 1: Create Web App

4

Task 1: Open the Azure Portal

1. On the Taskbar, click the **Microsoft Edge** icon.
2. In the open browser window, navigate to the **Azure Portal** (**https://portal.azure.com**).

1 https://code.visualstudio.com/
2 https://azure.microsoft.com/features/storage-explorer/
3 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod03_Deploying%20Serverless%20Workloads%20to%20Azure.md#exercise-1-create-web-app
4 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod03_Deploying%20Serverless%20Workloads%20to%20Azure.md#task-1-open-the-azure-portal

3. If prompted, authenticate with the user account account that has the owner role in the Azure subscription you will be using in this lab.

5

Task 2: Open Cloud Shell

1. At the top of the portal, click the **Cloud Shell** icon to open a new shell instance.
2. **Note**: The **Cloud Shell** icon is a symbol that is constructed of the combination of the *greater than* and *underscore* characters.
3. If this is your first time opening the **Cloud Shell** using your subscription, you will see a wizard to configure **Cloud Shell** for first-time usage. When prompted, in the **Welcome to Azure Cloud Shell** pane, click **Bash (Linux)**.
4. **Note**: If you do not see the configuration options for **Cloud Shell**, this is most likely because you are using an existing subscription with this course's labs. If so, proceed directly to the next task.
5. In the **You have no storage mounted** pane, click **Show advanced settings**, perform the following tasks:
   - Leave the **Subscription** drop-down list entry set to its default value.
   - In the **Cloud Shell region** drop-down list, select the Azure region matching or near the location where you intend to deploy resources in this lab
   - In the **Resource group** section, ensure that the **Create new** option is selected and then, in the text box below, type **AADesignLab0901-RG**.
   - In the **Storage account** section, ensure that the **Create new** option is selected and then, in the text box below, type a unique name consisting of a combination of between 3 and 24 characters and digits.
   - In the **File share** section, ensure that the **Create new** option is selected and then, in the text box below, type **cloudshell**.
   - Click the **Create storage** button.
6. Wait for the **Cloud Shell** to finish its first-time setup procedures before you proceed to the next task.

6

Task 3: Create an App Service plan

1. At the **Cloud Shell** command prompt at the bottom of the portal, type in the following command and press **Enter** to create a variable which value designates the name of the resource group you will use in this exercise:

```
RESOURCE_GROUP_APP='AADesignLab0502-RG'
```

5 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod03_Deploying%20Serverless%20Workloads%20to%20Azure.md#task-2-open-cloud-shell

6 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod03_Deploying%20Serverless%20Workloads%20to%20Azure.md#task-3-create-an-app-service-plan

2. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a variable which value designates the Azure region you will use for the deployment (replace the placeholder `<Azure region>` with the name of the Azure region to which you intend to deploy resources in this lab):

```
LOCATION='<Azure region>'
```

3. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create the resource group:

```
az group create --name $RESOURCE_GROUP_APP --location $LOCATION
```

4. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a new App Service plan:

```
az appservice plan create --is-linux --name "AADesignLab0502-$LOCATION"
--resource-group $RESOURCE_GROUP_APP --location $LOCATION --sku B2
```

[7]

Task 4: Create a Web App instance

1. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to view a list of possible runtimes for a Linux-based App Service web app instance:

```
az webapp list-runtimes --linux
```

2. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a new variable which value is a randomly generated string that you will use as the name of a new web app:

```
WEBAPPNAME1=webapp05021$RANDOM$RANDOM
```

3. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a new web app using a unique name:

```
az webapp create --name $WEBAPPNAME1 --plan AADesignLab0502-$LOCATION
--resource-group $RESOURCE_GROUP_APP --runtime "DOTNETCORE|2.1"
```

4. **Note**: In case the command fails due to duplicate web app name, re-run the last two steps until the command completes successfully
5. Wait for the deployment to complete before you proceed to the next task.

7 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod03_Deploying%20Serverless%20Workloads%20to%20Azure.md#task-4-create-a-web-app-instance

8

Task 5: View deployment results

1. In the hub menu in the Azure portal, click **Resource groups**.
2. On the **Resource groups** blade, click **AADesignLab0502-RG**.
3. On the **AADesignLab0502-RG** blade, click the entry representing the Azure web app you created earlier in this exercise.
4. On the web app blade, click the **Browse** button at the top of the blade.
5. Review the default page generated by Azure App Service.
6. Close the new browser tab and return to the browser tab displaying the Azure portal.

Review: In this exercise, you created a Linux-based App Service Plan that contained a blank web app.

9

Exercise 2: Deploy Web App code

10

Task 1: Deploy code with a Web App Extension using an Azure Resource Manager template and GitHub

1. At the top of the portal, click the **Cloud Shell** icon to open a new shell instance.
2. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a variable which value designates the name of the resource group you will use in this exercise:

```
RESOURCE_GROUP_APP='AADesignLab0502-RG'
```

3. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a variable which value designates the Azure region you will use for the deployment:

```
LOCATION=$(az group list --query "[?name == 'AADesignLab0502-RG'].location"
--output tsv)
```

4. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a new variable which value is a randomly generated string that you will use as the name of a new web app:

```
WEBAPPNAME2=webapp05022$RANDOM$RANDOM
```

8 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod03_Deploying%20Serverless%20Workloads%20to%20Azure.md#task-5-view-deployment-results

9 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod03_Deploying%20Serverless%20Workloads%20to%20Azure.md#exercise-2-deploy-web-app-code

10 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod03_Deploying%20Serverless%20Workloads%20to%20Azure.md#task-1-deploy-code-with-a-web-app-extension-using-an-azure-resource-manager-template-and-github

5. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a new web app using a unique name:

```
az webapp create --name $WEBAPPNAME2 --plan AADesignLab0502-$LOCATION
--resource-group $RESOURCE_GROUP_APP --runtime "NODE|9.4"
```

6. **Note**: In case the command fails due to duplicate web app name, re-run the last two steps until the command completes successfully
7. In the **Cloud Shell** pane, click the **Upload/Download files** icon and, in the drop-down menu, click **Upload**.
8. In the **Open** dialog box, navigate to the **\allfiles\AZ-301T03\Module_03\Labfiles\Starter** folder, select the **github.json** file, and click **Open**. The file contains the following Azure Resource Manager template:

```
{
    "$schema": "http://schemas.management.azure.com/schemas/2015-01-01/
deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "webAppName": {
            "type": "string"
        },
        "repositoryUrl": {
            "type": "string"
        },
        "branch": {
            "type": "string",
            "defaultValue": "master"
        }
    },
    "resources": [
        {
            "apiVersion": "2015-08-01",
            "type": "Microsoft.Web/sites",
            "name": "[parameters('webAppName')]",
            "location": "[resourceGroup().location]",
            "properties": {},
            "resources": [
                {
                    "apiVersion": "2015-08-01",
                    "name": "web",
                    "type": "sourcecontrols",
                    "dependsOn": [
                        "[resourceId('Microsoft.Web/Sites', parameters('we-
bAppName'))]"
                    ],
                    "properties": {
                        "RepoUrl": "[parameters('repositoryUrl')]",
                        "branch": "[parameters('branch')]",
                        "IsManualIntegration": true
                    }
```

```
                }
            ]
        }
    ]
}
```

9. In the **Cloud Shell** pane, click the **Upload/Download files** icon and, in the drop-down menu, click **Upload**.
10. In the **Open** dialog box, navigate to the **\allfiles\AZ-301T03\Module_01\Labfiles\Starter** folder, select the **parameters.json** file, and click **Open**. The file contains the following parameters for the Azure Resource Manager template you uploaded previously:

```
{
  "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploy-
mentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "webAppName": {
      "value": "$WEBAPPNAME2"
    },
    "repositoryUrl": {
      "value": "$REPOSITORY_URL"
    },
    "branch": {
      "value": "master"
    }
  }
}
```

11. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a variable which value designates the name of the GitHub repository hosting the web app code:

```
REPOSITORY_URL='https://github.com/Azure-Samples/nodejs-docs-hello-world'
```

12. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a variable which value designates the name of the GitHub repository hosting the web app code and which takes into account any special character the URL might include:

```
REPOSITORY_URL_REGEX="$(echo $REPOSITORY_URL | sed -e 's/\\/\\\\/g;
s/\//\\\//g; s/&/\\\&/g')"
```

13. **Note**: This is necessary because you will use the **sed** utility to insert this string into the Azure Resource Manager template parameters file. Alternatively, you could simply open the file and enter the URL string directly into the file.
14. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to replace the placeholder for the value of the **webAppName** parameter with the value of the **$WEBAPPNAME2** variable in the parameters file:

```
sed -i.bak1 's/"$WEBAPPNAME2"/"'"$WEBAPPNAME2"'"/' ~/parameters.json
```

15. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to replace the placeholder for the value of the **repositoryUrl** parameter with the value of the **$REPOSITORY_URL** variable in the parameters file:

```
sed -i.bak2 's/"$REPOSITORY_URL"/"'"$REPOSITORY_URL_REGEX"'"/' ~/parameters.json
```

16. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to verify that the placeholders were successfully replaced in the parameters file:

```
cat ~/parameters.json
```

17. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to deploy the GitHub-resident web app code by using a local Azure Resource Manager template and a local parameters file:

```
az group deployment create --resource-group $RESOURCE_GROUP_APP --template-file github.json --parameters @parameters.json
```

18. Wait for the deployment to complete before you proceed to the next task.
19. **Note**: The deployment should take about a minute.

11

Task 2: View deployment results

1. In the hub menu in the Azure portal, click **Resource groups**.
2. On the **Resource groups** blade, click **AADesignLab0502-RG**.
3. On the **AADesignLab0502-RG** blade, click the entry representing the Azure web app you created in the previous task.
4. On the web app blade, click the **Browse** button at the top of the blade.
5. Review the sample Node.js web application deployed from GitHub.
6. Close the new browser tab and return to the browser tab displaying the Azure portal.

12

Task 3: Deploy Code with a Docker Hub container image

1. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a variable which value designates the name of the resource group you will use in this task:

```
RESOURCE_GROUP_CONTAINER='AADesignLab0502-RG'
```

11 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod03_Deploying%20Serverless%20Workloads%20to%20Azure.md#task-2-view-deployment-results

12 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod03_Deploying%20Serverless%20Workloads%20to%20Azure.md#task-3-deploy-code-with-a-docker-hub-container-image

2. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a variable which value designates the Azure region you will use for the deployment:

```
LOCATION=$(az group list --query "[?name == 'AADesignLab0502-RG'].location"
--output tsv)
```

3. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a new variable which value is a randomly generated string that you will use as the name of a new web app:

```
WEBAPPNAME3=webapp05023$RANDOM$RANDOM
```

4. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to create a new web app using a unique name:

```
az webapp create --name $WEBAPPNAME3 --plan AADesignLab0502-$LOCATION
--resource-group $RESOURCE_GROUP_CONTAINER --deployment-container-image
ghost
```

5. **Note**: In case the command fails due to duplicate web app name, re-run the last two steps until the command completes successfully
6. Wait for the deployment to complete before you proceed to the next task.
7. **Note**: The deployment should take less than a minute.
8. Close the **Cloud Shell** pane.

13

Task 4: View deployment results

1. In the hub menu in the Azure portal, click **Resource groups**.
2. On the **Resource groups** blade, click **AADesignLab0502-RG**.
3. On the **AADesignLab0502-RG** blade, click the entry representing the Azure web app you created in the previous task.
4. On the web app blade, click the **Browse** button at the top of the blade.
5. **Note**: If the application does not appear, switch to the web app blade, click **Restart** button at the top of the blade and then click **Browse** again.
6. Review the blog application deployed from Docker Hub.
7. Close the new browser tab and return to the browser tab displaying the Azure portal.

Review: In this exercise, you deployed code using an Azure Resource Manager template and a Docker Hub image to App Service web apps.

13 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod03_Deploying%20Serverless%20Workloads%20to%20Azure.md#task-4-view-deployment-results

14

Exercise 3: Deploy a Function App

15

Task 1: Deploy a Function App with code using an Azure Resource Manager template

1. In the upper left corner of the Azure portal, click **Create a resource**.
2. At the top of the **New** blade, in the **Search the Marketplace** text box, type **Template Deployment** and press **Enter**.
3. On the **Everything** blade, in the search results, click **Template deployment**.
4. On the **Template deployment** blade, click the **Create** button.
5. On the **Custom deployment** blade, click the **Build your own template in the editor** link.
6. On the **Edit template** blade, click the **Quickstart template** link.
7. In the **Load a quickstart template** pane, in the **Select a template** drop-down list, select the **201-function-app-dedicated-github-deploy** template.
8. Click the **OK** button.
9. Back on the **Edit template** blade, click the **Save** button to persist the template.
10. Back on the **Custom deployment** blade, perform the following tasks:
    - Leave the **Subscription** drop-down list entry set to its default value.
    - In the **Resource group** section, select the **Use existing** option.
    - In the **Resource group** section, ensure that **Create new** option is selected and, in the text box below, type **AADesignLab0503-RG**.
    - In the **App Name** text box, type a unique name for the new Function App.
    - In the **Sku** drop-down list, select the **Basic** option.
    - Leave the **Worker Size** drop-down list set to its default value.
    - Leave the **Storage Account Type** drop-down list set to its default value.
    - Leave the **Repo URL** field set to its default value.
    - Leave the **Branch** text box set to its default value.
    - Leave the **Location** text box set to its default value.
    - In the **Terms and Conditions** section, select the **I agree to the terms and conditions stated above** checkbox.
    - Click the **Purchase** button.

14 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod03_Deploying%20Serverless%20Workloads%20to%20Azure.md#exercise-3-deploy-a-function-app

15 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod03_Deploying%20Serverless%20Workloads%20to%20Azure.md#task-1-deploy-a-function-app-with-code-using-an-azure-resource-manager-template

11. Wait for the deployment to complete before you proceed to the next task.
12. **Note**: The deployment should take about a minute.

16

Task 2: View deployment results

1. In the hub menu in the Azure portal, click **Resource groups**.
2. On the **Resource groups** blade, click **AADesignLab0503-RG**.
3. On the **AADesignLab0503-RG** blade, click the entry representing the Function App you created in the previous task.
4. On the Function App blade, locate the **Url** entry and click the hyperlink below to see the Function App landing page in a new browser tab.
5. Close the new browser tab and return to the browser tab displaying the Azure portal.

Review: In this exercise, you deployed a Function App and code using an Azure Resource Manager template.

17

Exercise 4: Remove lab resources

18

Task 1: Open Cloud Shell

1. At the top of the portal, click the **Cloud Shell** icon to open the Cloud Shell pane.
2. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to list all resource groups you created in this lab:

```
az group list --query "[?starts_with(name,'AADesignLab05')]".name --output tsv
```

3. Verify that the output contains only the resource groups you created in this lab. These groups will be deleted in the next task.

16 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod03_Deploying%20Serverless%20Workloads%20to%20Azure.md#task-2-view-deployment-results-1
17 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod03_Deploying%20Serverless%20Workloads%20to%20Azure.md#exercise-4-remove-lab-resources
18 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod03_Deploying%20Serverless%20Workloads%20to%20Azure.md#task-1-open-cloud-shell

19

Task 2: Delete resource groups

1. At the **Cloud Shell** command prompt, type in the following command and press **Enter** to delete the resource groups you created in this lab

```
az group list --query "[?starts_with(name,'AADesignLab05')]".name --output
tsv | xargs -L1 bash -c 'az group delete --name $0 --no-wait --yes'
```

2. Close the **Cloud Shell** prompt at the bottom of the portal.

Review: In this exercise, you removed the resources used in this lab.

19 https://github.com/MicrosoftLearning/AZ-301-MicrosoftAzureArchitectDesign/blob/master/Instructions/AZ-301T03_Lab_Mod03_Deploying%20Serverless%20Workloads%20to%20Azure.md#task-2-delete-resource-groups

Review Questions

Module 3 Review Questions

Serverless computing

You need to design a serverless framework to host applications.

What are the core benefits of Web Apps? What other types of App Services does Azure offer?

Suggested Answer ↓

Web Apps are a fully managed and low friction PaaS offering to host your web application in the Azure platform. The serverless Azure App Services include Web Apps, Mobile Apps, and APIs.

Serverless computing (Azure Functions)

You architect a solution for your company.

You need to build a serverless processing framework.

Which Azure service provides serverless processing? What are the two models for using this service?

Suggested Answer ↓

Azure Functions are designed to make it faster to process events and distribute output to services that interact with the functions. You can create Function Apps that use the App Service or Consumption models.

Integration offerings

You are designing a solution that integrates several different applications and services. You plan to implement an Azure Platform-as-a-Service (PaaS) solution.

What integration solutions does Azure offer?

Suggested Answer ↓

Azure provides API Management and Logic Apps as integration PaaS solutions. Azure API Management helps organizations publish APIs to external partner and internal developers. Logic Apps provide workflows, designed using JOSN, that can connect various components of your application together using minimal or no code.