Resource Manager Templates
App Services, Azure SQL

Lecture 05

Deep Azure @ McKesson

Zoran B. Djordjević

Structure of Azure, revisited

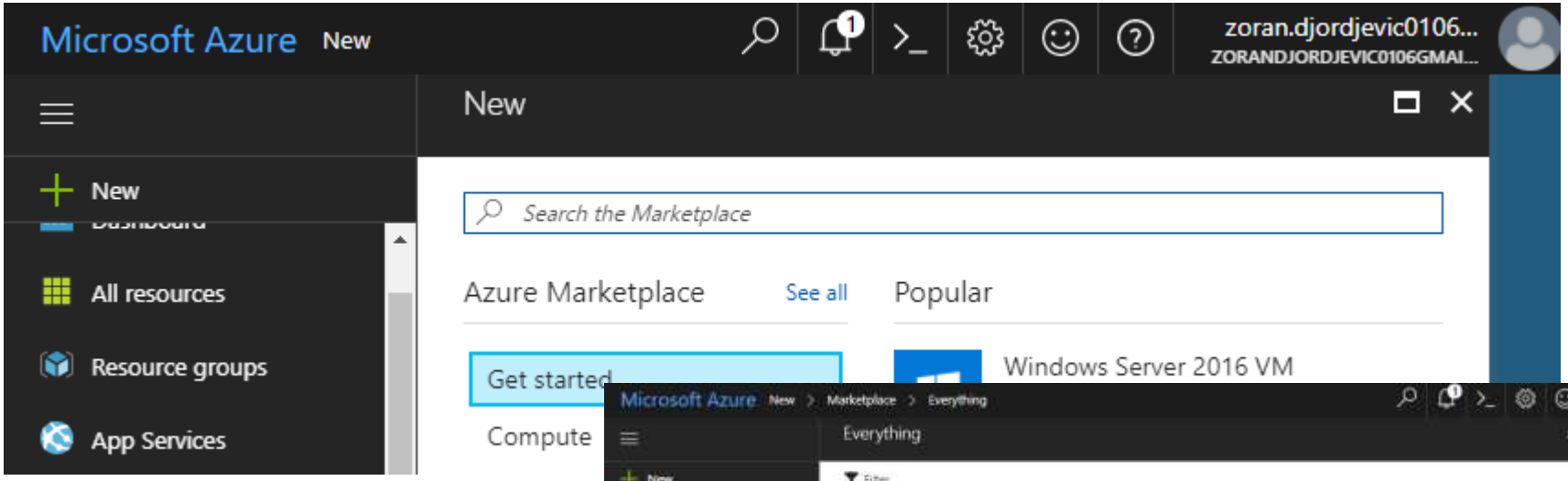
- The Azure platform consists primarily of three parts:
  - **Azure Execution Model**, denotes the areas where you can provide your services and applications
  - **Azure Application Building Blocks** and
  - **Azure Data Services**
- most users see only these three parts.
- Many other services are working under the hood of the platform and ensure the ongoing operation. These services include, for example, the
  - Azure traffic manager,
  - Azure load balancer, and the
  - Azure resource manager,
  - Azure security services
  - Azure active directory, and others
- It appears that Microsoft is positioning Azure Resource Manger as the primary tool for deployment and management of other Azure resources and services.

Resource Manager, Continued

- We will continue detailed analysis of Azure resource manager and explore the following topics:
  - Azure resource tags
  - Azure resource locks
  - Working with ARM templates and the Azure resource explorer
  - Creating your own ARM template
- Before moving to new topics we will briefly review RM features we covered last time.

Resource Group

- In the portal, click on **New** and then click on **See all** in **MARKETPLACE**



- Now the Azure marketplace opens.

  Type **Template deployment** in Search

Library of Template Development Offers

- Large library of templates appears. Click and explorer. Many of offerings r familiar.

Filter

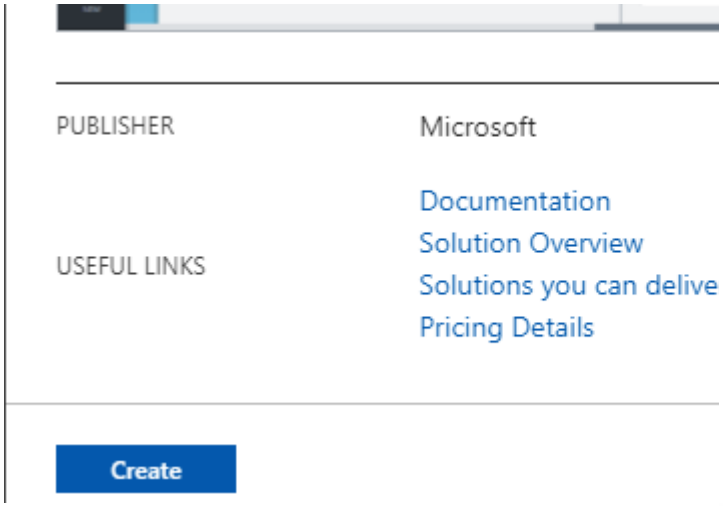
Template development

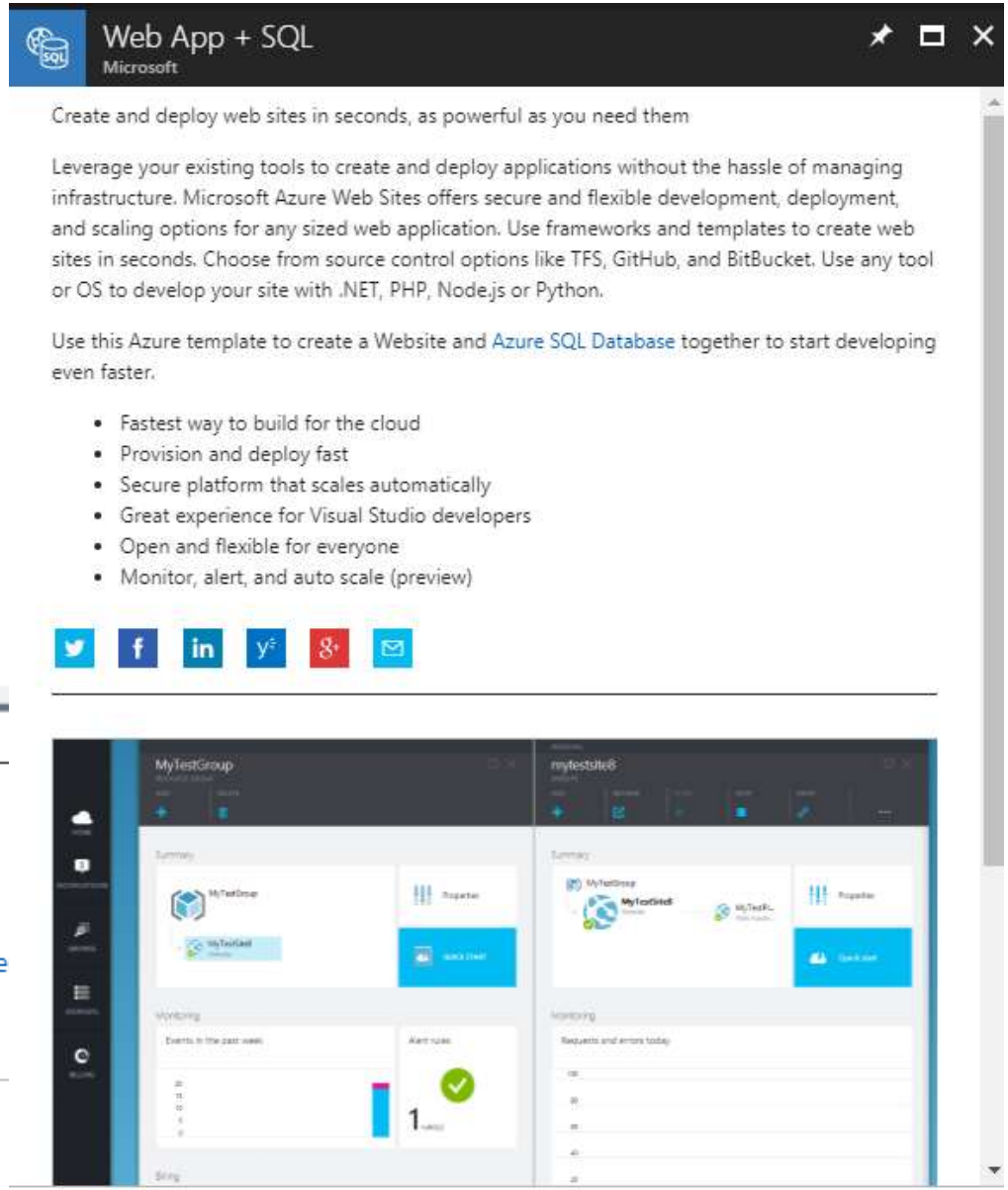
Results

NAME	PUBLISHER	CATEGORY
PrestaShop Kickstart Template	PrestaShop SA	Compute
PrestaShop Advanced Template	PrestaShop SA	Compute
PrestaShop Performance Template	PrestaShop SA	Compute
STRATO Blockchain Multinode - Developer Edition	BlockApps	Compute
STRATO Blockchain Singlenode - Developer Edition	BlockApps	Compute
Xcalar Data Platform Template	Xcalar, Inc.	Compute
Web App	Microsoft	Web + Mobile
DevTest Labs for Blockchain as a Service	Microsoft	Developer tools
Ethereum Studio	ether.camp	Compute
Confluent Enterprise 3.x	Confluent	Compute
DevTest Labs	Microsoft	Developer tools

Select for example: Web App + SQL

- Web App + SQL is an old friend. Info page opens
- Scroll down to see useful links.
- Explore:
  - Solution Overview
  - Pricing Details
  - Documentation
- Later we will hit Create button on the bottom left





App Service pricing

- This is a Marketplace. Select and purchase the product you need:

Why Azure | Solutions | Products | Documentation | Pricing | Training | Marketplace | Partners | Blog | Resources | Support | FREE ACCOUN

App Service pricing

Create web and mobile apps for any platform

Explore: Pricing options | Calculator | App Service overview | Documentation

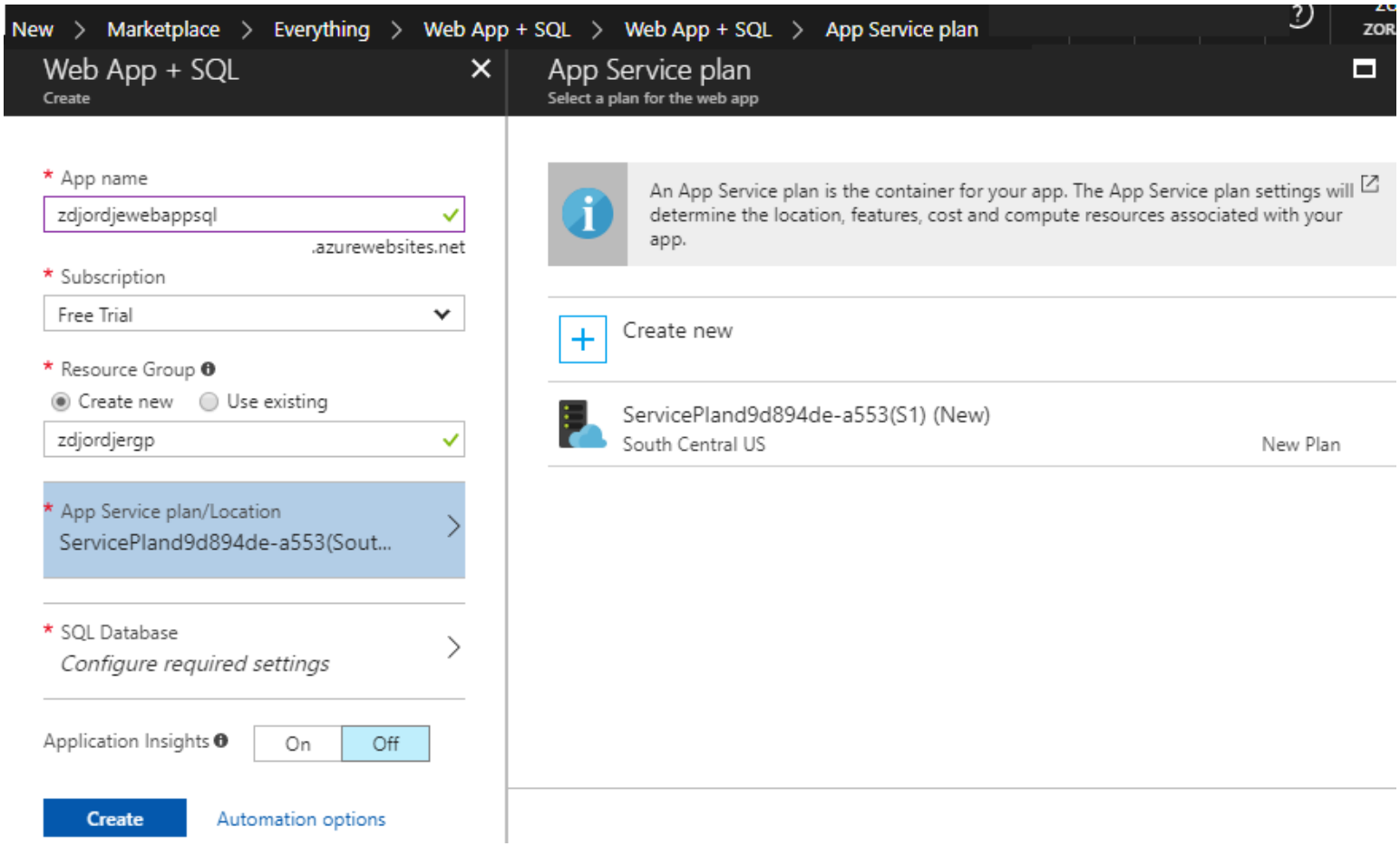
Azure App Service brings together everything you need to create websites, mobile backends, and web APIs for any platform or device. Free, Shared (preview), and Basic plans provide different options to test your apps within your budget. Standard and Premium plans are for production workloads and run on dedicated Virtual Machine instances. E instance can support multiple application and domains. The Isolated plan hosts your apps in a private, dedicated Azure environment and is ideal for apps that require secure connections with your on-premises network, or additional performance and scale.

	FREE Try for free	SHARED Host basic apps	BASIC Dedicated environment for dev/test	STANDARD Run production workloads	PREMIUM Enhanced performance and scale	ISOLATED High-Performan Security and Isolation
Web, mobile, or API apps	10	100	Unlimited	Unlimited	Unlimited	Unlimited
Disk space	1 GB	1 GB	10 GB	50 GB	250 GB	1 TB
Maximum instances	–	–	Up to 3	Up to 10	Up to 20	Up to 100*

- After informing ourselves we want to create an instance of selected offering.
- Let us hit the `Select` button.

Custom Deployment Blade

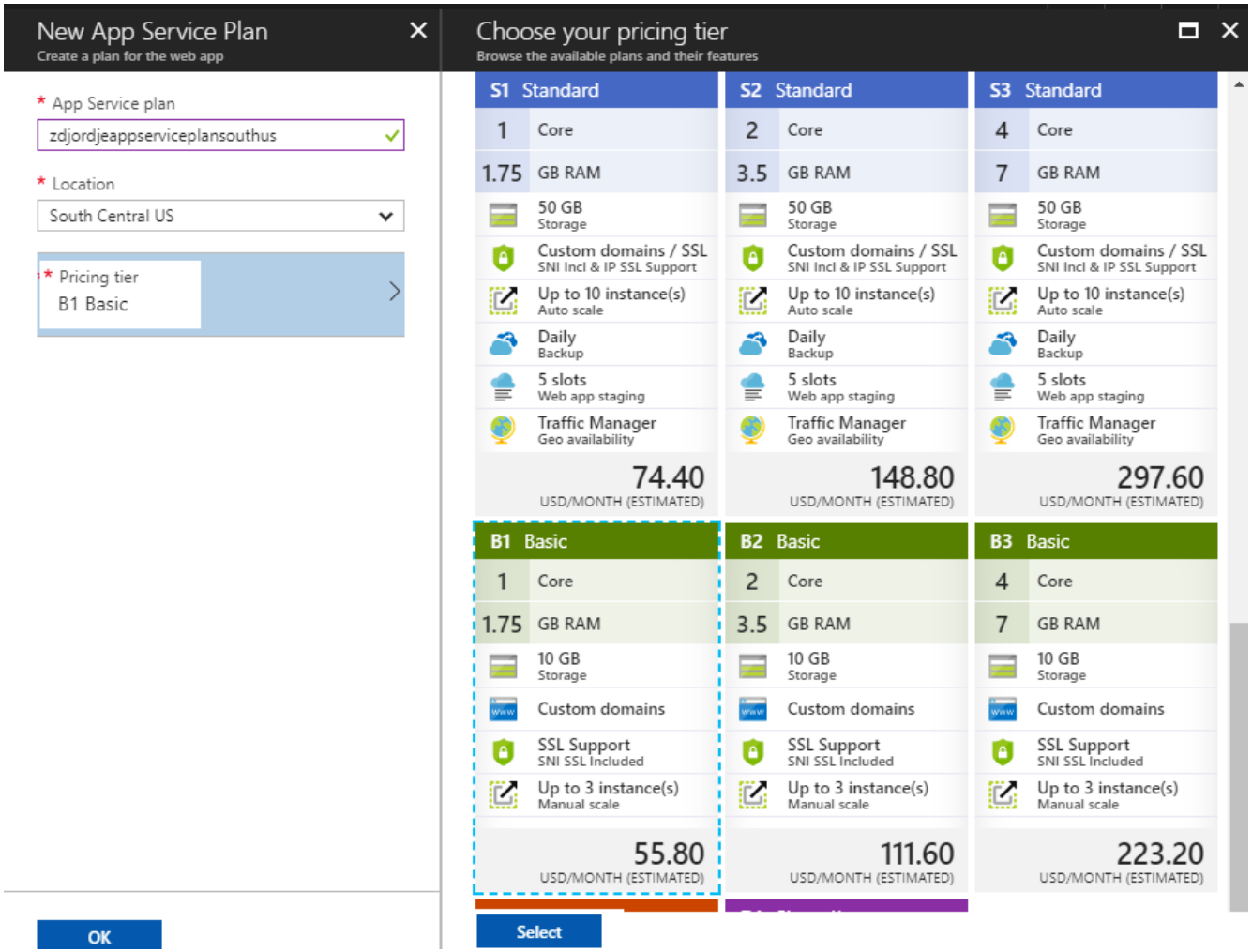
- On new wizard (deployment blade) we have to enter a few fields. If we have an existing group we want to use we can do so or we create a new one.
- You can accept an offered Service Plan or you might prefer to create a new one with new name and location. Select `Create new (App Service plan)`



Service Plan

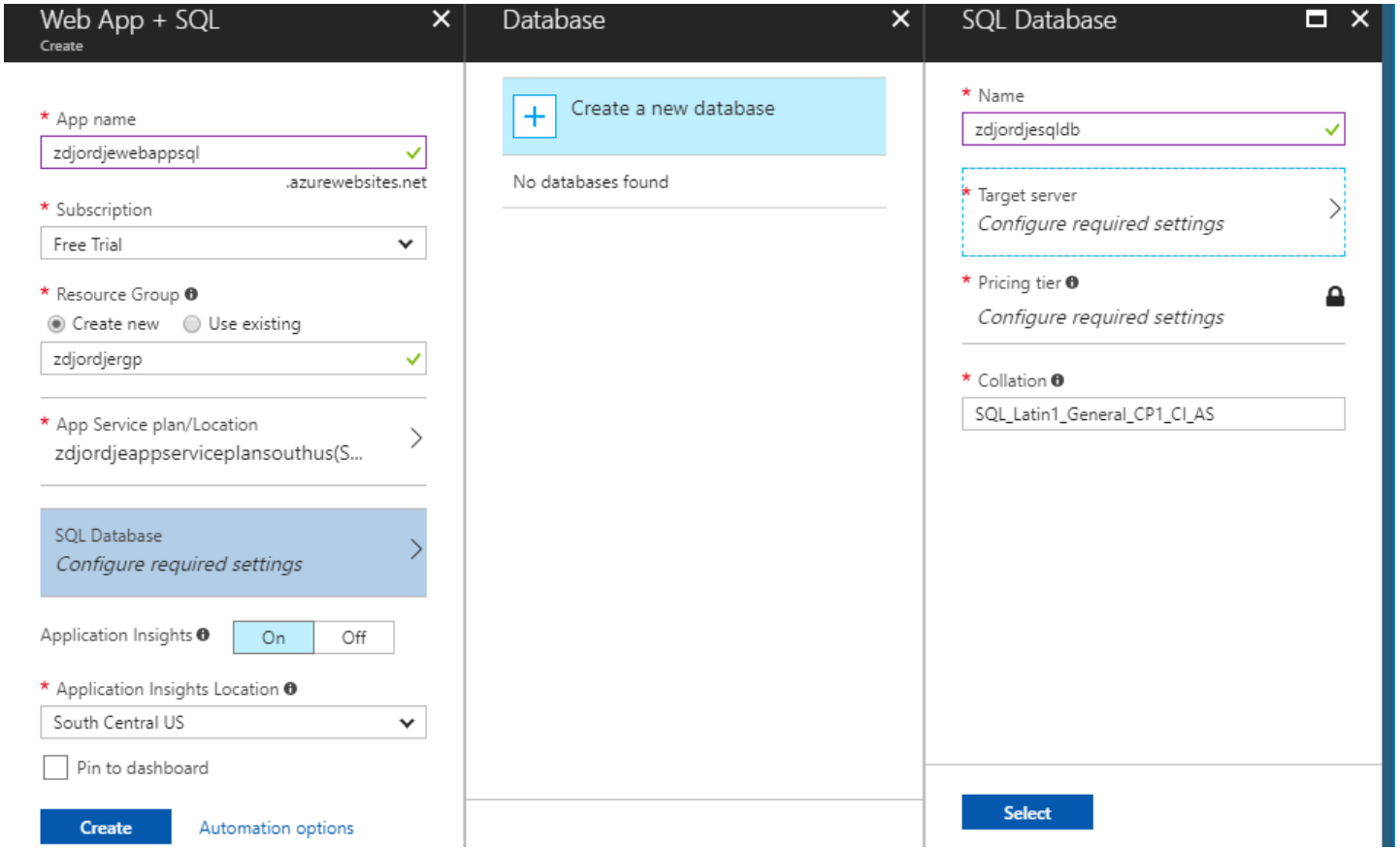
- Provide `name and location`. Under `Choose your pricing tier` select your tier and hit `Select`.

- You should study those different pricing tiers and select one that matches your needs.
- Here we are selecting the least expensive tier with standard features.
- Then, under `New App Service Plan`, select `OK`.



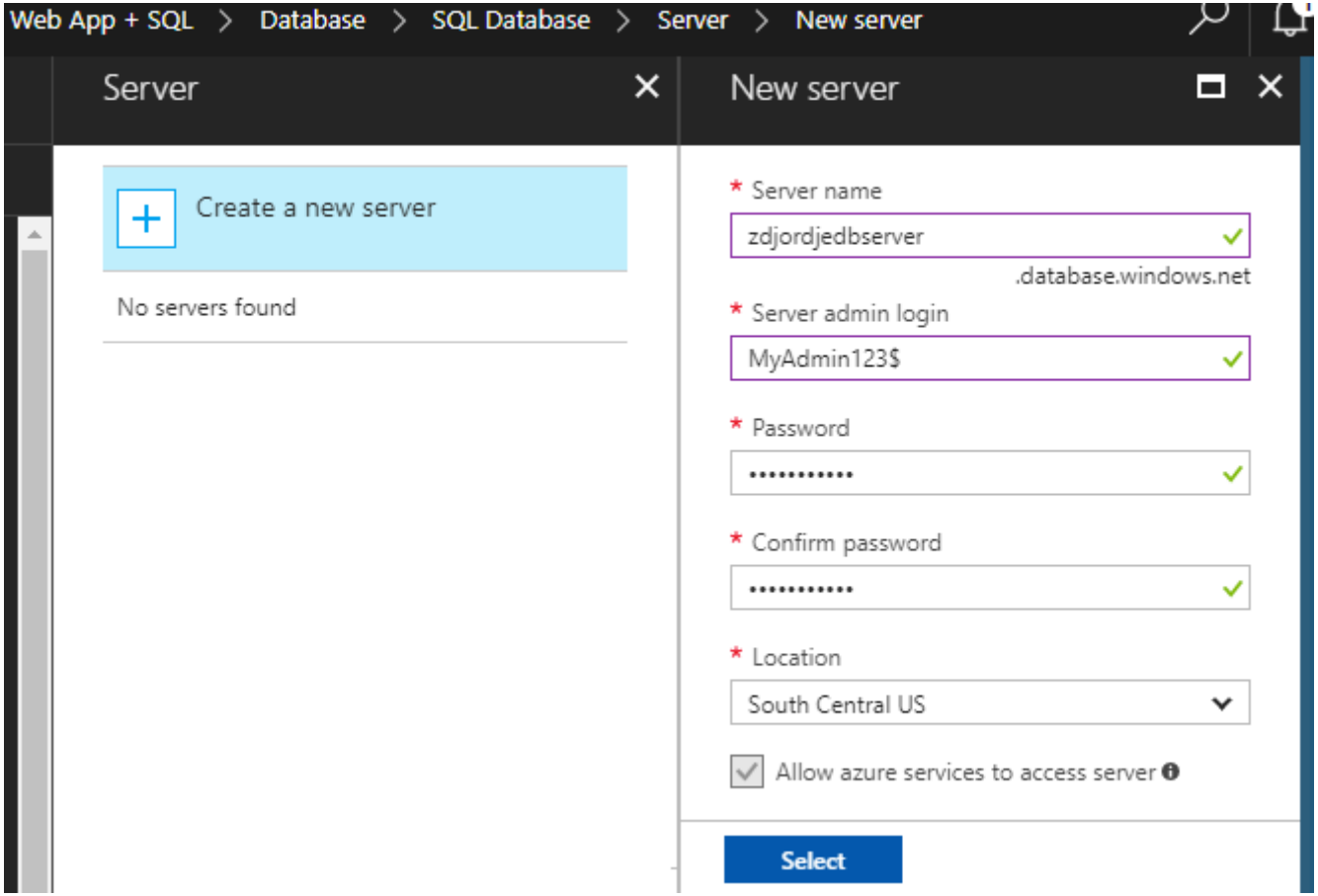
Configure SQL Database

- Next, select (>) next to `SQL Database, Configure required settings`, and on the next page hit `Create a new database.`
- Provide name of new database and then select `Target server, Configure required settings.`



Configure new Server

- Provide requested fields. Unfortunately, even username has to be long and special. When done h
- it `Select` on the bottom of the screen



Select Configured SQL Database

Web App + SQL
Create

* App name
zdjordjewebappsql
.azurewebsites.net

* Subscription
Free Trial

* Resource Group
Create new Use existing
zdjordjergp

* App Service plan/Location
zdjordjeappserviceplansouthus(S...

SQL Database
Configure required settings

Application Insights On Off

* Application Insights Location
South Central US

Pin to dashboard

Create Automation options

Database

Create a new database

No databases found

SQL Database

* Name
zdjordjesqldb

* Target server
zdjordjedbserver (South Central ...

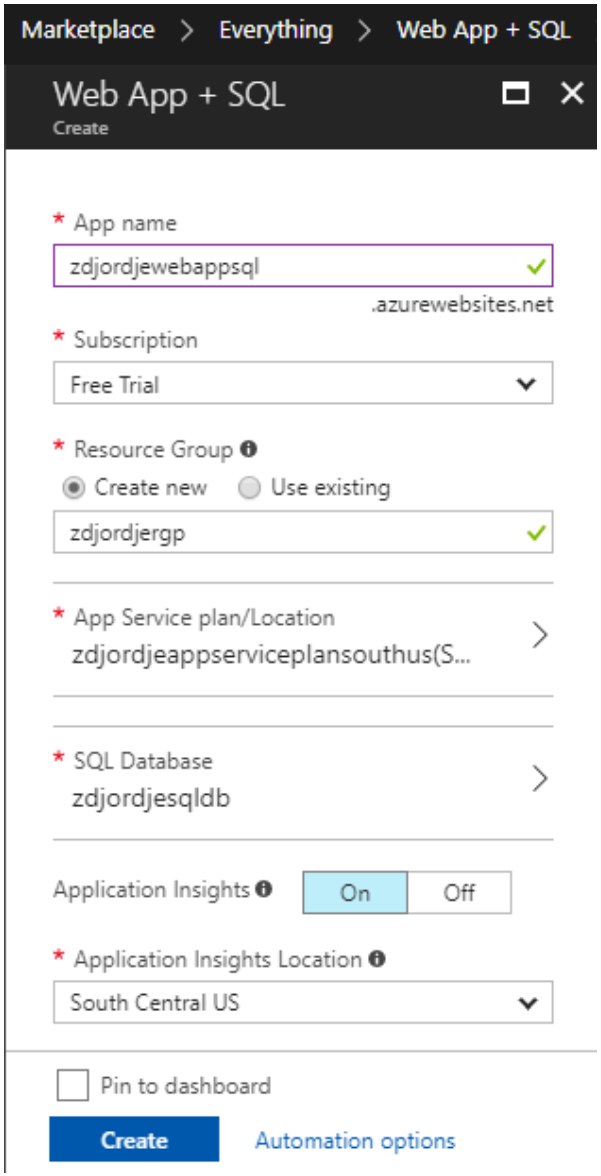
* Pricing tier
Standard S0: 10 DTU, 250 GB

* Collation
SQL_Latin1_General_CP1_CI_AS

Select

Automation Option

- On the last screen you can hit `Create` and Azure would create your Web App with SQL.
- Let us select "Automation options" instead.



Generated Template

Template

Download | Add to library | Deploy

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

Template | Parameters | CLI | PowerShell | .NET | Ruby

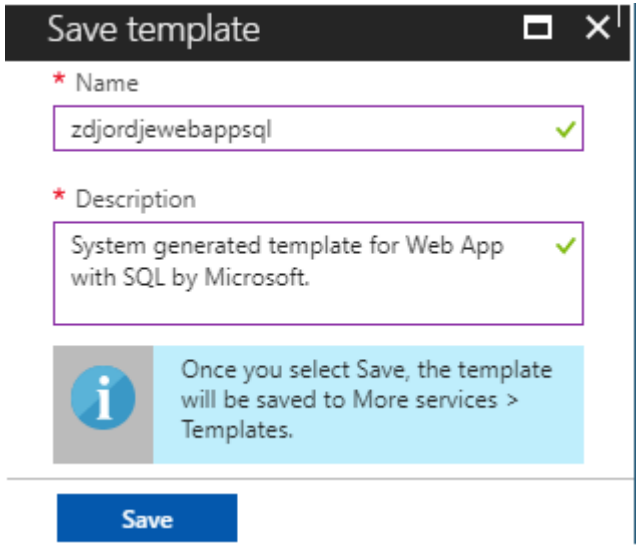
- Parameters (20)
- Variables (0)
- Resources (4)
  - [parameters('name')] (Microsoft.Web/sites)
  - [parameters('hostingPlanName')] (Microsoft.Web/serverfarms)
  - [parameters('name')] (microsoft.insights/components)
  - [parameters('serverName')] (Microsoft.Sql/servers)

```
    },
    "resources": [
        {
            "name": "[parameters('name')]",
            "type": "Microsoft.Web/sites",
            "properties": {
                "siteConfig": {
                    "connectionStrings": [
                        {
                            "name": "defaultConnection",
                            "ConnectionString": "[concat('Data Source=tcp:' , reference
('Microsoft.SQL/servers/zdjordjedbserver').fullyQualifiedDomainName , ',1433;Initial
Catalog=zdjordjesqldb;User Id=MyAdmin123$@',reference('Microsoft.SQL/servers/zdjordjedbserver')
.fullyQualifiedDomainName,';Password=',parameters('administratorLoginPassword'),';')]",
                            "type": "SQLAzure"
                        }
                    ],
                    "appSettings": [
                        {
                            "name": "APPINSIGHTS_INSTRUMENTATIONKEY",
                            "value": "[reference(resourceId('microsoft.insights/components/',
parameters('name')), '2015-05-01').InstrumentationKey]"
                        }
                    ]
                },
                "name": "[parameters('name')]",
                "serverFarmId": "[concat('/subscriptions/', parameters('subscriptionId'),
'/resourcegroups/', parameters('serverFarmResourceGroup'),
'/providers/Microsoft.Web/serverfarms/', parameters('hostingPlanName'))]",
                "hostingEnvironment": "[parameters('hostingEnvironment')]"
            },
            "dependsOn": [
```

- System generated a template which we could `Deploy`, `Add to library` or `Download`.
- If you click on any of 4 resources on the left, a section corresponding to that resource will appear on the right.

Save or Download Templates

- You might want to save your template in the library.
- You can find saved template on the left navigation pane under: `More services > Templates`
- If you hit `Download`, you will see `template.json`, `parameters.json` and several scripts and program.
- Script: `deploy.sh` contains CLI commands to read several command line inputs, pass them to the template and deploy your Web App.



Name	Date modified	Type	Size
deploy	11/7/2017 11:41 AM	Windows PowerS...	4 KB
deploy	11/7/2017 11:41 AM	Shell Script	3 KB
deployer.rb	11/7/2017 11:41 AM	RB File	4 KB
DeploymentHelper.cs	11/7/2017 11:41 AM	Visual C# Source f...	6 KB
parameters	11/7/2017 11:41 AM	JSON File	2 KB
template	11/7/2017 11:41 AM	JSON File	7 KB
template	11/7/2017 11:41 AM	Compressed (zipp...	7 KB

deploy.sh

- You can see: `deploy.sh, deploy.ps1, deployer.rb, DeploymentHelper.cs` on line in Azure portal as well. Select one of options next to Template.

Template

Download | Add to library | Deploy

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

Template | Parameters | CLI | PowerShell | .NET | Ruby

```
1  #!/bin/bash
2  set -euo pipefail
3  IFS=$'\n\t'
4
5  # -e: immediately exit if any command has a non-zero exit status
6  # -o: prevents errors in a pipeline from being masked
7  # IFS new value is less likely to cause confusing bugs when looping arrays or arguments (e.g. $@)
8
9  usage() { echo "Usage: $0 -i <subscriptionId> -g <resourceGroupName> -n <deploymentName> -l <resourceGroupLocation>" 1>&2; exit 1; }
10
11 declare subscriptionId=""
12 declare resourceGroupName=""
13 declare deploymentName=""
14 declare resourceGroupLocation=""
15
16 # Initialize parameters specified from command line
17 while getopts ":i:g:n:l:" arg; do
18     case "${arg}" in
19         i)
20             subscriptionId=${OPTARG}
21             ;;
22         g)
23             resourceGroupName=${OPTARG}
24             ;;
25         n)
26             deploymentName=${OPTARG}
27             ;;
28         l)
29             resourceGroupLocation=${OPTARG}
30             ;;
31         esac
32 done
```

To run this script, you need:

- `subscriptionId,`
- `resourceGroupName`
- `deploymentName`
- `resourceGroupLocation`

Azure Quick Start Templates

- https://github.com/Azure/azure-quickstart-templates holds a large collection of various templates you could use as a start for your own development.

GitHub, Inc. [US] github.com/Azure/azure-quickstart-templates/tree/master/101-1vm-2nics-2subnets-1vnet

Azure / azure-quickstart-templates

Watch 415 | Star 2,920

Code | Issues 326 | Pull requests 27 | Projects 0 | Wiki | Insights

Branch: master | azure-quickstart-templates / 101-1vm-2nics-2subnets-1vnet / | Create new file

bmoore-msft update from CI failures | Latest comm

..	
README.md	simplified overall template
azuredeploy.json	simplified overall template
azuredeploy.parameters.json	update from CI failures
metadata.json	simplified overall template

README.md

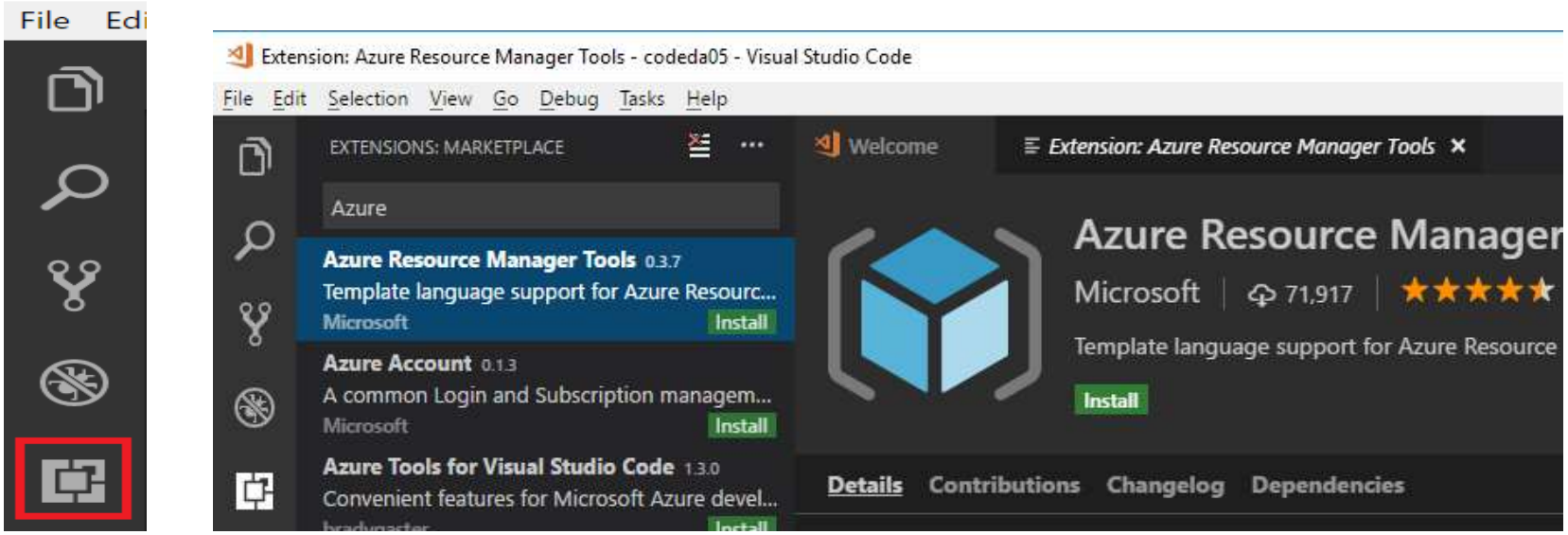
Multi-NIC Virtual Machine Creation using Two Subnets

Deploy to Azure | Visualize

This template creates a new VM with two NICs which connect to two different subnets within the same VNet.

Create or Revise RM Templates in Visual Studio Code

- To create and revise templates, you need a JSON editor.
- Visual Studio Code is a lightweight, open-source, cross-platform code editor.
- It appears that MS recommends using Visual Studio Code for creating or revising Resource Manager templates.
- Go to https://code.visualstudio.com/ and Download Visual Studio Code. You have downloads for Windows 64&32 bit, Linux 64&32 bit and mcOS.
- Run downloaded installer as Administrator. Run Visual Studio Code.
- Select Extensions, Search for Azure Resource Manager. Hit Install.
- You may install other Azure related tools.



Open Visual Studio Code

- Welcome Screen is useful. You can always get to it by going: Help > Welcome.
- Open the folder with your code. List of files will appear on the left. Select the template.json.

template.json - codeda05 - Visual Studio Code

File Edit Selection View Go Debug Tasks Help

EXPLORER

OPEN EDITORS

CODEDA05

- deploy.ps1
- deploy.sh
- deployer.rb
- DeploymentHelper.cs
- parameters.json
- template.json
- template.zip

Welcome | template.json

```
69                  "siteConfig": {
70                      "connectionStrings": [
71                          {
72                              "name": "defaultConnection",
73                              "ConnectionString": "[concat('Data Source=tcp:' , reference('Microsoft.
74                              "type": "SQLAzure"
75                          }
76                      ],
77                      "appSettings": [
78                          {
79                              "name": "APPINSIGHTS_INSTRUMENTATIONKEY",
80                              "value": "[reference(resourceId('microsoft.insights/components/', param
81                          }
82                      ]
83                  },
84                  "name": "[parameters('name')]",
85                  "serverFarmId": "[concat('/subscriptions/', parameters('subscriptionId'),'/resource
86                  "hostingEnvironment": "[parameters('hostingEnvironment')]"
87              },
88              "dependsOn": [
89                  "[concat('Microsoft.Web/serverfarms/', parameters('hostingPlanName'))]",
90                  "[resourceId('microsoft.insights/components/', parameters('name'))]"
91              ],
92              "apiVersion": "2016-03-01",
93              "location": "[parameters('location')]",
94              "tags": {
95                  "[concat('hidden-related:', '/subscriptions/', parameters('subscriptionId'),'/resou
96              }
97          },
98          {
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL Azure Tools

Editing Templates, Autocompletion

- However, when developing your own templates, you want to find and specify properties and values that are available for the resource type. VS Code reads the schema for the resource type, and suggests properties and values.
- To see the autocomplete feature, go the properties element of your template and add a new line. Type a quotation mark, and notice that VS Code immediately suggests names that available within the properties element.

```
    "properties": {
        "name": "[parameters('hostingPlanName')]",
        "workerSizeId": "[parameters('workerSize')]",
        "numberOfWorkers": "1",
        "hostingEnvironment": "[parameters('hostingEnvironment')]"
    ""
    } serverFarmId
    " siteConfig
        "Tier": "[parameters('sku')]",
        "Name": "[parameters('skuCode')]"
    }
```

Add Tags to Your Resources

- We have just learned how to create an Azure resource group, and how to add an Azure resource. We still need a way to organize our resources logically, for example, for the calculation of cost or for a targeted tracking.
- The Azure resource manager offers a solution for this, Azure resource tags.
- Azure resource tags are any key/value pairs that appear useful to describe a resource. They also allow you to search for resources based on a tag.
- Tags are the first level elements below “resources” and you can add tags as key-value pairs to any resource. For example:

```
"resources": [
    {
        "name": "[variables('storageName')]",
        "type": "Microsoft.Storage/storageAccounts",
        "apiVersion": "2016-01-01",
        "sku": {
            "name": "[parameters('storageSKU')]"
        },
        "kind": "Storage",
        "location": "[resourceGroup().location]",
        "tags": {
            "Dept":"IT",
            "Plant":"Houston"
        },
        "properties": {}
    }
],
"outputs": {}
```

Objects as Tag Element

- You can define an object parameter that stores several tags, and apply that object to the tag element.
- Each property in the object becomes a separate tag for the resource.
- The example on the right has a parameter named `tagValues` that is applied to the tag element.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/de
  "contentVersion": "1.0.0.0",
  "parameters": {
    "tagValues": {
      "type": "object",
      "defaultValue": {
        "Dept": "Finance",
        "Environment": "Production"
      }
    }
  },
  "resources": [
    {
      "apiVersion": "2016-01-01",
      "type": "Microsoft.Storage/storageAccounts",
      "name": "[concat('storage', uniqueString(resourceGroup().id))]",
      "location": "[resourceGroup().location]",
      "tags": "[parameters('tagValues')]",
      "sku": {
        "name": "Standard_LRS"
      },
      "kind": "Storage",
      "properties": {}
    }
  ]
}
```

JSON string as a Tag Name

- To store many values in a single tag, apply a JSON string that represents the values. The entire JSON string is stored as one tag that cannot exceed 256 characters.
- The example on the right has a single tag named `CostCenter` that contains several values from a JSON string:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploy
    "contentVersion": "1.0.0.0",
    "resources": [
    {
      "apiVersion": "2016-01-01",
      "type": "Microsoft.Storage/storageAccounts",
      "name": "[concat('storage', uniqueString(resourceGroup().id))]",
      "location": "[resourceGroup().location]",
      "tags": {
        "CostCenter": "{\"Dept\":\"Finance\",\"Environment\":\"Production\"}"
      },
      "sku": {
        "name": "Standard_LRS"
      },
      "kind": "Storage",
      "properties": { }
    }
    ]
}
```

Syntax of RM Templates

Template Format

- In its simplest structure, a template contains the following elements:

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

Element name	Required	Description
$schema	Yes	Location of the JSON schema file that describes the version of the template language.
contentVersion	Yes	Version of the template (such as 1.0.0.0). This value can be used to make sure that the right template is being used.
parameters	No	Values that are provided when deployment is executed to customize resource deployment.
variables	No	Values that are used as JSON fragments in the template to simplify template language expressions.
resources	Yes	Resource types that are deployed or updated in a resource group.
outputs	No	Values that are returned after deployment.

Properties of Elements, parameters, variables

- Each element contains properties you can set. The following example contains the full syntax for a template.

```
"contentVersion": "",
"parameters": {
    "<parameter-name>" : {
        "type" : "<type-of-parameter-value>",
        "defaultValue": "<default-value-of-parameter>",
        "allowedValues": [ "<array-of-allowed-values>" ],
        "minValue": <minimum-value-for-int>,
        "maxValue": <maximum-value-for-int>,
        "minLength": <minimum-length-for-string-or-array>,
        "maxLength": <maximum-length-for-string-or-array-parameters>,
        "metadata": {
            "description": "<description-of-the parameter>"
        }
    }
},
"variables": {
    "<variable-name>": "<variable-value>",
    "<variable-name>": {
        <variable-complex-type-value>
    }
},
```

Properties of `resources, outputs`

```
"resources": [
      {
          "condition": "<boolean-value-whether-to-deploy>",
          "apiVersion": "<api-version-of-resource>",
          "type": "<resource-provider-namespace/resource-type-name>",
          "name": "<name-of-the-resource>",
          "location": "<location-of-resource>",
          "tags": {
              "<tag-name1>": "<tag-value1>",
              "<tag-name2>": "<tag-value2>"
          },
          "comments": "<your-reference-notes>",
          "copy": {
              "name": "<name-of-copy-loop>",
              "count": "<number-of-iterations>",
              "mode": "<serial-or-parallel>",
              "batchSize": "<number-to-deploy-serially>"
          },
          "dependsOn": [
              "<array-of-related-resource-names>"
          ],
          "properties": {
              "<settings-for-the-resource>",
              "copy": [
                  {
                      "name": ,
                      "count": ,
                      "input": {}
                  }
              ]
          },
          "resources": [
              "<array-of-child-resources>"
          ]
      }
    ],
```

```
"outputs": {
        "<outputName>" : {
            "type" : "<type-of-output-value>",
            "value": "<output-value-expression>"
        }
    }
```

Expressions and Functions

- However, expressions and functions extend the JSON values available within the template. Expressions are written within JSON string literals whose first and last characters are the brackets: [and], respectively. The value of the expression is evaluated when the template is deployed. While written as a string literal, the result of evaluating the expression can be of a different JSON type, such as an array or integer, depending on the actual expression.
- Typically, you use expressions with functions to perform operations for configuring the deployment. Just like in JavaScript, function calls are formatted as `functionName(arg1,arg2,arg3).`
- You reference properties by using the dot and [index] operators. For example:

```
"variables": {
    "location": "[resourceGroup().location]",
    "usernameAndPassword": "[concat(parameters('username'), ':',
parameters('password'))]",
    "authorizationHeader": "[concat('Basic ',
base64(variables('usernameAndPassword')))]"
}
```

Parameters

- In the parameters section of the template, you specify which values you can input when deploying the resources. These parameter values enable you to customize the deployment by providing values that are tailored for a particular environment (such as dev, test, and production). You do not have to provide parameters in your template, but without parameters your template would always deploy the same resources with the same names, locations, and properties.
- You define parameters with the following structure:

```
"parameters": {
    "<parameter-name>" : {
        "type" : "<type-of-parameter-value>",
        "defaultValue": "<default-value-of-parameter>",
        "allowedValues": [ "<array-of-allowed-values>" ],
        "minValue": <minimum-value-for-int>,
        "maxValue": <maximum-value-for-int>,
        "minLength": <minimum-length-for-string-or-array>,
        "maxLength": <maximum-length-for-string-or-array-parameters>,
        "metadata": {
            "description": "<description-of-the parameter>"
        }
    }
}
```

`parameter`, elements

Element name	Required	Description
parameterName	Yes	Name of the parameter. Must be a valid JavaScript identifier.
type	Yes	Type of the parameter value. See the list of allowed types bellow.
defaultValue	No	Default value for the parameter, if no value is provided for the parameter.
allowedValues	No	Array of allowed values for the parameter to make sure that the right value is provided.
minValue	No	The minimum value for int type parameters, this value is inclusive.
maxValue	No	The maximum value for int type parameters, this value is inclusive.
minLength	No	The minimum length for string, secureString, and array type parameters, this value is inclusive.
maxLength	No	The maximum length for string, secureString, and array type parameters, this value is inclusive.
description	No	Description of the parameter that is displayed to users through the portal.

The allowed types: string, secureString, int, bool, object, secureObject, array

Variables

- In the variables section, you construct values that can be used throughout your template. You do not need to define variables, but they often simplify templates You define variables with the following structure:

```
"variables": {
    "<variable-name>": "<variable-value>",
    "<variable-name>": {
        <variable-complex-type-value>
    }
}
```

- Next variable is constructed from two parameter values. The following from other variables:

```
"variables": {
    "connectionString": "[concat('Name=', parameters('username'), ';Password=',
parameters('password'))]"
}
```

```
"variables": {
    "environmentSettings": {
        "test": {
            "instancesSize": "Small",
            "instancesCount": 1
        },
        "prod": {
            "instancesSize": "Large",
            "instancesCount": 4
        }
    },
    "currentEnvironmentSettings":
"[variables('environmentSettings')[parameters('environmentName')]]",
```

resouces

- In the resources section, you define the resources that are deployed or updated. This section can get complicated because you must understand the types you are deploying to provide the right values. Resources are defined as following structure

```
"resources": [
  {
      "condition": "<boolean-value-whether-to-deploy>",
      "apiVersion": "<api-version-of-resource>",
      "type": "<resource-provider-namespace/resource-type-name>",
      "name": "<name-of-the-resource>",
      "location": "<location-of-resource>",
      "tags": {
          "<tag-name1>": "<tag-value1>",
          "<tag-name2>": "<tag-value2>"
      },
      "comments": "<your-reference-notes>",
      "copy": {
          "name": "<name-of-copy-loop>",
          "count": "<number-of-iterations>",
          "mode": "<serial-or-parallel>",
          "batchSize": "<number-to-deploy-serially>"
      },
      "dependsOn": [
          "<array-of-related-resource-names>"
      ],
      "properties": {
          "<settings-for-the-resource>",
          "copy": [
              {
                  "name": ,
                  "count": ,
                  "input": {}
              }
          ]
      },
      "resources": [
          "<array-of-child-resources>"
      ]
  }
]
```

`resources`, elements

Element name	Required	Description
condition	No	Boolean value that indicates whether the resource is deployed.
apiVersion	Yes	Version of the REST API to use for creating the resource.
type	Yes	Type of the resource. This value is a combination of the namespace of the resource provider and the resource type (such as **Microsoft.Storage/storageAccounts**).
name	Yes	Name of the resource. The name must follow URI component restrictions defined in RFC3986. In addition, Azure services that expose the resource name to outside parties validate the name to make sure it is not an attempt to spoof another identity.
location	Varies	Supported geo-locations of the provided resource. You can select any of the available locations, but typically it makes sense to pick one that is close to your users. Usually, it also makes sense to place resources that interact with each other in the same region. Most resource types require a location, but some types (such as a role assignment) do not require a location. See Set resource location in Azure Resource Manager templates.
tags	No	Tags that are associated with the resource. See Tag resources in Azure Resource Manager templates.
comments	No	Your notes for documenting the resources in your template

`resources`, elements

copy	No	If more than one instance is needed, the number of resources to create. The default mode is parallel. Specify serial mode when you do not want all or the resources to deploy at the same time. For more information, see Create multiple instances of resources in Azure Resource Manager.
dependsOn	No	Resources that must be deployed before this resource is deployed. Resource Manager evaluates the dependencies between resources and deploys them in the correct order. When resources are not dependent on each other, they are deployed in parallel. The value can be a comma-separated list of a resource names or resource unique identifiers. Only list resources that are deployed in this template. Resources that are not defined in this template must already exist. Avoid adding unnecessary dependencies as they can slow your deployment and create circular dependencies. For guidance on setting dependencies, see Defining dependencies in Azure Resource Manager templates.
properties	No	Resource-specific configuration settings. The values for the properties are the same as the values you provide in the request body for the REST API operation (PUT method) to create the resource. You can also specify a copy array to create multiple instances of a property. For more information, see Create multiple instances of resources in Azure Resource Manager.
resources	No	Child resources that depend on the resource being defined. Only provide resource types that are permitted by the schema of the parent resource. The fully qualified type of the child resource includes the parent resource type, such as **Microsoft.Web/sites/extensions**. Dependency on the parent resource is not implied. You must explicitly define that dependency.

Child Resources

- The resources section contains an array of the resources to deploy. Within each resource, you can also define an array of child resources. Therefore, your resources section could have a structure like:

```
"resources": [
  {
      "name": "resourceA",
  },
  {
      "name": "resourceB",
      "resources": [
        {
            "name": "firstChildResourceB",
        },
        {
            "name": "secondChildResourceB",
        }
      ]
  },
  {
      "name": "resourceC",
  }
]
```

outputs

- In the Outputs section, you specify values that are returned from deployment. For example, you could return the URI to access a deployed resource.
- The following example shows the structure of an output definition:

```
"outputs": {
    "<outputName>" : {
        "type" : "<type-of-output-value>",
        "value": "<output-value-expression>"
    }
}
```

- This example shows a value that is returned in the Outputs section

```
"outputs": {
    "siteUri" : {
        "type" : "string",
        "value":
"[concat('http://',reference(resourceId('Microsoft.Web/sites',
parameters('siteName'))).hostNames[0])]"
    }
}
```

Template Limits

- Limit the size of your template to 1 MB, and each parameter file to 64 KB. The 1-MB limit applies to the final state of the template after it has been expanded with iterative resource definitions, and values for variables and parameters.
- You are also limited to:
  - 256 parameters
  - 256 variables
  - 800 resources (including copy count)
  - 64 output values
  - 24,576 characters in a template expression
- You can exceed some template limits by using a nested template. For more information, see Using linked templates when deploying Azure resources.
- To reduce the number of parameters, variables, or outputs, you can combine several values into an object.

Linked Templates

- From within one Azure Resource Manager template, you can link to another template, which enables you to decompose your deployment into a set of targeted, purpose-specific templates.
- As with decomposing an application into several code classes, decomposition provides benefits in terms of testing, reuse, and readability.
- You can pass parameters from a main template to a linked template, and those parameters can directly map to parameters or variables exposed by the calling template.
- The linked template can also pass an output variable back to the source template, enabling a two-way data exchange between templates.

Linking to a Template

- You create a link between two templates by adding a deployment resource within the main template that points to the linked template.
- You set the **templateLink** property to the URI of the linked template. You can provide parameter values for the linked template directly in your template or in a parameter file. The following example uses the **parameters** property to specify a parameter value directly.

```
"resources": [
  {
      "apiVersion": "2017-05-10",
      "name": "linkedTemplate",
      "type": "Microsoft.Resources/deployments",
      "properties": {
        "mode": "incremental",
        "templateLink": {
          "uri":
"https://www.contoso.com/AzureTemplates/newStorageAccount.json",
          "contentVersion": "1.0.0.0"
        },
        "parameters": {
          "StorageAccountName":{"value":
"[parameters('StorageAccountName')]"}
        }
      }
  }
]
```

Dependencies and passing of values

- Like other resource types, you can set dependencies between the linked template and other resources.
- When other resources require an output value from the linked template, you can make sure the linked template is deployed before them.
- Or, when the linked template relies on other resources, you can make sure other resources are deployed before the linked template.
- You can retrieve a value from a linked template with the following syntax:

```
"[reference('linkedTemplate').outputs.exampleProperty.value]"
```

Linked Templates Must be Accessible

- The Resource Manager service must be able to access the linked template. You cannot specify a local file or a file that is only available on your local network for the linked template.
- You can only provide a URI value that includes either http or https.
- One option is to place your linked template in a storage account, and use the URI for that item, such as shown in the following example:

```
"templateLink": {
    "uri":
"http://mystorageaccount.blob.core.windows.net/templates/template.json",
    "contentVersion": "1.0.0.0",
}
```

- Although the linked template must be externally available, it does not need to be generally available to the public. You can add your template to a private storage account that is accessible to only the storage account owner. Then, you create a shared access signature (SAS) token to enable access during deployment.

Linking to a parameter file

- The next example uses the `parametersLink` property to link to a parameter file.

```
"resources": [
  {
     "apiVersion": "2017-05-10",
     "name": "linkedTemplate",
     "type": "Microsoft.Resources/deployments",
     "properties": {
       "mode": "incremental",
       "templateLink": {

"uri":"https://www.contoso.com/AzureTemplates/newStorageAccount.json",
          "contentVersion":"1.0.0.0"
       },
       "parametersLink": {
          "uri":"https://www.contoso.com/AzureTemplates/parameters.json",
          "contentVersion":"1.0.0.0"
       }
     }
  }
]
```

- The URI value for the linked parameter file cannot be a local file, and must include either http or https.

Using variables to link templates

- The previous examples showed hard-coded URL values for the template links. This approach might work for a simple template but it does not work well when working with a large set of modular templates.
- Instead, you can create a static variable that stores a base URL for the main template and then dynamically create URLs for the linked templates from that base URL.
- The benefit of this approach is you can easily move or fork the template because you only need to change the static variable in the main template. The main template passes the correct URIs throughout the decomposed template.
- The following example shows how to use a base URL to create two URLs for linked templates (sharedTemplateUrl and vmTemplate).

```
"variables": {
    "templateBaseUrl": "https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/postgresql-on-ubuntu/",
    "sharedTemplateUrl": "[concat(variables('templateBaseUrl'), 'shared-resources.json')]",
    "vmTemplateUrl": "[concat(variables('templateBaseUrl'), 'database-2disk-resources.json')]"
}
```

Function `deployment()`

- You can also use `deployment()` to get the base URL for the current template, and use that to get the URL for other templates in the same location.
- This approach is useful if your template location changes (maybe due to versioning) or you want to avoid hard coding URLs in the template file.

```
"variables": {
    "sharedTemplateUrl": "[uri(deployment().properties.templateLink.uri,
'shared-resources.json')]"
}
```

Example of Linked Templates, `helloworld.json`

- The following example templates show a simplified arrangement of linked templates to illustrate linking of templates.
- Example assumes the templates have been added to the same container in a storage account with public access turned off.
- The linked template passes a value back to the main template in the **outputs** section.

`helloworld.json` (template)

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

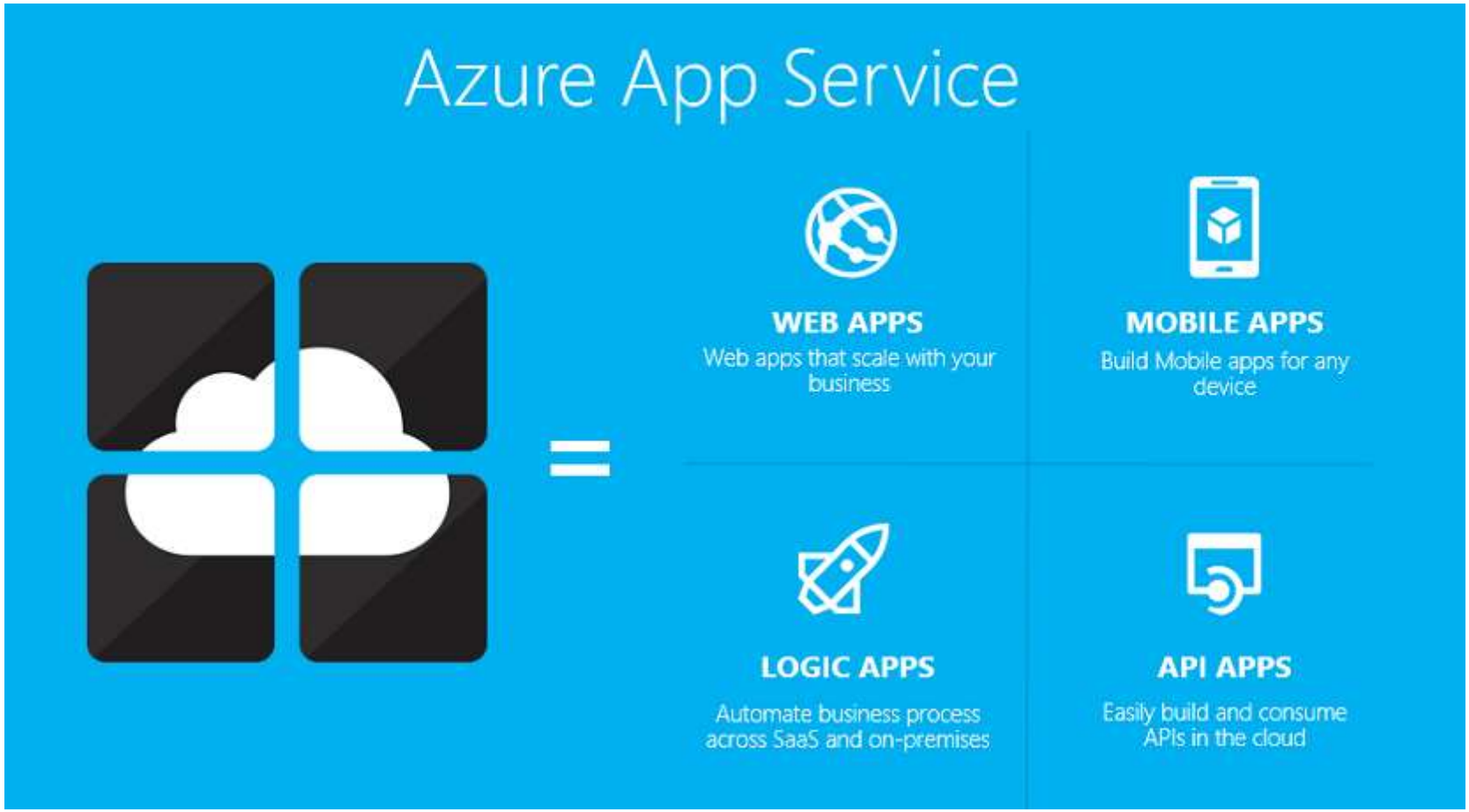
Linked Templates, `parent.json`

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "containerSasToken": { "type": "string" }
  },
  "resources": [
    {
      "apiVersion": "2017-05-10",
      "name": "linkedTemplate",
      "type": "Microsoft.Resources/deployments",
      "properties": {
        "mode": "incremental",
        "templateLink": {
          "uri": "[concat(uri(deployment().properties.templateLink.uri, 'helloworld.json'),
parameters('containerSasToken'))]",
          "contentVersion": "1.0.0.0"
        }
      }
    }
  ],
  "outputs": {
    "result": {
      "type": "string",
      "value": "[reference('linkedTemplate').outputs.result.value]"
    }
  }
}
```

Get the token using Azure CLI

```
expiretime=$(date -u -d '30 minutes' +%Y-%m-%dT%H:%MZ)
connection=$(az storage account show-connection-string \
    --resource-group ManageGroup \
    --name storagecontosotemplates \
    --query connectionString)
token=$(az storage container generate-sas \
    --name templates \
    --expiry $expiretime \
    --permissions r \
    --output tsv \
    --connection-string $connection)
url=$(az storage blob url \
    --container-name templates \
    --name parent.json \
    --output tsv \
    --connection-string $connection)
parameter='{"containerSasToken":{"value":"?'$token'"}}'
```

Azure App Service(s)



Azure App Service

- Azure App Service includes the Web App and Mobile App capabilities.
- Azure App Service also includes powerful Logic/Workflow App and API App capabilities with built-in connectors that make it easy to build logic workflows that integrate with dozens of popular SaaS and on-premises applications (Office 365, SalesForce, Dynamics, OneDrive, Box, DropBox, Twilio, Twitter, Facebook, Marketo, and more).
- *Azure App Service* is a service for hosting web applications, REST APIs, and mobile back ends.
- In Ap Service, we can develop in our favorite language, .NET, .NET Core, Java, Ruby, Node.js, PHP, or Python. We can run and scale apps on Windows or Linux VMs.
- Azure App Service adds the power of Microsoft Azure to our applications, such as security, load balancing, autoscaling, and automated management.
- With App Service we can take advantage of Azure DevOps capabilities, such as continuous deployment from VSTS, GitHub, Docker Hub, and other sources, package management, staging environments, custom domain, and SSL certificates.
- With App Service, we pay for the Azure compute resources we use. The compute resources we use are determined by the *App Service plan* we run.

App Service Plans

- App Service plans represent the collection of physical resources used to host your apps.
- App Service plans define:
  - Region (West US, East US, etc.)
  - Scale count (one, two, three instances, etc.)
  - Instance size (Small, Medium, Large)
  - SKU (stock Free, Shared, Basic, Standard, Premium, PremiumV2, Isolated)
- Web Apps, Mobile Apps, API Apps, Function Apps (or Functions), in Azure App Service all run in an App Service plan. Apps in the same subscription, and region can share an App Service plan.
- All applications assigned to an **App Service plan** share the resources defined by it. This sharing saves money when hosting multiple apps in a single App Service plan.
- Your **App Service plan** can scale from **Free** and **Shared** tiers to **Basic**, **Standard**, **Premium**, and **Isolated** tiers. Each higher tier gives you access to more resources and features.
- If your App Service plan is set to **Basic** tier or higher, then you can control the **size** and scale count of the VMs.
- For example, if your plan is configured to use two "small" instances in the **Standard** tier, all apps in that plan run on both instances. Apps also have access to the **Standard** tier features. Plan instances on which apps are running are fully managed and highly available.
- The pricing tier (SKU, stock keeping unit) of the App Service plan determines the cost and not the number of apps hosted in it.

App Service Features

- **Autoscaling**: With App Service, you can quickly scale up or scale out to handle any incoming customer load. Manually select the number and size of VMs, or set up autoscaling to scale your mobile-app back end based on load or schedule.
- **Staging environments**: App Service can run multiple versions of your site, so you can perform A/B testing, test in production as part of a larger DevOps plan, and do in-place staging of a new back end.
- **Continuous deployment**: App Service can integrate with common supply chain management (SCM) systems, so you can automatically deploy a new version of your back end by pushing to a branch of your SCM system.
- **Virtual networking**: App Service can connect to on-premises resources by using virtual network, Azure ExpressRoute, or hybrid connections.
- **Isolated and dedicated environments**: You can run App Service in a fully isolated and dedicated environment for securely running Azure App Service apps at high scale. This environment is ideal for application workloads that require high scale, isolation, or secure network access.

App Service, Web Apps

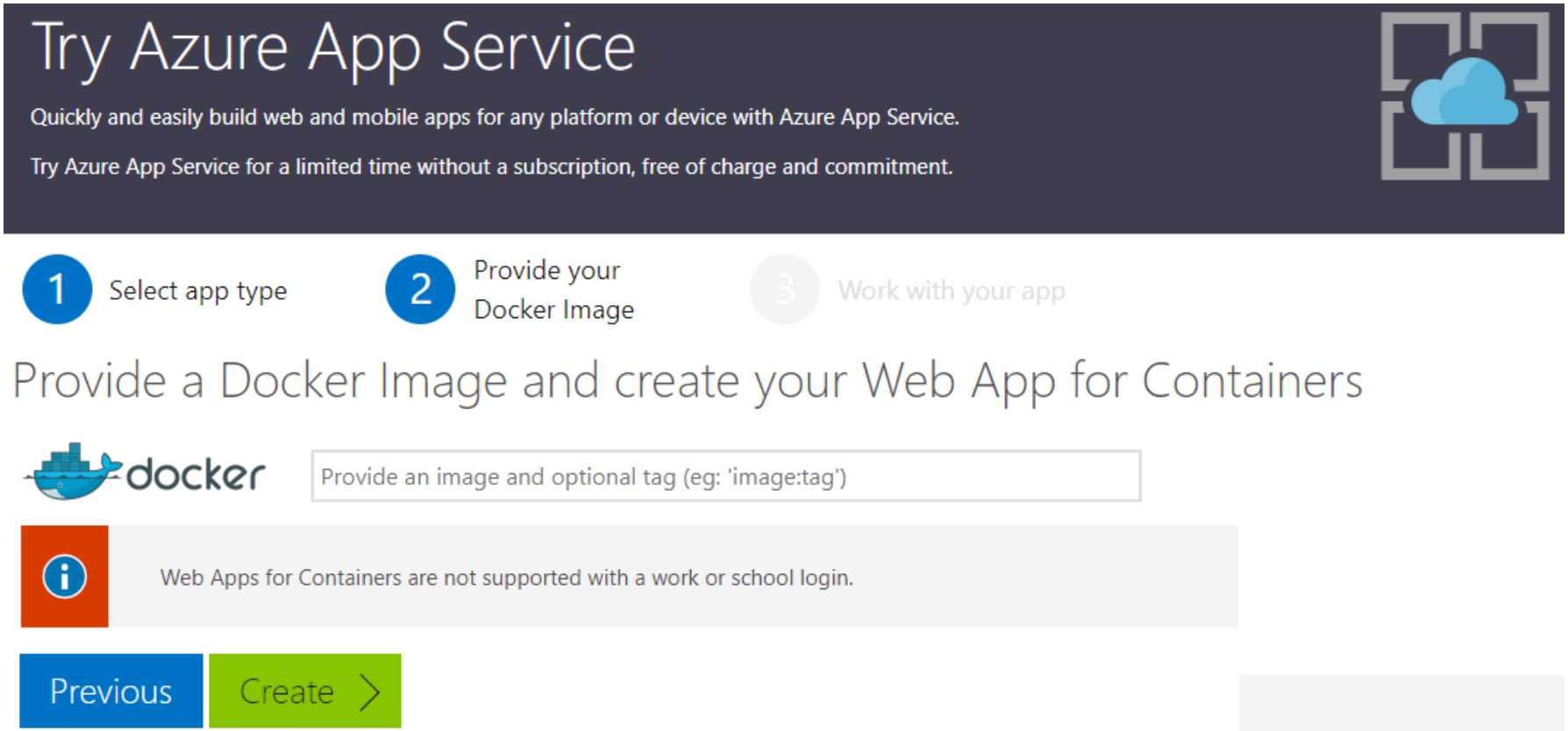
- **Multiple languages and frameworks** - Web Apps has first-class support for ASP.NET, ASP.NET Core, Java, Ruby, Node.js, PHP, or Python. You can also run PowerShell and other scripts or executables as background services.
- **DevOps optimization** - Set up continuous integration and deployment with Visual Studio Team Services, GitHub, BitBucket, Docker Hub, or Azure Container Service. Promote updates through test and staging environments. Manage your apps in Web Apps by using Azure PowerShell or the cross-platform command-line interface (CLI).
- **Global scale with high availability** - Scale up or out manually or automatically. Host your apps anywhere in Microsoft's global datacenter infrastructure, and the App Service SLA promises high availability.
- **Connections to SaaS platforms and on-premises data** - Choose from more than 50 connectors for enterprise systems (such as SAP), SaaS services (such as Salesforce), and internet services (such as Facebook). Access on-premises data using Hybrid Connections and Azure Virtual Networks.

App Service, Web Apps

- **Security and compliance** - App Service is ISO, SOC, and PCI compliant. Authenticate users with Azure Active Directory or with social login (Google, Facebook, Twitter, and Microsoft). Create IP address restrictions and manage service identities.
- **Application templates** - Choose from an extensive list of application templates in the Azure Marketplace, such as WordPress, Joomla, and Drupal.
- **Visual Studio integration** - Dedicated tools in Visual Studio streamline the work of creating, deploying, and debugging.
- **API and mobile features** - Web Apps provides turn-key CORS support for RESTful API scenarios, and simplifies mobile app scenarios by enabling authentication, offline data sync, push notifications, and more.
- **Serverless code** - Run a code snippet or script on-demand without having to explicitly provision or manage infrastructure, and pay only for the compute time your code actually uses (see Azure Functions).

App Service, Web App for Containers

- Provide a Docker Image and create your Web App for Containers

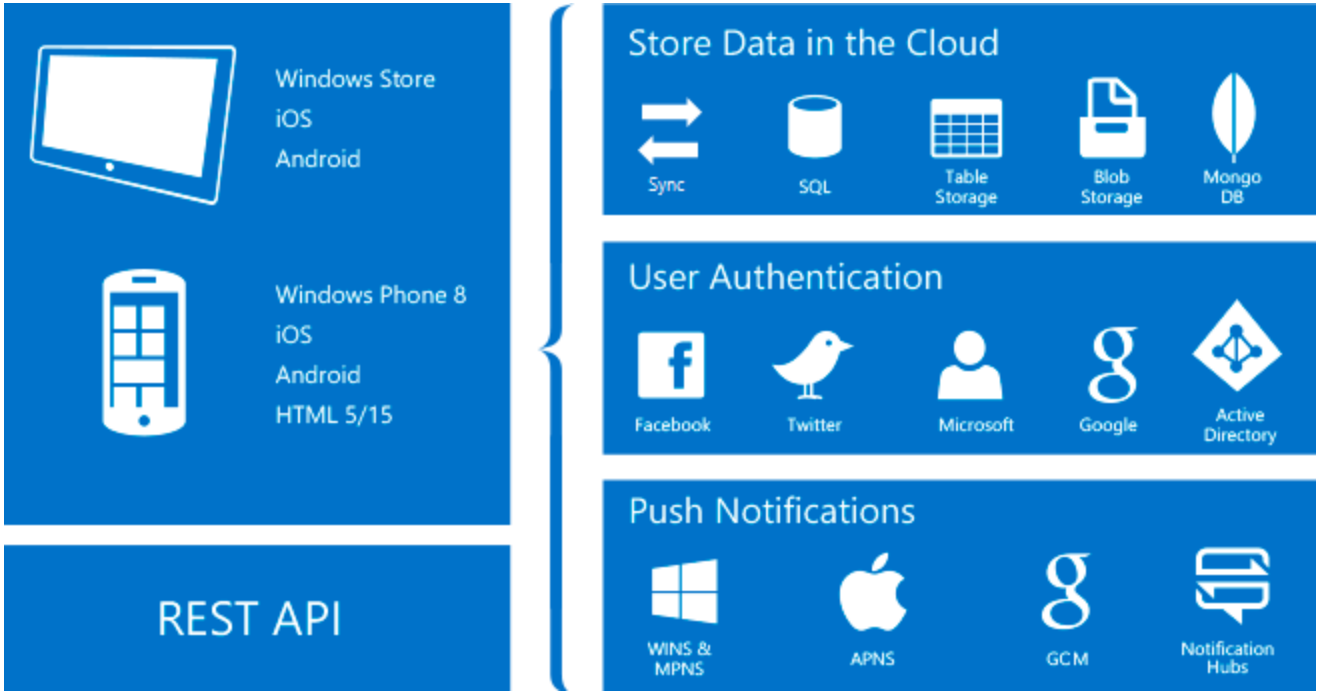


App Service, Azure for Containers

- Azure Container Service (AKS) makes it simple to create, configure, and manage a cluster of virtual machines that are preconfigured to run containerized applications. This enables you to use your existing skills, or draw upon a large and growing body of community expertise, to deploy and manage container-based applications on Microsoft Azure.
- By using AKS, you can take advantage of the enterprise-grade features of Azure, while still maintaining application portability through Kubernetes and the Docker image format.
- Using Azure Container Service (AKS)
- The goal of AKS is to provide a container hosting environment by using open-source tools and technologies that are popular among our customers today. To this end, we expose the standard Kubernetes API endpoints. By using these standard endpoints, you can leverage any software that is capable of talking to a Kubernetes cluster.

App Service, Mobile Apps

- Azure App Service is a fully managed platform as a service (PaaS) offering for professional developers. The service brings a rich set of capabilities to web, mobile, and integration scenarios.+
- The Mobile Apps feature of Azure App Service gives enterprise developers and system integrators a mobile-application development platform that's highly scalable and globally available.



App Service vs. Mobile Apps

- In *Azure App Service* the *Mobile App* backend code runs in the same container as Web App and API App. As such you can take advantage of all the features in this container, including some of those that are not currently present in Mobile Services
- Add continuously running backend logic via Web Jobs
- Ensure your backend code is always running
- Use custom CNames to provide friendly and stable names to your mobile backend endpoints
- Geo-scale your app with Traffic Manager
- Include any libraries and packages you want.
- (For .NET) Leverage any feature of ASP.NET, including MVC
- (For Node.js) Leverage any pure JavaScript library of the Node ecosystem, including common MVC libraries.

Why Mobile Apps

With the Mobile Apps, you can:

- **Build native and cross-platform apps**: Whether you're building native iOS, Android, and Windows apps or cross-platform Xamarin or Cordova (PhoneGap) apps, you can take advantage of App Service by using native SDKs.
- **Connect to your enterprise systems**: With the Mobile Apps feature, you can add corporate sign-in in minutes, and connect to your enterprise on-premises or cloud resources.
- **Build offline-ready apps with data sync**: Make your mobile workforce more productive by building apps that work offline, and use Mobile Apps to sync data in the background when connectivity is present with any of your enterprise data sources or software as a service (SaaS) APIs.
- **Push notifications to millions in seconds**: Engage your customers with instant push notifications on any device, personalized to their needs and sent when the time is right.

Mobile Apps Features

- **Authentication and authorization**: Select from an ever-growing list of identity providers, including Azure Active Directory for enterprise authentication, plus social providers such as Facebook, Google, Twitter, and Microsoft accounts. Mobile Apps offers an OAuth 2.0 service for each provider. You can also integrate the SDK for the identity provider for provider-specific functionality.
- **Data access**: Mobile Apps provides a mobile-friendly OData v3 data source that's linked to Azure SQL Database or an on-premises SQL server. Because this service can be based on Entity Framework, you can easily integrate with other NoSQL and SQL data providers, including Azure Table storage, MongoDB, Azure Cosmos DB, and SaaS API providers such as Office 365 and Salesforce.com.
- **Offline sync**: Our client SDKs make it easy to build robust and responsive mobile applications that operate with an offline dataset. You can sync this dataset automatically with the back-end data, including conflict-resolution support.
- **Push notifications**: Our client SDKs integrate seamlessly with the registration capabilities of Azure Notification Hubs, so you can send push notifications to millions of users simultaneously.
- **Client SDKs**: We provide a complete set of client SDKs that cover native development (iOS, Android, and Windows), cross-platform development (Xamarin.iOS and Xamarin.Android, Xamarin.Forms), and hybrid application development (Apache Cordova). Each client SDK is available with an MIT license and is open source.

API Apps

- Microsoft refuses to tell us what is API App service. The following is marketing:
- API App allows you to build and consume APIs in the cloud using the language of your choice
- API App provides security support for Azure Active Directory, single sign-on, and OAuth
- API Apps makes it possible for you to bring in existing APIs written with .NET, PHP, Node.js, Java, or Python
- API Apps makes it possible for you to consume APIs on any website with CORS support.

Continuous integration and deployment

- Use the API Apps feature of Microsoft Azure App Service to connect your favorite version control system to your API app, and automatically deploy commits, which makes code changes easier than ever. Move your API to production, run tests against a copy of your app provided by deployment slots, and then redirect traffic to the new version without downtime.

Simple authentication

- Your API is just a few clicks away from being highly-secured through Azure Active Directory, social network single sign-on, or OAuth. No code changes are required, and we keep the sign-on SDKs for your services up to date.

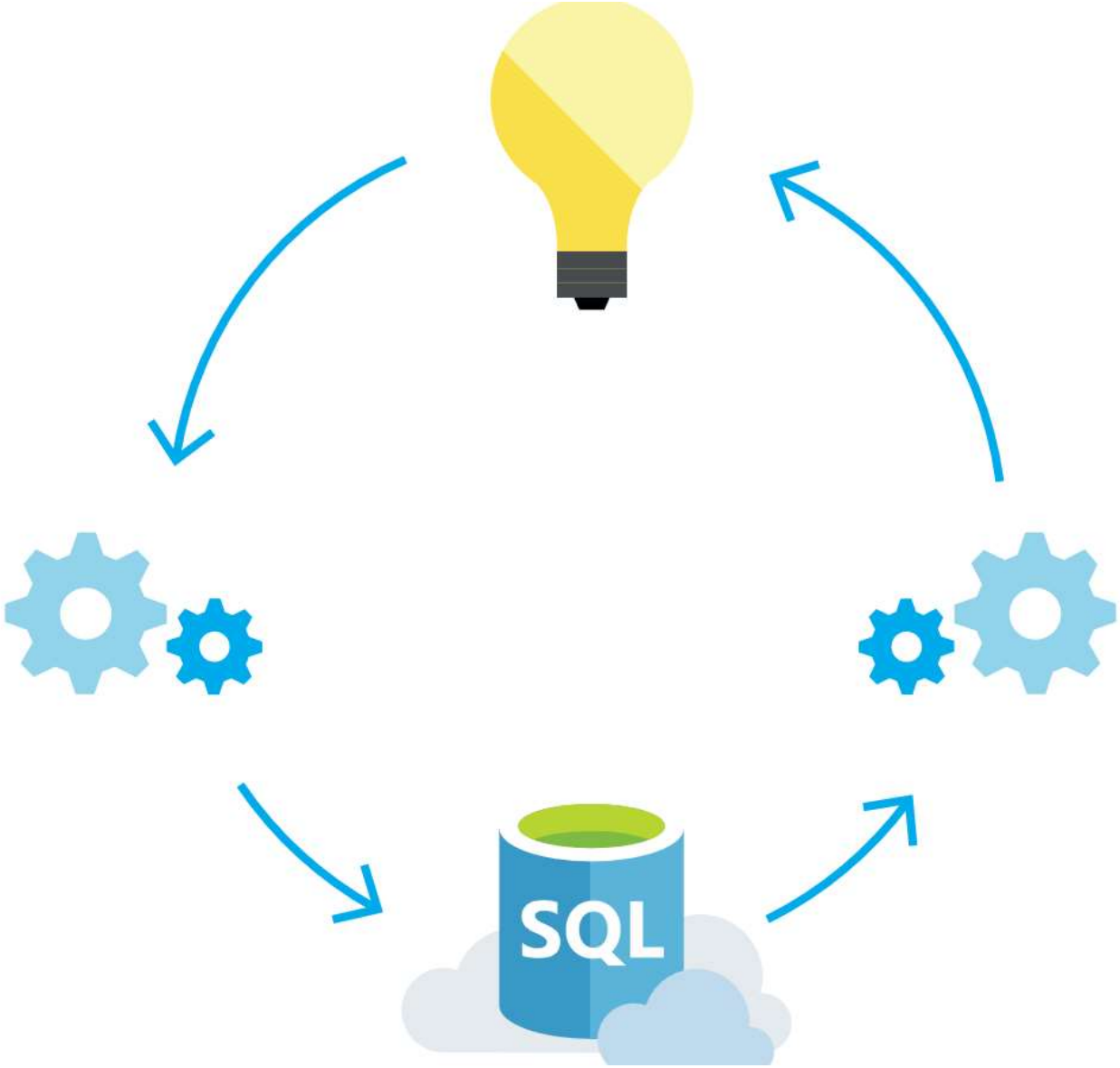
On-premises connectivity

- Connect your API app to your corporate or local network using on-premises connections with enterprise-grade security. Serve APIs to your intranet as if they were running locally, or connect them to existing internal network resources.

CORS

- Cross-origin resource sharing (CORS) is a World Wide Web Consortium (W3C) specification (commonly considered part of HTML5) that lets JavaScript overcome the same-origin policy security restriction imposed by browsers. The same-origin policy means that your JavaScript can only make AJAX calls back to the same origin of the containing Web page (where “origin” is defined as the combination of host name, protocol and port number). For example, JavaScript on a Web page from http://foo.com can’t make AJAX calls to http://bar.com (or to http://www.foo.com, https://foo.com or http://foo.com:999, for that matter).
- CORS relaxes this restriction by letting servers indicate which origins are allowed to call them. CORS is enforced by browsers but must be implemented on the server, and the most recent release of ASP.NET Web API 2 has full CORS support. With Web API 2, you can configure policy to allow JavaScript clients from a different origin to access your APIs.
- The general mechanics of CORS are such that when JavaScript is attempting to make a cross-origin AJAX call the browser will “ask” the server if this is allowed by sending headers in the HTTP request (for example, Origin). The server indicates what’s allowed by returning HTTP headers in the response (for example, Access-Control-Allow-Origin). This permission check is done for each distinct URL the client invokes, which means different URLs can have different permissions.

Azure SQL Database



Azure SQL Database

- Azure SQL Database is a relational database-as-a service using the Microsoft SQL Server Engine.
- SQL Database is a high-performance, reliable, and secure database you can use to build data-driven applications and websites in the programming language of your choice, without needing to manage infrastructure.

Use built-in intelligence to protect and optimize your database

- Azure SQL Database uses built-in intelligence that learns your unique database patterns and automatically tunes the database for improved performance and protection. Threat Detection monitors your database round-the-clock and detects potential malicious activities, alerting you upon detection so you can intervene right away.

Optimize performance for your workloads

- When demand for your app grows from a handful of devices and customers to millions, SQL Database scales, on the fly, with minimal downtime.
- Additionally, SQL Database provides in-memory OLTP that improves throughput and latency on transactional processing workloads up to 30x over traditional table and database engines and delivers faster business insights with up to 100X faster queries and reports over traditional row-oriented storage.[1]

Azure SQL Database

Build multitenant apps with customer isolation and efficiency

- If you're a software as a service (SaaS) app developer writing a multitenant app that serves many customers, you often have to make tradeoffs in customer performance, efficiencies, and security. SQL Database removes the compromise and helps you maximize your resource utilization and manage thousands of databases as one while ensuring one-customer-per-database with elastic pools.

Work in your preferred development environment

- SQL Database allows you to focus on what you do best: building great apps. Seamlessly enable DevOps by developing in SQL Server containers and deploying in SQL Database with the easy-to-use tools you already have, such as Visual Studio and SQL Server Management Studio. Or, build your applications with Python, Java, Node.js, PHP, Ruby, and .NET on the MacOS, Linux, and Windows platforms and deliver with the speed and efficiency your business demands

Helps protect and secure app data

- SQL Database helps you build security-enhanced apps in the cloud by providing advanced built-in protection and security featuresthat dynamically mask sensitive data and encrypt it at rest and in motion. Ensure high availability with three hot replicas and built-in automatic failover that guarantees a 99.99% availability SLA.[2] Accelerate recovery from catastrophic failures and regional outages to an RPO of less than 5 seconds with active-geo replication. With physical and operational security, SQL Database helps you meet the most stringent regulatory compliances, such as ISO/IEC 27001/27002, Fed RAMP/FISMA, SOC, HIPPA and PCI DSS.

Azure CLI for SQL Database

- SQL Databases can be created, deleted, copied and manipulated in all possible ways with Azure CLI, and PowerShell

```
C:\..> az sql --help
Group
    az sql: Manage Azure SQL Databases and Data Warehouses.
Subgroups:
    db          : Manage databases.
    dw          : Manage data warehouses.
    elastic-pool: Manage elastic pools.
    server      : Manage SQL servers.
```

az sql db --help

```
C:\..>az sql db --help

Group
    az sql db: Manage databases.

Subgroups:
    audit-policy          : Manage a database's auditing policy.
    op                    : Manage operations on a database.
    replica               : Manage replication between databases.
    tde                   : Manage a database's transparent data encryption.
    threat-policy         : Manage a database's threat detection policies.

Commands:
    copy                  : Create a copy of a database.
    create                : Create a database.
    delete                : Delete a database.
    export                : Export a database to a bacpac.
    import                : Imports a bacpac into an existing database.
    list                  : List databases a server or elastic pool.
    list-deleted          : Gets a list of deleted databases that can be
                            restored.
    list-editions         : Show database editions available for the currently
                            active subscription.
    list-usages           : Returns database usages.
    restore               : Create a new database by restoring from a backup.
    show                  : Get the details for a database.
    show-connection-string: Generates a connection string to a database.
    update                : Update a database.
```

az sql db create --help

```
C:\..> az sql db create --help
Command
    az sql db create: Create a database.
Arguments
    --name -n           [Required]: Name of the Azure SQL Database.
    --resource-group -g [Required]: Name of resource group. You can configure the
default group
                                    using `az configure --defaults group=<name>`.
    --server -s         [Required]: Name of the Azure SQL server.
    --collation                   : The collation of the database. If createMode is not
                                        Default,
                                    this value is ignored.
    --edition                     : The edition of the database.
    --elastic-pool                : The name of the elastic pool the database is in. If
                                    elasticPoolName and requestedServiceObjectiveName
                                    are both updated, the value of
                                    requestedServiceObjectiveName is ignored.
                                    Not supported for DataWarehouse edition.
    --max-size                    : The max storage size of the database. Only the
                                    following sizes are supported (in addition to
                                    limitations being placed on each edition): 100MB,
                                    500MB, 1GB, 5GB, 10GB, 20GB, 30GB, 150GB,
                                    200GB, 500GB. If no unit is specified, defaults to
                                    bytes (B).
```

az sql db create --help

```
    --no-wait        : Do not wait for the long running operation to finish.
    --sample-name    : Indicates the name of the sample schema to apply when creating
                       this database. If createMode is not Default, this value is
                       ignored. Not supported for DataWarehouse edition.
    --service-objective  : The name of the configured service level objective of the
                           database. This is the service level objective that is in the
                           process of being applied to the database. Once successfully
                           updated, it will match the value of serviceLevelObjective
                       property. To see possible values, query the capabilities API
                 (/subscriptions/{subscriptionId}/providers/Microsoft.Sql/locations/
                       {locationID}/capabilities) referred to by operationId:
                                   "Capabilities_ListByLocation.".
    --tags               : Resource tags.

Global Arguments
    --debug              : Increase logging verbosity to show all debug logs.
    --help -h            : Show this help message and exit.
    --output -o          : Output format.  Allowed values: json, jsonc, table, tsv.
                                   Default: json.
    --query              : JMESPath query string. See http://jmespath.org/ for more
                                   information and examples.
    --verbose            : Increase logging verbosity. Use --debug for full debug logs.
```

References

- All material in this set of slides originated from various pages of Azure documentation