

Azure for Architects

Second Edition

Implementing cloud design, DevOps, containers, IoT, and serverless solutions on your public cloud



Packt

www.packt.com

Ritesh Modi

Azure for Architects

Second Edition

Implementing cloud design, DevOps, containers, IoT,
and serverless solutions on your public cloud

Ritesh Modi



BIRMINGHAM - MUMBAI

Azure for Architects

Second Edition

Copyright © 2019 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Commissioning Editor: Vijn Boricha

Acquisition Editor: Shrilekha Inani

Content Development Editors: Abhishek Jadhav

Technical Editor: Aditya Khadye

Copy Editor: Safis Editing

Project Coordinator: Jagdish Prabhu

Proofreader: Safis Editing

Indexers: Priyanka Dhadke

Graphics: Tom Scaria

Production Coordinator: Shraddha Falebhai

First published: October 2017

Second edition: January 2019

Production reference: 1310119

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham B3 2PB, UK.

ISBN 978-1-78961-450-3

www.packtpub.com



mapt.io

Mapt is an online digital library that gives you full access to over 5,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Learn better with Skill Plans built especially for you
- Get a free eBook or video every month
- Mapt is fully searchable
- Copy and paste, print, and bookmark content

Packt.com

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.Packt.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at customercare@packtpub.com for more details.

At www.Packt.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Contributors

About the author

Ritesh Modi is an ex-Microsoft Senior Technology Evangelist. He is Microsoft regional director, as well as Regional lead for Microsoft Certified Trainers.

He is an architect, a senior evangelist, cloud architect, published author, speaker, and a known leader for his contributions towards blockchain, Ethereum, data centers, Azure, bots, cognitive services, DevOps, artificial intelligence, and automation. He is the author of five books.

He has spoken at more than 15 conferences including TechEd and PowerShell Asia, and is a published author for MSDN magazine. He has more than a decade of experience in building and deploying enterprise solutions for customers. He has more than 25 technical certifications.

I have personally grown into a person who has more patience, perseverance, and tenacity while writing this book. I must thank the people who mean the world to me. I am talking about my mother, Bimla Modi, my wife, Sangeeta Modi, and my daughter, Avni Modi. I also thank the Packt team for their support.

About the reviewers

Kasam Shaikh, a Microsoft Azure enthusiast, is a seasoned professional with a can-do attitude and 10 years of industry experience working as a cloud architect with one of the leading IT companies in Mumbai, India. He is a certified Azure architect, recognized as an MVP by a leading online community, as well as a global AI speaker, and has authored books on Azure Cognitive, Azure Bots, and Microsoft Bot frameworks. He is head of the Azure India (az-INDIA) community, the fastest growing online community for learning Azure.

Alexey Bokov is an experienced cloud architect, and has worked for Microsoft as Azure Technical Evangelist and Senior Engineer since 2011, where he helped software developers all around the world to develop applications based on the Azure platform. His main area of interest is security in cloud, and especially security and data protection for containerized applications

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit authors.packtpub.com and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Table of Contents

Preface	xiii
Chapter 1: Getting Started	1
Cloud computing	2
The advantages of cloud computing	3
Deployment patterns in Azure	3
IaaS	4
PaaS	4
SaaS	4
Understanding Azure	4
Azure as an intelligent cloud	6
ARM	6
The ARM architecture	7
Limitations of Azure Service Manager (ASM)	7
ARM advantages	8
ARM concepts	8
Resource providers	9
Resource types	9
Resource groups	9
Resources and resource instances	10
ARM features	10
Virtualization	12
Containers	12
Docker	14
Interacting with the intelligent cloud	14
The Azure portal	15
PowerShell	15
Azure CLI	16
The Azure REST API	16

Table of Contents

ARM templates	16
Deployments	17
Summary	17
Chapter 2: Azure Solution Availability and Scalability	19
High availability	20
SLA	21
Factors affecting high availability	21
Planned maintenance	21
Unplanned maintenance	21
Application deployment architecture	22
High availability versus scalability	22
High availability versus disaster recovery	22
Azure high availability	23
Concepts	23
Availability sets	23
The fault domain	24
The update domain	24
Availability zones	24
Load balancing	25
VM high availability	26
Computing high availability	26
Storage high availability	28
PaaS high availability	28
High-availability platforms	29
Data high availability	30
Azure Cosmos DB	30
Azure SQL replication	31
Azure Table storage	31
Application high availability	31
Load balancers in Azure	32
Azure load balancers	33
Public load balancing	33
Internal load balancing	34
Port forwarding	36
Azure Application Gateway	36
Azure Traffic Manager	37
Architectural considerations for high availability	38
High availability within Azure regions	39
High availability across Azure regions	40
Best practices	41
Application high availability	41
Deployment	41
Data management	42
Monitoring	42

Table of Contents

Scalability	42
Scalability versus performance	44
Azure scalability	44
Concepts	44
PaaS scalability	46
PaaS – scaling up and down	48
PaaS – scaling out and in	49
IaaS scalability	50
VMSS	51
VMSS architecture	52
VMSS scaling	52
Upgrades and maintenance	55
Application updates	56
Guest updates	56
Image updates	56
Best practices of scaling provided by VMSS	57
The preference for scaling out	57
Bare-metal versus dormant instances	57
Configuring the maximum and minimum number of instances appropriately	57
Concurrency	57
Stateless	58
Caching and the Content Distribution Network (CDN)	58
N+1 design	58
Summary	58
Chapter 3: Security and Monitoring	59
Security	60
Security life cycle	61
Azure security	63
IaaS security	64
Network security groups	64
NSG design	66
Firewalls	66
Reducing the attack surface area	68
Implementing jump servers	69
PaaS security	69
Log Analytics	70
Storage	71
Azure SQL	75
Azure Key Vault	78
Security monitoring and auditing	78
Azure Monitor	79
Azure Security Center	80
Monitoring	81

Table of Contents

Azure monitoring	82
Azure activity logs	82
Azure diagnostic logs	83
Azure application logs	83
Guest and host operating system logs	83
Azure Monitor	84
Azure Application Insights	84
Azure Log Analytics	84
Application Insights	84
Provisioning	85
Log Analytics	87
Provisioning	88
Log Analytics agents	90
Search	92
Solutions	93
Alerts	94
Executing runbooks on alerts	97
Integrating PowerBI	101
Summary	104
Chapter 4: Cross-Subscription Deployments Using ARM Templates	105
ARM templates	106
Deploying resource groups with ARM templates	109
Deploying resources across subscriptions and resource groups	112
Another example of cross-subscription and resource-group deployments	113
Deploying cross-subscription and resource-group deployments using linked templates	116
Summary	120
Chapter 5: ARM Templates Modular Design and Implementation	121
Problems with the single template	122
Reduced flexibility in changing templates	122
Troubleshooting large templates	122
Dependency abuse	122
Reduced agility	122
No reusability	123
Understanding the Single Responsibility Principle	123
Faster troubleshooting and debugging	123
Modular templates	124
Deployments resources	124

Table of Contents

Linked templates	125
Nested templates	126
Free-flow configurations	128
Known configurations	129
Summary	139
Chapter 6: Designing and Implementing Serverless Solutions	141
Serverless	142
The evolution of serverless	142
Principles of serverless technology	145
The advantages of Azure Functions	145
FaaS	147
Azure Functions runtime	147
Azure Functions bindings and triggers	147
Monitoring	150
Authentication and authorization	151
Azure Functions configuration	152
Platform configuration	152
App Service function settings	154
Azure Functions cost plans	154
Azure Functions use cases	155
Types of Azure Functions	156
Creating your first Azure Functions	156
Creating an event-driven function	160
Function proxies	163
Understanding workflows	164
Durable Functions	165
Steps for creating a Durable Functions	166
Creating a connected architecture with functions	172
Summary	176
Chapter 7: Azure Integration Solutions	177
Azure Event Grid	177
The Event Grid architecture	178
Resource events	181
Custom events	185
Azure Logic Apps	187
Activity	188
Connectors	188
Working on a logic app	188

Table of Contents

Creating an end-to-end solution using serverless technologies	197
The problem statement	197
Vision	197
Solution	198
Architecture	198
Azure Automation	199
A custom Azure Event Grid topic	200
Azure Logic Apps	200
Azure Functions	200
Prerequisites	200
Implementation	200
Step 1	201
Step 2	201
Step 3	203
Step 4	205
Step 5	206
Step 6	212
Step 7	214
Step 8	219
Step 9	226
Step 10	227
Step 11	236
Testing	244
Summary	245
Chapter 8: Cost Management	247
Understanding billing	248
Invoicing	252
Enterprise Agreement customers	253
Usage and quotas	254
Resource providers	254
The usage and billing APIs	255
Azure pricing models	255
Azure Hybrid Benefit	256
Azure reserved virtual machine instances	256
Pay-as-you-go accounts	256
Enterprise Agreements	257
The Cloud Solution Provider model	257
The Azure pricing calculator	257
Best practices	260
Compute best practices	260
Storage best practices	261
Platform as a Service (PaaS) best practices	262
General best practices	263

Table of Contents

Summary	263
Chapter 9: Designing Policies, Locks, and Tags	265
Azure tags	266
Tags with PowerShell	268
Tags with Azure Resource Manager templates	268
Resource groups versus resources	269
Azure policies	269
Built-in policies	270
Policy language	270
Allowed fields	272
Azure locks	273
Azure RBAC	274
Custom Roles	277
How are locks different from RBAC?	277
An example of implementing Azure governance features	277
Background	277
RBAC for Company Inc	278
Azure policies	278
Deployments to certain locations	278
Tags of resources and resource groups	278
Diagnostic logs and Application Insights for all resources	279
Azure locks	279
Summary	279
Chapter 10: Azure Solutions Using Azure Container Services	281
Azure Container Registry	282
Azure Container Instances	293
Azure Kubernetes Service	297
Kubernetes architecture	299
Master nodes	299
Pods	300
API server	300
Kubelets	300
Kube-Proxy	300
Replication controller/controller manager	300
Azure Kubernetes architecture	301
Provisioning Azure Kubernetes Service	301
App Service containers	306
Comparing all container options	311
Containers on virtual machines	311
Containers on virtual machines with Kubernetes as the orchestrator	312
Azure Kubernetes Service	312
Containers on Azure App Service	313

Table of Contents

Containers in Azure Container Instances	313
Containers in Azure Functions	314
Containers in Service Fabric	314
Summary	314
Chapter 11: Azure DevOps	315
DevOps	316
DevOps practices	319
Configuration management	320
Desired State Configuration	321
Chef, Puppet, and Ansible	322
ARM templates	322
Continuous integration	322
Build automation	324
Test automation	324
Packaging	324
Continuous deployment	325
Test environment deployment	326
Test automation	326
Staging environment deployment	327
Acceptance tests	327
Deployment to production	327
Continuous delivery	327
Continuous learning	327
Azure DevOps	328
TFVC	330
Git	331
Preparing for DevOps	331
Provisioning Azure DevOps organization	333
Provisioning the Azure Key Vault	333
Provisioning a configuration-management server/service	333
Provisioning log analytics	334
Azure Storage account	334
Source images	334
Monitoring tools	334
Management tools	335
DevOps for PaaS solutions	335
Azure App Services	336
Deployment slots	337
Azure SQL	337
The build-and-release pipeline	337
DevOps for virtual machine (IaaS)-based solutions	346
Azure Virtual Machines	347

Table of Contents

Azure public load balancers	347
The build pipeline	348
The release pipeline	349
DevOps for container-based (IaaS) solutions	350
Containers	350
Docker	351
Dockerfile	351
The build pipeline	351
The release pipeline	352
Azure DevOps and Jenkins	353
Azure Automation	355
Provisioning the Azure Automation account	356
Creating DSC configuration	357
Importing the DSC configuration	358
Compiling the DSC configuration	359
Assigning configurations to nodes	360
Browsing the server	360
Azure for DevOps	361
Summary	363
Chapter 12: Azure OLTP Solutions Using Azure SQL	
Sharding, Pools, and Hybrid	365
Azure cloud services	366
OLTP applications	367
Relational databases	367
Deployment models	368
Databases on Azure virtual machines	368
Databases hosted as managed services	369
Azure SQL Database	369
Application features	370
Single Instance	370
High availability	370
Backups	372
Geo-replication	373
Scalability	375
Security	375
Firewall	376
Azure SQL Server on dedicated networks	377
Encrypted databases at rest	379
Dynamic data masking	380
Azure Active Directory integration	381
Elastic pools	381
Managed Instance	383

Table of Contents

SQL database pricing	385
DTU-based pricing	385
vCPU-based pricing	387
How to choose the appropriate pricing model	388
Summary	389
Chapter 13: Azure Big Data Solutions with Azure Data Lake Storage and Data Factory	391
Data integration	391
ETL	392
A primer on Data Factory	393
A primer on Data Lake Storage	394
Understanding big data processing	395
Ingesting data	395
Processing data	395
Storing data	396
Presenting data	396
Migrating data from Azure Storage to Data Lake Gen2 Storage	396
Preparing the source storage account	396
Provisioning a new resource group	396
Provisioning a Storage account	397
Creating a new Data Lake Gen2 service	398
Provision Azure Data Factory Pipeline	399
Repository settings	401
Creating the first dataset	405
Creating the second dataset	408
Creating a third dataset	409
Creating a pipeline	411
Add one more copy data activity	412
Publishing	413
The final result	417
Summary	417
Chapter 14: Azure Stream Analytics and Event Hubs	419
Introducing events	420
Events	420
Event streaming	420
Event Hubs	422
The architecture of Event Hubs	423
Consumer groups	428
Throughput	429

A primer on Stream Analytics	430
The hosting environment	433
Streaming Units	433
A sample application using Event Hubs and Stream Analytics	434
Provisioning a new resource group	434
Creating an Event Hubs namespace	435
Creating an event hub	436
Provisioning a logic app	436
Provisioning the storage account	439
Creating a storage container	439
Creating Stream Analytics jobs	440
Running the application	442
Summary	443
Chapter 15: Designing IoT Solutions	445
IoT	445
IoT architecture	447
Connectivity	448
Identity	449
Capture	450
Ingestion	450
Storage	450
Transformation	450
Analytics	451
Presentation	452
Azure IoT	452
Identity	452
Capture	453
Ingestion	453
Storage	453
Transformation and analytics	454
Presentation	455
IoT Hubs	455
Protocols	456
Device registration	456
Message management	458
Device-to-cloud messaging	458
Cloud-to-device messaging	459
Security	460
Security in IoT	460

Table of Contents

Scalability	461
The SKU edition	461
Units	463
High availability	463
Summary	464
Other Books You May Enjoy	465
Index	469

Preface

Over the years, Azure cloud services have grown quickly, and the number of organizations adopting Azure for their cloud services has also been on the increase. Leading industry giants are discovering that Azure fulfills their extensive cloud requirements.

This book starts with an extensive introduction to all the categories of designs available with Azure. These design patterns focus on different aspects of the cloud, including high availability and data management. Gradually, we move on to various other aspects, such as building your cloud deployment and architecture. Every architect should have a good grasp of some of the important architectural concerns related to any application. These relate to high availability, security, scalability, and monitoring. They become all the more important because the entire premise of the cloud is dependent on these important concerns. This book will provide architects with all the important options related to scalability, availability, security, and the monitoring of **Infrastructure of a Service (IaaS)** and **Platform as a Service (PaaS)** deployments. Data has become one of the most important aspects of cloud applications. This book covers the architecture and design considerations for deploying **Online Transaction Processing (OLTP)** applications on Azure. Big data and related data activities, including data cleaning, filtering, formatting, and the use of **Extract-Transform-Load (ETL)** services are provided by the Azure Data Factory service. Finally, serverless technologies are gaining a lot of traction due to their orchestration using Azure Logic Apps. This will also be covered comprehensively in this book.

By the end of this book, you will be able to develop a fully-fledged Azure cloud instance.

Who this book is for

If you are a cloud architect, DevOps engineer, or developer who wants to learn about key architectural aspects of the Azure cloud platform, then this book is for you.

Prior basic knowledge of the Azure cloud platform is good to have.

What this book covers

Chapter 1, Getting Started, introduces the Azure cloud platform. It provides details regarding IaaS and PaaS and provides an introduction to some of the important features that help in designing solutions.

Chapter 2, Azure Solution Availability and Scalability, takes you through an architect's perspective for deploying highly available and scalable applications on Azure.

Chapter 3, Security and Monitoring, helps you to understand how security is undoubtedly the most important non-functional requirement for architects to implement.

Chapter 4, Cross-Subscription Deployments Using ARM Templates, explains how ARM templates are the preferred mechanism for provisioning resources.

Chapter 5, ARM Templates – Modular Design and Implementation, focuses on writing modular, maintainable, and extensible Azure Resource Manager (ARM) templates.

Chapter 6, Designing and Implementing Serverless Solutions, focuses on providing an explanation of the serverless paradigm, Azure Functions, and their capabilities.

Chapter 7, Azure Integration Solutions, is a continuation of the previous chapter, continuing the discussion on serverless technologies, covering Azure Event Grid as part of serverless events, and Azure Logic Apps as part of serverless workflows.

Chapter 8, Cost Management, focuses on calculating the cost of deployment on Azure using the Azure cost calculator. It also demonstrates how changing the location, size, and type of resources affects the cost of solutions and provides best practices for reducing the overall cost of Azure deployments.

Chapter 9, Designing Policies, Locks, and Tags, helps you to understand the best practices for implementing policies and locks, and how both can work together to provide complete control over Azure resources.

Chapter 10, Azure Solutions Using Azure Container Services, sheds some light on numerous services, including Azure Container Services, Azure Container Registry, and Azure Container Instances for hosting containers, as well managing them using orchestration services such as Kubernetes.

Chapter 11, Azure DevOps, is about adopting and implementing practices that reduce risk considerably and ensure that high-quality software can be delivered to the customer.

Chapter 12, Azure OLTP Solutions Using Azure SQL Sharding, Pools, and Hybrid, focuses on various aspects of using the transaction data store, such as Azure SQL, and other open source databases typically used in OLTP applications.

Chapter 13, Azure Big Data Solutions Using Azure Data Lake Storage and Data Factory, focuses on big data solutions on Azure. We will study Data Lake Storage, Data Lake Analytics, and Data Factory.

Chapter 14, Azure Stream Analytics and Event Hubs, concerns the creation of solutions for these events. It focuses on reading these events, storing and processing them, and then making sense of them.

Chapter 15, Designing IoT Solutions, covers topics related to IoT Hub, Stream Analytics, Event Hubs, registering devices, device-to-platform conversion, and logging and routing data to appropriate destinations.

To get the most out of this book

This book assumes a basic level of knowledge of cloud computing and Azure. To use this book, all you need is a valid Azure subscription and internet connectivity. A Windows 10 OS with 4 GB of RAM is sufficient for using PowerShell and executing ARM templates. Click on the following link to create your Azure account: <https://azure.microsoft.com/enus/free/>.

Download the EPUB/mobi and example code files

An EPUB and mobi version of this book is available free of charge on Github. You can download them and the code bundle at <https://github.com/PacktPublishing/Azure-for-Architect-Second-Edition>.

You can download the example code files for this book from your account at <http://www.packt.com>. If you purchased this book elsewhere, you can visit <http://www.packt.com/support> and register to have the files emailed directly to you.

You can download the code files by following these steps:

1. Log in or register at <http://www.packt.com>.
2. Select the **SUPPORT** tab.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box and follow the on-screen instructions.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

The code bundle for the book is also hosted on GitHub at <https://github.com/PacktPublishing/Azure-for-Architect-Second-Edition>. In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at <https://github.com/PacktPublishing/>. Check them out!

Conventions used

There are a number of text conventions used throughout this book.

CodeInText: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. For example; "Browse to the extracted *.ova file for Kali Linux and click **Open**."

A block of code is set as follows:

```
html, body, #map {  
    height: 100%;  
    margin: 0;  
    padding: 0  
}
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

Any command-line input or output is written as follows:

```
$ mkdir css
$ cd css
```

Bold: Indicates a new term, an important word, or words that you see on the screen, for example, in menus or dialog boxes, also appear in the text like this. For example: "Select **System info** from the **Administration** panel."



Warnings or important notes appear like this.



Tips and tricks appear like this.



Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at customercare@packtpub.com.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book we would be grateful if you would report this to us. Please visit, <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at copyright@packt.com with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit <http://authors.packtpub.com>.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit packt.com.

1

Getting Started

Every few years, technological innovations emerge that change the entire landscape and ecosystem around them. If we go back in time, you will recall that the 1970s and 1980s were the time of mainframes. These mainframes were huge – occupying large rooms – and almost all computing work was carried out by them. They were difficult to procure and also time-consuming to use. Many enterprises used to order one months in advance before they could have an operational mainframe set up.

Then, the first part of the 1990s was the era of personal computing and the internet. Computers became much smaller in size and were comparatively easier to procure. Continuous innovation on the personal computing and internet fronts changed the entire computer industry. Many people had a desktop computer on which they could run multiple programs and connect to the internet. The rise of the internet also propagated the rise of client-server deployments. Now there could be centralized servers hosting applications, and services that could be reached by anyone who had a connection to the internet anywhere on the globe. This was also a time when server technology gained a lot of prominence; Windows NT was released during this time and was soon followed by Windows 2000 and Windows 2003 at the turn of the century.

The most remarkable innovation of the 2000s was the rise and adoption of portable devices, especially smartphones, and with these came a plethora of apps. Apps could connect to centralized servers on the internet and carry out business as usual. Users were no longer dependent on browsers in order to do this work; all servers were either self-hosted or hosted using a service provider, such as an **Internet Service Provider (ISP)**.

Users did not have much control over their servers. Multiple customers and their deployments were part of the same server, even without customers knowing about it.

However, there was something else happening toward the middle and later parts of the first decade of the 2000s. This was the rise of cloud computing, and it again rewrote the entire landscape of the IT industry. Initially, adoption was slow, and people approached it with caution, either because the cloud was in its infancy and still had to mature, or because people had various negative notions about what it was.

We will cover the following topics in the chapter:

- Cloud computing
- **Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS)**
- Understanding Azure
- **Azure Resource Manager (ARM)**
- Virtualization, containers, and Docker
- Interacting with the intelligent cloud

Cloud computing

Today, cloud computing is one of the most promising upcoming technologies, and enterprises, no matter how big or small, are adopting it as a part of their IT strategy. It is difficult these days to have any meaningful conversation about an IT strategy without including cloud computing in the overall solution discussions.

Cloud computing, or simply the cloud in layman's terms, refers to the availability of resources on the internet. These resources are made available to users on the internet as services. For example, storage is available on demand through the internet for users to use to store their files, documents, and more. Here, storage is a service that is offered by a cloud provider.

A cloud provider is an enterprise or consortium of companies that provides cloud services to other enterprises and consumers. They host and manage these services on behalf of the user. They are responsible for enabling and maintaining the health of services. Typically, there are large data centers across the globe that have been opened by cloud providers in order to cater to the IT demands of users.

Cloud resources consist of hosting services on on-demand infrastructure, such as computing, networks, and storage facilities. This flavor of the cloud is known as IaaS.

The advantages of cloud computing

Cloud adoption is at an all-time high and is growing because of several advantages, such as these:

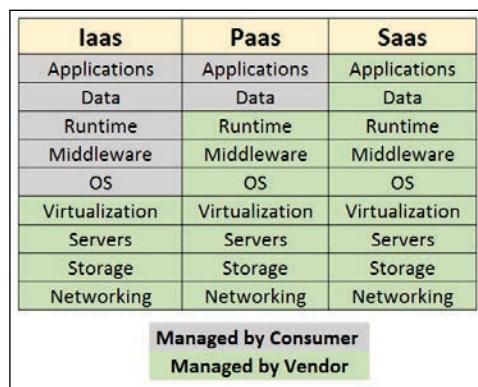
- **Pay-as-you-go model:** Customers do not need to purchase hardware and software for cloud resources. There is no capital expenditure for using a cloud resource; customers simply pay for the time that they use a resource for.
- **Global access:** Cloud resources are available globally through the internet. Customers can access their resources on demand from anywhere.
- **Unlimited resources:** The scale of the cloud is unlimited; customers can provision as many resources as they want, without any constraints. This is also known as unlimited scalability.
- **Managed services:** The cloud provider provides numerous services that are managed by them for customers. This takes away any technical and financial burdens from the customer.

Deployment patterns in Azure

There are three different deployment patterns that are available in Azure; they are as follows:

- IaaS
- PaaS
- SaaS

The difference between these three deployment patterns is the level of control that is exercised by customers via Azure. The following diagram displays the different levels of control within each of these deployment patterns:



Cloud services – IaaS, PaaS, and SaaS

It is clear from the previous diagram that customers have more control when using IaaS deployments and this level of control continually decreases as you progress from PaaS to SaaS deployments.

IaaS

IaaS is a deployment type that allows customers to provision their own infrastructure on Azure. Azure provides a number of infrastructure resources and customers can provision these on demand. Customers are responsible for maintaining and governing their own infrastructure. Azure will ensure the maintenance of the physical infrastructure on which these virtual infrastructure resources are hosted. Under this approach, customers require active management and operations in the Azure environment.

PaaS

PaaS takes away infrastructure deployment and control from the customer. This is a higher-level abstraction compared to IaaS. In this approach, customers bring their own application, code, and data, and deploy them on the Azure-provided platform. These platforms are managed and governed by Azure and customers are solely responsible for their applications. Customers perform activities related to their application deployment only. This model provides faster and easier options for the deployment of applications compared to IaaS.

SaaS

SaaS is a higher-level abstraction compared to PaaS. In this approach, software and its services are available for end user consumption. Customers only bring their data into these services and they do not have any control over these services.

Understanding Azure

Azure provides all the benefits of the cloud while remaining open and flexible. Azure supports a wide variety of operating systems, languages, tools, platforms, utilities, and frameworks. For example, it supports Linux and Windows, SQL Server, MySQL, and PostgreSQL; it supports languages such as C#, Python, Java, Node.js, and Bash; it supports NoSQL databases such as MongoDB and Cosmos DB; and it also supports continuous integration tools, such as Jenkins and **Visual Studio Team Services (VSTS)**.

The whole idea behind this ecosystem is to enable users to have the freedom to choose their own language, platform and operating system, database, storage, and tools and utilities. Users should not be constrained from the technology perspective; instead, they should be able to build and focus on their business solution, and Azure provides them with a world-class technology stack that they can use.

Azure is very much compatible with the user's choice of technology stack. For example, Azure supports all popular (open source and commercial) database environments. Azure provides Azure SQL, MySQL, and Postgres PaaS services. It provides a Hadoop ecosystem and offers HDInsight, a 100% Apache Hadoop-based PaaS. It also provides a Hadoop on Linux **virtual machine (VM)** implementation for customers who prefer the IaaS approach. Azure also provides a Redis Cache service and supports other popular database environments, such as Cassandra, Couchbase, and Oracle as an IaaS implementation.

The number of services is increasing by the day in Azure and the most updated list of services can be found at <https://azure.microsoft.com/en-in/services/>.

Azure also provides a unique cloud computing paradigm – known as the hybrid cloud. The hybrid cloud refers to a deployment strategy in which a subset of services is deployed on a public cloud, while other services are deployed in an on-premises private cloud or data center. There is a **Virtual Private Network (VPN)** connection between both the public and private cloud. Azure offers users the flexibility to divide and deploy their workload on both the public cloud and an on-premises data center.

Azure has data centers across the globe and combines these data centers into regions. Each region has multiple data centers to ensure that recovery from disasters is quick and efficient. At the time of writing, there are 38 regions across the globe. This provides users with the flexibility to deploy their services in their choice of location and region. They can also combine these regions to deploy a solution that is disaster-resistant and deployed near their customer base.



In China and Germany, the Azure cloud is separate for general use and for governmental use. This means that the cloud services are maintained in separate data centers.

Azure as an intelligent cloud

Azure provides infrastructure and services to ingest millions and billions of transactions using hyper-scale processing. It provides multi-petabytes of storage for data, and it provides a host of interconnected services that can pass data among themselves. With such capabilities in place, data can be processed to generate meaningful knowledge and insights. There are multiple types of insights that can be generated through data analysis, which are as follows:

- **Descriptive:** This type of analysis provides details about what is happening or has happened in the past.
- **Predictive:** This type of analysis provides details about what is going to happen in the future or the near future.
- **Prescriptive:** This type of analysis provides details about what should be done to either enhance or prevent current or future events from happening.
- **Cognitive:** This type of analysis actually executes actions that are determined by prescriptive analytics in an automated manner.

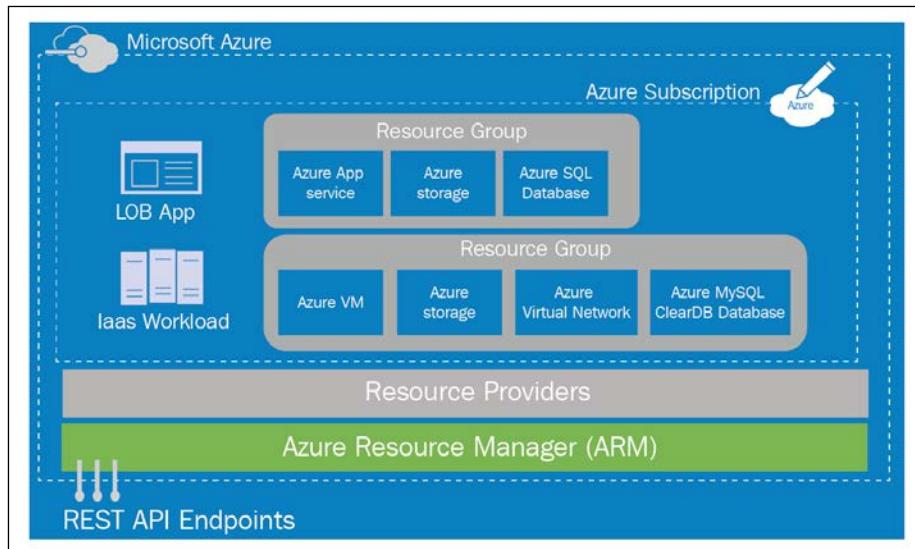
While deriving insights from data is good, it is equally important to act on them. Azure provides a rich platform in which to ingest large volumes of data, process and transform it, store and generate insights from it, and display it on real-time dashboards. It is also possible to take action on these insights automatically. These services are available to every user of Azure and they provide a rich ecosystem on which to create solutions. Enterprises are creating numerous applications and services that are completely disrupting industries because of the easy availability of these intelligent services from Azure, which are combined to create meaningful value for end customers. Azure ensures that services that are commercially nonviable to implement by small and medium companies can now be readily consumed and deployed in a few minutes.

ARM

ARM is the technology platform and orchestration service from Microsoft that ties up all the components that were discussed earlier. It brings Azure's resource providers, resources, and resource groups together to form a cohesive cloud platform. It makes Azure Services available to subscriptions, the resource types available to resource groups, the resource and resource APIs accessible to the portal and other clients, and it authenticates access to these resources. It also enables features such as tagging, authentication, **Role-Based Access Control (RBAC)**, resource locking, and policy enforcement for subscriptions and its resource groups. It also provides deployment and management features using the Azure portal, Azure PowerShell, and command-line interface tools.

The ARM architecture

The architecture of ARM and its components are shown in the following diagram. As you can see in the following diagram, **Azure Subscription** comprises multiple resource groups. Each resource group contains resource instances that are created from resource types that are available in the resource provider:



The ARM architecture

Limitations of Azure Service Manager (ASM)

ASM has inherent constraints. For example, ASM deployments are slow and blocking; operations are blocked if an earlier operation is already in progress. Some of the limitations of ASM are as follows:

- **Parallelism:** Parallelism is a challenge in ASM. It is not possible to execute multiple transactions successfully in parallel. The operations in ASM are linear and so they are executed one after another. If multiple transactions are executed at the same time, there will either be parallel operation errors or the transactions will get blocked.
- **Resources:** Resources in ASM are provisioned and managed in isolation of each other; there is no relation between ASM resources. Grouping services and resources or configuring them together is not possible.
- **Cloud services:** Cloud services are the units of deployment in ASM. They are reliant on affinity groups and are not scalable due to their design and architecture.

Granular and discrete roles and permissions cannot be assigned to resources in ASM. Users are either service administrators or co-administrators in the subscription. They either get full control over resources or do not have access to them at all. ASM provides no deployment support. Deployments are either manual or you will need to resort to writing procedural scripts in PowerShell or .NET. ASM APIs are not consistent between resources.

ARM advantages

ARM provides distinct advantages and benefits over ASM, which are as follows:

- **Grouping:** ARM allows the grouping of resources together in a logical container. These resources can be managed together and go through a common life cycle as a group. This makes it easier to identify related and dependent resources.
- **Common life cycles:** Resources in a group have the same life cycle. These resources can evolve and be managed together as a unit.
- **RBAC:** Granular roles and permissions can be assigned to resources providing discrete access to users. Users can also have only those rights that are assigned to them.
- **Deployment support:** ARM provides deployment support in terms of templates enabling DevOps and **Infrastructure as Code (IaC)**. These deployments are faster, consistent, and predictable.
- **Superior technology:** The cost and billing of resources can be managed as a unit. Each resource group can provide their usage and cost information.
- **Manageability:** ARM provides advanced features, such as security, monitoring, auditing, and tagging, for better manageability of resources. Resources can be queried based on tags. Tags also provide cost and billing information for resources that are tagged similarly.
- **Migration:** There is easier migration and updating of resources within and across resource groups.

ARM concepts

With ARM, everything in Azure is a resource. Examples of resources are VMs, network interfaces, public IP addresses, storage accounts, and virtual networks. ARM is based on concepts that are related to resource providers and resource consumers. Azure provides resources and services through multiple resource providers that are consumed and deployed in groups.

Resource providers

These are services that are responsible for providing resource types through ARM. The top-level concept in ARM is the resource provider. These providers are containers for resource types. Resource types are grouped into resource providers. They are responsible for deploying and managing resources. For example, a VM resource type is provided by a resource provider called **Microsoft.Compute Namespace**. The **Representational State Transfer (REST)** API operations are versioned to distinguish between them. The version naming is based on the dates on which they are released by Microsoft. It is necessary that a related resource provider is available to a subscription to deploy a resource. Not all resource providers are available to a subscription out of the box. If a resource is not available in the subscription, then you need to check whether the required resource provider is available in each region. If it is available, the user can explicitly register in the subscription.

Resource types

Resource types are an actual resource specification defining the resource's public API interface and implementation. They implement the working and operations supported by the resource. Similar to resource providers, resource types also evolve over time in terms of their internal implementation, and there are multiple versions of their schemas and public API interfaces. The version names are based on the dates that they are released by Microsoft as a preview or **General Availability (GA)**. The resource types become available as a subscription after a resource provider is registered to it. Also, not every resource type is available in every Azure region. The availability of a resource is dependent on the availability and registration of a resource provider in an Azure region and must support the API version needed for provisioning it.

Resource groups

Resource groups are units of deployment in ARM. They are containers grouping multiple resource instances in a security and management boundary. A resource group is uniquely named in a subscription. Resources can be provisioned on different Azure regions and yet belong to the same resource group. Resource groups provide additional services to all the resources within it. Resource groups provide metadata services, such as tagging, which enables the categorization of resources, the policy-based management of resources, RBAC, the protection of resources from accidental deletion or updates, and more. As mentioned before, they have a security boundary, and users that don't have access to a resource group cannot access resources contained within it. Every resource instance needs to be part of a resource group; otherwise, it cannot be deployed.

Resources and resource instances

Resources are created from resource types and should be unique within a resource group. The uniqueness is defined by the name of the resource and its type together. If we compare this with object-oriented programming constructs, resource instances can be seen as objects and resource types can be seen as classes. The services are consumed through the operations that are supported and implemented by resource instances. They define properties that should be configured before usage. Some are mandatory properties, while others are optional. They inherit the security and access configuration from their parent resource group. These inherited permissions and role assignments can be overridden for each resource. A resource can be locked in such a way that some of its operations can be blocked and not made available to roles, users, and groups even though they have access to it. Resources can be tagged for easy discoverability and manageability.

ARM features

Here are some of the major features that are provided by ARM:

- **RBAC: Azure Active Directory (Azure AD)** authenticates users to provide access to subscriptions, resource groups, and resources. ARM implements OAuth and RBAC within the platform, enabling authorization and access control for resources, resource groups, and subscriptions based on roles assigned to a user or group. A permission defines access to the operations in a resource. These permissions can allow or deny access to the resource. A role definition is a collection of these permissions. Roles map Azure AD users and groups to particular permissions. Roles are subsequently assigned to a scope; this can be an individual, a collection of resources, a resource group, or the subscription. The Azure AD identities (users, groups, and service principles) that are added to a role gain access to the resource according to the permissions defined in the role. ARM provides multiple out-of-the-box roles. It provides system roles, such as the **owner**, **contributor**, and **reader**. It also provides resource-based roles, such as SQL DB contributor and VM contributor. ARM also allows the creation of custom roles.

- **Tags:** Tags are name-value pairs that add additional information and metadata to resources. Both resources and resource groups can be tagged with multiple tags. Tags help in the categorization of resources for better discoverability and manageability. Resources can be quickly searched for and easily identified. Billing and cost information can also be fetched for resources that have the same tags. While this feature is provided by ARM, an IT administrator defines its usage and taxonomy with regard to resources and resource groups. Taxonomy and tags, for example, can relate to departments, resource usage, location, projects, or any other criteria that is deemed fit from a cost, usage, billing, or search perspective. These tags can then be applied to resources. Tags that are defined at the resource group level are not inherited by their resources.
- **Policies:** Another security feature that is provided by ARM is custom policies. Custom policies can be created to control access to resources. Policies are defined as conventions and rules, and they must be adhered to while interacting with resources and resource groups. The policy definition contains an explicit denial of actions on resources or access to resources. By default, every access is allowed if it is not mentioned in the policy definition. These policy definitions are assigned to the resource, resource group, and subscription scope. It is important to note that these policies are not replacements or substitutes for RBAC. In fact, they complement and work together with RBAC. Policies are evaluated after a user is authenticated by Azure AD and authorized by the RBAC service. ARM provides a JSON-based policy definition language for defining policies. Some examples of policy definition are that a policy must tag every provisioned resource, and resources can only be provisioned to specific Azure regions.
- **Locks:** Subscriptions, resource groups, and resources can be locked to prevent accidental deletion or updates by an authenticated user. Locks applied at higher levels flow downstream to the child resources. Alternatively, locks that are applied at the subscription level lock every resource group and the resources within it.
- **Multi-region:** Azure provides multiple regions for provisioning and hosting resources. ARM allows resources to be provisioned at different locations while still residing within the same resource group. A resource group can contain resources from different regions.
- **Idempotent:** This feature ensures predictability, standardization, and consistency in resource deployment by ensuring that every deployment will result in the same state of resources and configuration, no matter the number of times it is executed.
- **Extensible:** ARM provides an extensible architecture to allow creating and plugging new resource providers and resource types onto the platform.

Virtualization

Virtualization was a breakthrough innovation that completely changed the way that physical servers were looked at. It refers to the abstraction of a physical object into a logical object.

The virtualization of physical servers led to virtual servers known as VMs. These VMs consume and share the same physical CPU, memory, storage, and other hardware with the physical server on which they are hosted. This enabled faster and easier provisioning of application environments on demand, providing high availability and scalability with reduced cost. One physical server was enough to host multiple VMs with each VM containing its own operating system and hosting services on it.

There was no longer any need to buy additional physical servers for deploying new applications and services. The existing physical servers were sufficient to host more VMs. Furthermore, as part of rationalization, many physical servers were consolidated into a few with the help of virtualization.

Each VM contains the entire operating system, and each VM is completely isolated from other VMs, including the physical hosts. Although a VM uses the hardware that is provided by the host physical server, it has full control over its assigned resources and its environment. These VMs can be hosted on a network such as a physical server with its own identity.

Azure can create Linux and Windows VMs in a few minutes. Microsoft provides its own images, along with images from its partners and the community; users can also provide their own images. VMs are created using these images.

Containers

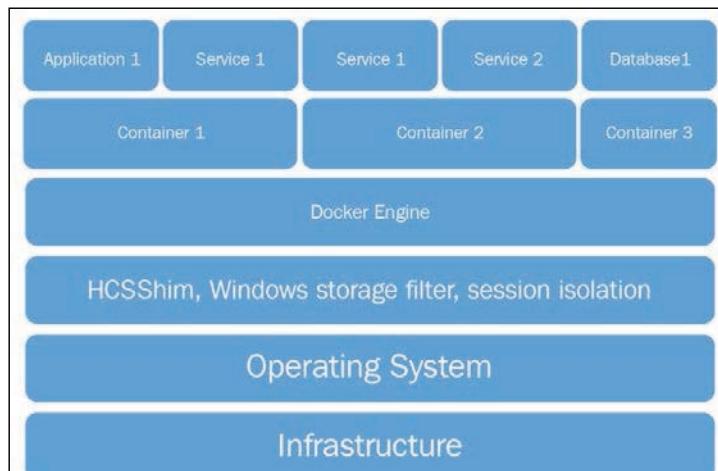
Containers are also a virtualization technology; however, they do not virtualize a server. Instead, a container is an operating system-level virtualization. What this means is that containers share the operating system kernel (which is provided by the host) among themselves along with the host. Multiple containers running on a host (physical or virtual) share the host operating system kernel. Containers ensure that they reuse the host kernel instead of each having a dedicated kernel to themselves.

Containers are completely isolated from their host or from other containers running on the host. Windows containers use Windows storage filter drivers and session isolation to isolate operating system services such as the filesystem, registry, processes, and networks. The same is true even for Linux containers running on Linux hosts. Linux containers use the Linux namespace, control groups, and union filesystem to virtualize the host operating system.

The container appears as if it has a completely new and untouched operating system and resources. This arrangement provides lots of benefits, such as the following:

- Containers are fast to provision and take a few seconds to create. Most of the services in a container are provided by the host operating system.
- Containers are lightweight and require fewer computing resources than VMs. The operating system resource overhead is no longer required with containers.
- Containers are much smaller than VMs.
- Containers can help solve problems related to managing multiple application dependencies in an intuitive, automated, and simple manner.
- Containers provide infrastructure in order to define all application dependencies in a single place.

Containers are an inherent feature of Windows Server 2016 and Windows 10; however, they are managed and accessed using a Docker client and a Docker daemon. Containers can be created on Azure with a Windows Server 2016 SKU as an image. Each container has a single main process that must be running for the container to exist. A container will stop when this process ends. Additionally, a container can either run in the interactive mode or in detached mode like a service:



Container architecture

The preceding diagram shows all the technical layers that enable containers. The bottom-most layer provides the core infrastructure in terms of network, storage, load balancers, and network cards. At the top of the infrastructure is the compute layer, consisting of either a physical server or both physical and virtual servers on top of a physical server. This layer contains the operating system with the ability to host containers. The operating system provides the execution driver that the layers above use to call the kernel code and objects to execute containers. Microsoft has created **Host Container System Shim (HCSShim)** for managing and creating containers and uses Windows storage filter drivers for image and file management.

The container environment isolation ability is provided to the Windows session. Windows Server 2016 and Nano Server provide the operating system, enable the container features, and execute the user-level Docker client and Docker engine. The Docker engine uses the services of HCSShim, storage filter drivers, and sessions to spawn multiple containers on the server, with each containing a service, application, or database.

Docker

Docker provides management features to Windows containers. It comprises the following two executables:

- The Docker daemon
- The Docker client

The Docker daemon is the workhorse for managing containers. It is a Windows service responsible for managing all activities on the host that are related to containers. The Docker client interacts with the Docker daemon and is responsible for capturing inputs and sending them across to the Docker daemon. The Docker daemon provides the runtime, libraries, graph drivers, and engine to create, manage, and monitor containers and images on the host server. It also has the ability to create custom images that are used for building and shipping applications to multiple environments.

Interacting with the intelligent cloud

Azure provides multiple ways in which to connect, automate, and interact with the intelligent cloud. All these methods require users to be authenticated with valid credentials before they can be used. The different ways to connect to Azure are as follows:

- The Azure portal
- PowerShell

- Azure Command-Line Interface (CLI)
- Azure REST API

The Azure portal

The Azure portal is a great place to get started. With the Azure portal, users can log in and start creating and managing Azure resources manually. The portal provides an intuitive and user-friendly user interface through the browser. The Azure Portal provides an easy way to navigate to resources using **blades**. The blades display all the properties of a resource, including its logs, cost, its relationship with other resources, tags, security options, and more. The entire cloud deployment can be managed through the Portal.

PowerShell

PowerShell is an object-based command-line shell and scripting language that is used for the administration, configuration, and management of infrastructure and environments. It is built on top of the .NET framework and provides automation capabilities. PowerShell has truly become a first-class citizen among IT administrators and automation developers for managing and controlling the Windows environment. Today, almost every Windows environment and many Linux environments can be managed by PowerShell. In fact, almost every aspect of Azure can also be managed by PowerShell. Azure provides rich support for PowerShell. It provides a PowerShell module for each resource provider containing hundreds of cmdlets. Users can use these cmdlets in their scripts to automate interaction with Azure. The Azure PowerShell module is available through the web platform installer and through the **PowerShell Gallery**. Windows Server 2016 and Windows 10 provide package management and `PowerShellGet` modules for the quick and easy downloading and installation of PowerShell modules from the PowerShell Gallery. The `PowerShellGet` module provides the `Install-Module` cmdlet for downloading and installing modules on the system.

Installing a module is a simple act of copying the module files at well-defined module locations, which can be done as follows:

```
Import-Module PowerShellGet  
Install-Module -Name AzureRM -verbose
```

Azure CLI

Azure also provides Azure CLI 2.0, which can be deployed on Linux, Windows, and macOS operating systems. Azure CLI 2.0 is Azure's new command-line utility for managing Azure resources. Azure CLI 2.0 is optimized for managing and administering Azure resources from the command line, and for building automation scripts that work against ARM. The CLI can be used to execute commands using the Bash shell or Windows command line. Azure CLI is very famous among non-Windows users as it allows us to talk to Azure on Linux and macOS. The steps for installing Azure CLI 2 are available at <https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest>.

The Azure REST API

All Azure resources are exposed to users through REST endpoints. REST APIs are service endpoints that implement HTTP operations (or methods) by providing **create, retrieve, update, or delete (CRUD)** access to the service's resources. Users can consume these APIs to create and manage resources. In fact, the CLI and PowerShell mechanism uses these REST APIs internally to interact with resources on Azure.

ARM templates

In an earlier section, we looked at deployment features, such as multi-service, multi-region, extensible, and idempotent features that are provided by ARM. ARM templates are the primary means of provisioning resources in ARM. ARM templates provide implementation support for ARM's deployment features.

ARM templates provide a declarative model through which resources, their configuration, scripts, and extensions are specified. ARM templates are based on the **JavaScript Object Notation (JSON)** format. They use JSON syntax and conventions to declare and configure resources. JSON files are text-based, user-friendly, and easily readable files.

They can be stored in a source code repository and have version control. They are also a means to represent IaC that can be used to provision resources in an Azure resource group again and again, predictably and uniformly. A template needs a resource group for deployment. It can only be deployed to a resource group, and the resource group should exist before executing a template deployment. A template is not capable of creating a resource group.

Templates provide the flexibility to be generic and modular in their design and implementation. Templates provide the ability to accept parameters from users, declare internal variables, define dependencies between resources, link resources within the same resource group or different resource groups, and execute other templates. They also provide scripting language type expressions and functions that make them dynamic and customizable at runtime.

Deployments

PowerShell allows the following two modes for the deployment of templates:

- **Incremental:** Incremental deployment adds resources declared in the template that don't exist in a resource group, leaves resources unchanged in a resource group that is not part of a template definition, and leaves resources unchanged in a resource group that exists in both the template and resource group with the same configuration state.
- **Complete:** Complete deployment, on the other hand, adds resources declared in a template to the resource group, deletes resources that do not exist in the template from the resource group, and leaves resources unchanged that exist in both the resource group and template with the same configuration state.

Summary

The cloud is not more than 10 years old. It is a new paradigm and still in its nascent stage. There will be a lot of innovation and capabilities added over time. Azure is one of the top cloud providers today and it provides rich capabilities through IaaS, PaaS, SaaS, and hybrid deployments. In fact, the Azure stack, which is an implementation of the private cloud from Microsoft, will be released soon. This will have the same features available on a private cloud as on the public cloud. They both will, in fact, connect and work seamlessly and transparently together.

It is very easy to get started with Azure, but developers and architects can also fall into a trap if they do not design and architect their solutions appropriately. This book is an attempt to provide guidance and directions toward architecting solutions the right way, using appropriate services and resources. Every service on Azure is a resource. It is important to understand how these resources are organized and managed in Azure.

This chapter provided context around ARM and groups – which are the core frameworks that provide the building blocks for resources. ARM offers a set of services to resources that help provide uniformity, standardization, and consistency in managing them. The services, such as RBAC, tags, policies, and locks, are available to every resource provider and resource. Azure also provides rich automation features to automate and interact with resources. Tools such as PowerShell, ARM templates, and Azure CLI can be incorporated as part of release pipelines, continuous deployment, and delivery. Users can connect to Azure from heterogeneous environments using these automation tools.

The next chapter will discuss some of the important architectural concerns that help solve common cloud-based deployment problems and ensure the application is secure, available, scalable, and maintainable in the long run.

2

Azure Solution Availability and Scalability

Architectural concerns, such as high availability and scalability, are some of the highest-priority items for any architect. This is common across many projects and solutions. However, this becomes even more important when deploying applications on the cloud because of the complexity involved. Most of the time, the complexity does not come from the application, but from the choices available in terms of similar resources on the cloud. The other complex issue that arises from the cloud is the constant availability of newer features. These new features can almost make an architect's architectural decisions completely redundant in hindsight.

In this chapter, we will go through an architect's perspective for deploying highly available and scalable applications on Azure.

Azure is a mature platform providing multiple options for implementing high availability and scalability at multiple levels. It is vital for an architect to know about them, including the differences between them and the costs involved, and finally, choose an appropriate solution that meets the best solution requirements. There is no one solution, but a good one for each project.

Running applications and systems that are available to users for consumption whenever they need them is one of the topmost priorities for organizations. They want their applications to be operational, functional, and to continue to be available to their customers even when some untoward event happens. High availability is the primary theme of this chapter. *Keeping the lights on* is the common metaphor that is used for high availability. Achieving high availability for applications is not an easy task, and organizations have to spend considerable time, energy, resources, and money to do so. Additionally, there is still the risk that an organization's implementation will not produce the desired results.

Azure provides a lot of high-availability features for **virtual machines (VMs)** and the **Platform-as-a-Service (PaaS)** service. In this chapter, we will go through the architectural and design features that are provided by Azure to ensure high availability for applications and services.

In this chapter, we will cover the following topics:

- High availability
- Azure high availability
- Architectural considerations for high availability
- Scalability
- Upgrades and maintenance

High availability

High availability is one of the major architectural concerns for any architect. It forms one of the core non-functional technical requirements for any serious service and its deployment. High availability refers to the feature of a service or application that keeps it operational on a continuous basis; it does so by meeting or surpassing its promised **service level agreement (SLA)**. Users are promised a certain SLA based on service type. The service should be available for consumption based on its SLA. For example, an SLA can define 99% availability for an application for the entire year. This means that it should be available for consumption by users for 361.35 days. If it becomes less than this, this constitutes a breach of the SLA. Most mission-critical applications define their high-availability SLA with 99.999% availability for a year. This means the application should be up, running, and available throughout the year, but it can only be down and unavailable for 5.2 hours.

It is important to note here that high availability is defined in terms of time (yearly, monthly, weekly, or a combination of these).

A service or application is made up of multiple components and these components are deployed on separate tiers and layers. Moreover, a service or application is deployed on an **operating system (OS)** and hosted on a physical machine or VM. It consumes network and storage services for various purposes. It might even be dependent on external systems. For these services or applications to be highly available, it is important that networks, storage, OSes, VMs or physical machines, and each component of the application is designed with the SLA and high availability in mind. A definite application life cycle process used to ensure high availability should be baked in from the start of application planning until its introduction to operations. This also involves introducing redundancy. Redundant resources should be included in the overall application and deployment architecture to ensure that if one resource goes down, another takes over and serves the requests of the customer.

SLA

An SLA is an agreement between two or more parties, where one is the customer, and the others are service providers. Particular aspects of the service – quality, availability, and responsibilities – are agreed between the service provider and the service user. The most common component of the SLA is that the services should be provided to the customer as agreed upon in the contract.

Factors affecting high availability

Planned maintenance, unplanned maintenance, and application deployment architecture are the major factors affecting the high availability of an application. We will be looking into each of these factors in the following sections.

Planned maintenance

Planned maintenance refers to the process of keeping the application and its surrounding ecosystem – comprising platforms, frameworks, software, the OS, and host and guest drivers – up to date with the latest stable releases. It is important to patch software, drivers, and OSes with the latest updates since this helps in keeping the environment healthy from a security, performance, and future-ready perspective. Not upgrading an environment is not an option and is a fact of life. Applications should even be upgraded with enhanced functionality, bugs, and hotfixes. Every organization plans for environment and application upgrades, and, typically, these involve shutting down and restarting the application and OS. It might also involve starting the physical host OS, which, in turn, will reboot all the guest VMs running on top of it. In Microsoft Azure, you can manage, get notifications, and view the planned maintenance windows for VMs. You can find more detailed information at <https://docs.microsoft.com/en-us/azure/virtual-machines/windows/maintenance-notifications>.

Unplanned maintenance

As the name suggests, unplanned maintenance refers to maintenance that cannot be planned and is ad hoc in nature. It refers to hardware and software failures such as storage corruption, network or router failure, power loss, and a host of other failures. Bugs in the underlying platform that bring the application down are also part of unplanned maintenance.

Application deployment architecture

Application architecture plays a crucial role in ensuring the high availability of an application. An application whose components are deployed on a single machine is not highly available. When the machine reboots, the application is not available to its users. In other words, an application might have downtime if any of its architectural components do not have redundant deployments. Each component of an application should be designed so that it can be deployed on multiple machines, and redundancy should not be a bottleneck. Some software can provide features that are related to high availability and they not dependent on host OSes or other third-party tools; SQL Server availability groups are an example of such a feature.

High availability versus scalability

High availability is different from scalability, although both are serious architectural concerns. Scalability refers to the flexibility and elasticity that is required to add more resources or reduce resources to an existing deployment in order to accommodate more users than normal without compromising an application's performance.

Scalability indirectly helps in making an application highly available. However, this does not mean that scalability eventually leads to high availability. High availability is an architectural concern that is not dependent on the number of users, while scalability rules are determined by the number of users consuming the service. High availability could be a requirement even if there were very few users. Essentially, high availability is about services being present and operational as and when users demand their consumption. Therefore, it is a function of consumption based on the SLA.

High availability versus disaster recovery

High availability is again different from disaster recovery; however, the difference can be very subtle. High availability is a function of the application being in a consumable state as and when the user asks for it. So, it is designed for operations that come before a disaster, while disaster recovery is a function that comes into the picture after a disaster. Disaster recovery refers to the architecture implementation through which services are up and running after a disaster, while high availability takes care of availability prior to a disaster. Disaster recovery includes data backup and archived and dormant servers across continents, while high availability consists of load balancers, the distribution of the load, and active-passive and active-active redundancy.

Azure high availability

Achieving high availability and meeting high SLA requirements is tough. Azure provides lots of features that enable high availability for applications, from the host and guest OS to applications using its PaaS. Architects can use these features to get high availability in their applications using configuration instead of building these features from scratch or depending on third-party tools.

In this section, we will look at the features and capabilities provided by Azure to make applications highly available. Before we get into the architectural and configuration details, it is important to understand Azure's high availability-related concepts.

Concepts

The fundamental concepts provided by Azure to attain high availability are as follows:

- Availability sets
- The fault domain
- The update domain
- Availability zones

Availability sets

High availability in Azure is primarily achieved through **redundancy**. Redundancy means that there is more than one resource instance of the same type that takes control in the event of a primary resource failure. However, just having more similar resources does not make them highly available. For example, there could be multiple VMs provisioned within a subscription, but simply having multiple VMs does not make them highly available. Azure provides a resource known as an availability set, and having multiple VMs associated with it makes them highly available. A minimum of two VMs should be hosted within the availability set to make them highly available. All VMs in the availability set become highly available because they are placed on separate physical racks in the Azure data center. During updates, these VMs are updated one at a time, instead of all at the same time. Availability sets provide a fault domain and update domain to achieve this and we will discuss this more in the next section. In short, availability sets provide redundancy at a data center level, similar to locally redundant storage.

It is important to note that availability sets provide high availability within a data center. If the entire data center is down, then the availability of the application will be affected. To ensure that applications are still available even when a data center goes down, Azure has introduced a new feature known as **availability zones**, which we will learn about shortly.

The fault domain

When a VM is provisioned and assigned to an availability set, it is hosted within a fault domain. Each availability set has either two or three fault domains by default, depending on the Azure regions. Some regions provide two, while others provide three fault domains in an availability set. Fault domains are non-configurable by users. When multiple VMs are created, they are placed on separate fault domains. If the number of VMs is greater than the amount of fault domains, the additional VMs are placed on existing fault domains. For example, if there are five VMs, there will be fault domains hosted on more than one VM. Fault domains are related to physical racks in the Azure data center. Fault domains provide high availability in the case of unplanned downtime due to hardware, power, and network failure. Since each VM is placed on a different rack with different hardware, a different power supply, and a different network, other VMs continue running if this rack snaps off.

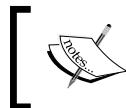
The update domain

A fault domain takes care of unplanned downtime while an update domain handles downtime from planned maintenance. Each VM is also assigned an update domain. There can be as many as 20 update domains in a single availability set. Update domains are non-configurable by users. When multiple VMs are created, they are placed on separate update domains. If more than 20 VMs are provisioned on an availability set, they are placed in a round-robin fashion on these update domains. Update domains take care of planned maintenance.

Availability zones

This is a relatively new concept introduced by Azure and is very similar to zone redundancy for storage accounts. Availability zones provide high availability within a region by placing VM instances on separate data centers within the region. Availability zones are applicable to many resources in Azure, including VMs, managed disks, VM scale sets, and load balancers. The complete list of resources that are supported by the availability zone can be found at <https://docs.microsoft.com/en-us/azure/availability-zones/az-overview#services-that-support-availability-zones>. Being unable to configure availability across zones was a gap in Azure for a long time, and it was eventually fixed with the introduction of availability zones.

Each Azure region comprises of multiple data centers. Some regions have more data centers, while others have less. These data centers within the region are known as zones. Deploying VMs in an availability zone ensures that these VMs are in different data centers and are on different racks and networks. These data centers in a region relate to high-speed networks and there is no lag in communication between these VMs.



You can find out more information about availability zones at <https://docs.microsoft.com/en-us/azure/availability-zones/az-overview.higer>.

If an application needs higher availability and wants to ensure that it is available even if an entire Azure region is down, the next rung of the ladder for availability is the Traffic Manager feature, which will be discussed later in this chapter.

Load balancing

Load balancing, as the name suggests, refers to the process of balancing a load among VMs and applications. With one VM, there is no need for a load balancer because the entire load is on a single VM and there is no other VM to share the load. However, with multiple VMs containing the same application and service, it is possible to distribute the load among them through load balancing. Azure provides a couple of resources for enabling load balancing, which are listed here:

- **Load balancers:** Azure Load Balancer helps to design solutions with high availability. Within the **Transmission Control Protocol (TCP)** stack, it is a layer 4 transport-level load balancer. This is a layer 4 load balancer that distributes incoming traffic among healthy instances of services that are defined in a load-balanced set. Level 4 load balancers work at the transport level and have network-level information, such as an IP address and port, to decide the target for the incoming request. Load balancers are discussed in more detail later in this chapter.
- **Application gateways:** Azure Application Gateway delivers high availability to your applications. An application gateway is a layer 7 load balancer that distributes incoming traffic among healthy instances of services. Level 7 load balancers can work at the application level and have application-level information such as cookies, HTTP, HTTPS, and sessions for the incoming request. Application gateways are discussed in more detail later in this chapter. Application gateways are also used when deploying Azure Kubernetes services specifically for scenarios in which ingress traffic from the internet should be routed to Kubernetes services in the cluster.

VM high availability

VMs provide compute capabilities. They provide processing power and hosting for applications and services. If an application is deployed on a single VM and that machine is down, then the application will not be available. If an application is composed of multiple tiers and each tier is deployed in its own single instance of a VM, even downtime for a single VM can render the entire application non-available. Azure tries to make even single VM instances highly available for 99.9% of the time, particularly if these VMs use premium storage for their disks.

Azure provides a higher SLA for those VMs that are grouped together in an availability set. It provides a 99.95% SLA for the availability of VMs that are part of an availability set with two or more VMs. The SLA is 99.99% if VMs are placed on availability zones.

Computing high availability

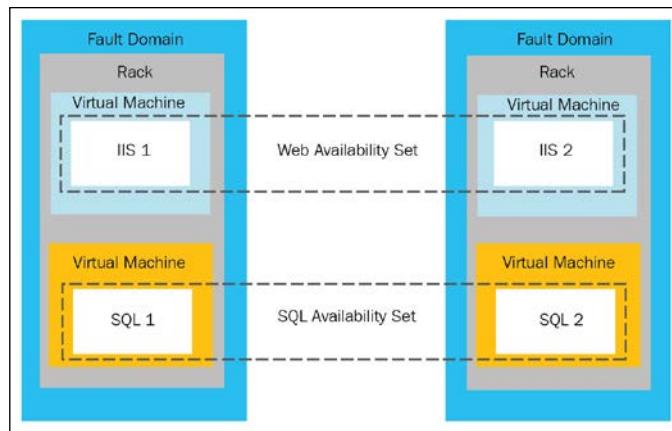
Applications demanding high availability should be deployed on multiple VMs on the same availability set. If applications are composed of multiple tiers, then each tier should have a group of VMs on their dedicated availability set. In short, if there are three tiers of an application, there should be three availability sets and a minimum of six VMs (two in each availability set) to make the entire application highly available.

So, how does Azure provide an SLA and high availability to VMs in an availability set with multiple VMs in each availability set? This is the question that might be coming to mind.

Here, the use of concepts that we considered before comes into play – that is, the fault and update domains. When Azure sees multiple VMs in an availability set, it places those VMs on a separate fault domain. In other words, these VMs are placed on separate physical racks instead of the same rack. This ensures that at least one VM continues to be available even if there is a power, hardware, or rack failure. There are two or three fault domains in an availability set and, depending on the number of VMs in an availability set, the VMs are placed on separate fault domains or repeated in a round-robin fashion. This ensures that high availability is not affected because of the failure of the rack.

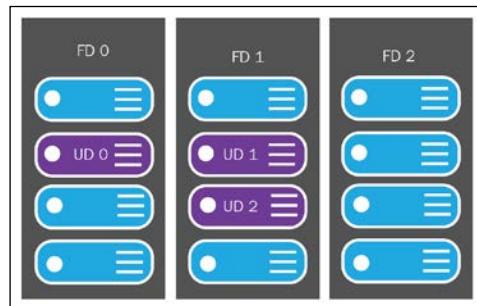
Azure also places these VMs on a separate update domain. In other words, Azure tags these VMs internally in such a way that these VMs are patched and updated one after another, such that any reboot in an update domain does not affect the availability of the application. This ensures that high availability is not impacted because of VM and host maintenance.

With the placement of VMs in separate fault and update domains, Azure ensures that not all of them are down at the same time, ensuring that at least some are alive and available for serving requests, even though they might be undergoing maintenance or facing physical downtime challenges:



The preceding diagram shows four VMs (two have **Internet Information Services (IIS)** and the other two have SQL Server installed on them). Both IIS and SQL VMs are part of their availability set. The IIS and SQL VMs are on separate fault domains and different racks in the data center. They will also be on separate upgrade domains.

The following diagram shows the relationship between fault and update domains:



Storage high availability

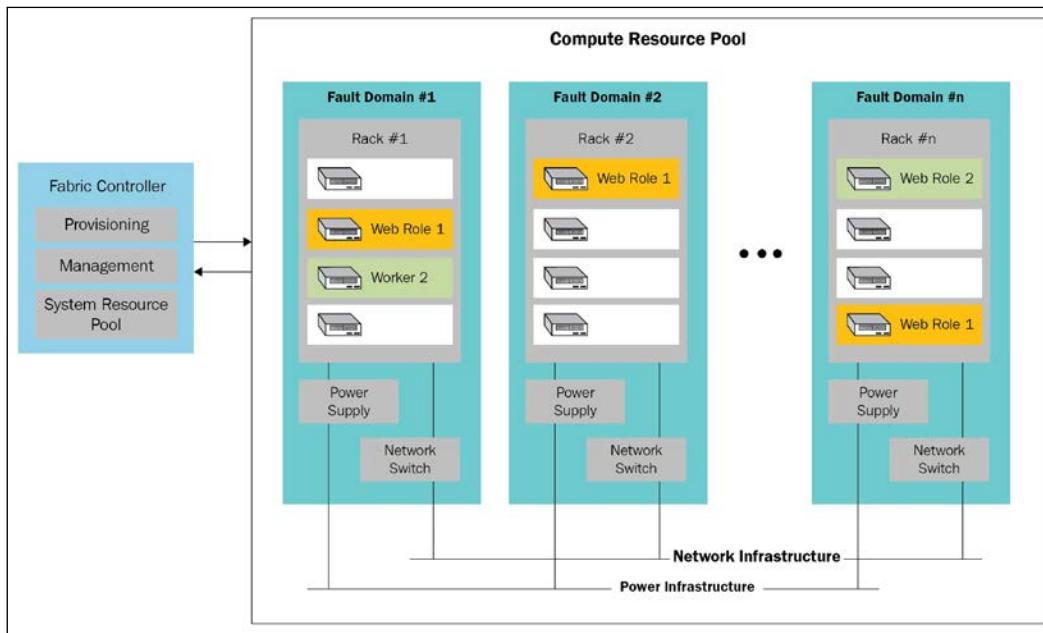
VMs are backed up by copying their **Virtual Hard Disk (VHD)** files. While availability sets provide high availability to compute instances, they do not ensure the high availability of VHD files for VMs stored in storage accounts. The VHD files for all VMs might be placed on the same storage cluster, and any cluster failure can render all the VMs non-available or less available than required. In short, it is not only compute services that need to be highly available, but even storage accounts storing the VHD files should be placed on separate clusters so that, in the event of failure, at least one or some VMs continue to be available, both from a computer and storage perspective.

Azure provides managed disks and disk management facilities. Managed disks provide better reliability for availability sets by ensuring that the disks of VMs in an availability set are sufficiently isolated from each other to avoid single points of failure. Azure does this by automatically placing the disks in different storage clusters. If a storage cluster fails due to hardware or software failure, only the VM instances with disks on those stamps fail. Each VM VHD in an availability set should be placed in a separate storage account, although VMs from different availability sets can be placed in the same storage account. Azure provides managed disks as an alternative to standard storage accounts. These managed disks place the VHD files automatically to the appropriate storage accounts internally, and users don't need to create a storage account. By creating a managed disk instead of storage accounts, users are able to delegate storage management to Azure.

PaaS high availability

Azure provides app services and cloud services for hosting managed platforms. Services and applications can be deployed on top of them. They provide flexibility, elasticity, and economies to create and deploy applications. These platforms are managed by Azure, and users do not interact with the base infrastructure on which they are deployed. They bring in a higher level of abstraction compared to **Infrastructure as a Service (IaaS)** by letting developers concentrate on their business problem and using the platform to fast-track their development and deployment process. This allows them to manage, operate, and monitor the base infrastructure. When an application is deployed in app services or cloud services, Azure provisions VMs that are not visible to users. The applications are deployed on these VMs, and the Azure Fabric Controller is responsible for provisioning, managing, and monitoring them. The Fabric Controller monitors the status of the hardware and software of the host and guest machine instances. When it detects a failure, it maintains SLAs by automatically relocating the VM instances.

When multiple cloud service role instances are deployed, Azure deploys these instances to different fault and update domains:



The previous diagram shows PaaS services with multiple VM instances deploying these web and worker roles on separate fault domains. Deploying them on separate fault domains means deploying them on separate racks within a data center. It also means that these services have separate network switches and power supplies, ensuring that even if one of the racks undergoes maintenance, or if there is a disruption in the power supply to the rack or failure of the network switch, there are other instances available to serve the customer's requests.

High-availability platforms

Azure has introduced a lot of new features in recent times with regard to high availability for PaaS. One of them is related to containers and the ecosystem surrounding them. Azure has introduced the following services:

- Containers in app services
- Azure container instance groups
- Azure Kubernetes services
- Other container orchestrators, such as DC/OS and Swarm

The other important platform that brings high availability is **Service Fabric**. Both Service Fabric and container orchestrators that include Kubernetes ensure that the desired number of application instances are always up and running in an environment. What this means is that even if one of the instances goes down in the environment, the orchestrator will know about it by means of active monitoring and will spin up a new instance on a different node, thereby maintaining the ideal number of instances. It does this without any manual or automated interference from the administrator.

While Service Fabric allows any type of application to become highly available, orchestrators such as Kubernetes, DC/OS, and Swarm are specific to containers. Also, it is important to understand that these platforms provide features that help in rolling updates, rather than a big bank update that might affect the availability of the application.

Data high availability

While VMs, app services, and containers provide high availability to compute, and managed disks provide high availability for storage, we also need to ensure that our data platforms and data are highly available.

Azure provides the following resources that make data highly available.

Azure Cosmos DB

Azure Cosmos DB is a truly global, highly available georeplicated NoSQL data store. Cosmos DB is available in almost all Azure regions, and it is also possible to configure georeplication between all of these regions. Cosmos DB allows you to create collections that are replicated across multiple regions asynchronously. It also provides flexibility in determining the consistency level, while configuring the replication strategy for high availability. These consistency levels can help an architect to determine the critical nature of the availability of data in other regions. These consistency levels are as follows:

- **Strong:** This ensures that every replicated region gets its data before returning to the user.
- **Bounded staleness:** This ensures that data is not stale in read regions beyond a certain point – that is, either a fixed number of writes or a time span.
- **Sessions:** This ensures that data is consistent in a session.

- **Ordered prefixes:** This is when the writes will come to replicated regions in a similar order as they were written in the write region.
- **Eventual:** It is possible to have dirty reads here, and there is no SLA for determining the availability of data. It follows the principle of eventual consistency.

Azure SQL replication

Azure SQL provides the replication of the database to other regions to make them highly available. Replication can be done to any region. However, an architect should choose a peer region for replication. These peer regions are a minimum of 300 miles apart and are still connected with high-speed networks. These peer regions are also patched one at a time, and so there is no risk that the patch will happen in parallel to these regions.

The data in the replicated site can be made read-available to the applications.

Azure Table storage

Azure also provides table storage, which is the key-value data that is stored in an Azure Storage account. Azure maintains three copies of the data and makes them available in times of need. The data is stored in partitions, with each partition identified using a partition key, while each row is assigned a row ID. Both the row ID and partition ID are part of the data payload. It provides storage for data without a schema, in a similar way to NoSQL data stores. In fact, NoSQL data can be stored in Azure tables easily.

A storage account can have multiple tables, and each table stores entities identified using partition and row identifiers.

Application high availability

High availability can come built-in within the software that is used for applications, otherwise it is built from the ground up within applications. One example of the high-availability feature provided by software is SQL Server's always-on availability groups. They help in keeping databases highly available.

Azure services also have a built-in high-availability mechanism. In Azure SQL, data is replicated synchronously within the region. Active georeplication allows up to four additional database copies in the same region or different regions. Azure storage has its own mechanism to make data available by replicating it to multiple data centers and regions.

Load balancers in Azure

Azure provides two resources that have the functionality of a load balancer. It provides a level 4 load balancer that works at the transport layer within the TCP OSI stack, and a level 7 load balancer (application gateway) that works at the application and session level.

Although both application gateways and load balancers provide the basic features of balancing a load, they serve different purposes. There are a number of use cases in which it makes more sense to deploy an application gateway rather than a load balancer.

An application gateway provides the following features that are not available with the Azure Load Balancers:

- **Web application firewall:** This is an additional firewall on top of the OS firewall and has the capability to peek into incoming messages. This helps in identifying and preventing common web-based attacks such as SQL injection, cross-site scripting attacks, and session hijacks.
- **Cookie-based session affinity:** Load balancers distribute incoming traffic to service instances that are healthy and relatively free. A request can be served by any service instance. However, there are applications that need advance features in which all subsequent requests following the first request should be processed by the same service instance. This is known as cookie-based session affinity. An application gateway provides cookie-based session affinity to keep a user session on the same service instance using cookies.
- **Secure Sockets Layer (SSL) offload:** The encryption and decryption of request and response data is performed by the SSL and is generally a costly operation. Web servers should ideally be spending resources on processing and serving requests, rather than the encryption and decryption of traffic. SSL offload helps in transferring this cryptography process from the web server to the load balancer, thereby providing more resources to web servers serving users. The request from the user is encrypted, but gets decrypted at the application gateway instead of the web server. The request from the application gateway to the web server is unencrypted.
- **End-to-end SSL:** While SSL offload is a nice feature for a certain application, there are certain mission-critical secure applications that need complete SSL encryption and decryption even if traffic passes through load balancers. An application gateway can be configured for end-to-end SSL cryptography as well.

- **URL-based content routing:** Application gateways are also useful for redirecting traffic to different servers based on the URL content of incoming requests. This helps in hosting multiple services alongside other applications.

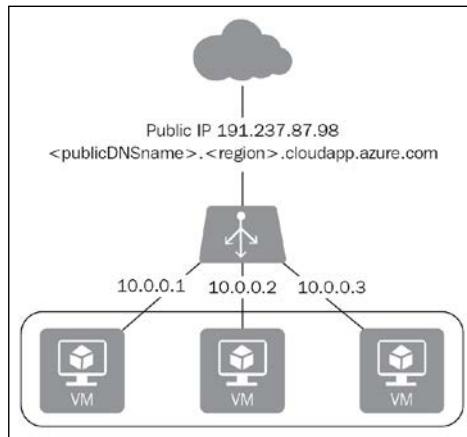
Azure load balancers

An Azure Load Balancer distributes incoming traffic based on the transport-level information that is available to it. It relies on the following features:

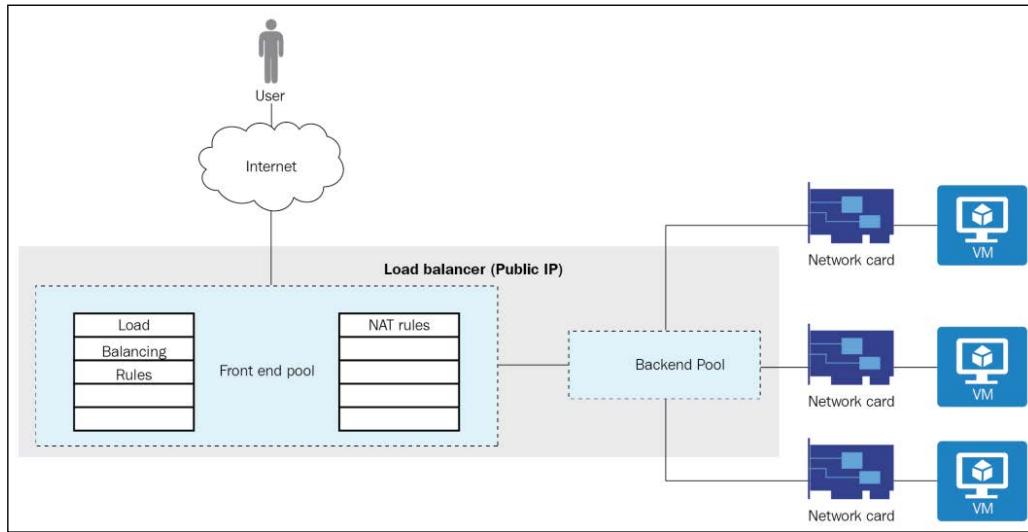
- An originating IP address
- A target IP address
- An originating port number
- A target port number
- A type of protocol – either TCP or HTTP

Public load balancing

In a public load balancing configuration, load balancers are assigned a public IP address. Assigning a public IP address ensures that the load balancer can accept requests coming in from the internet. Without a public IP address, it is not possible to access a resource from the internet. A load balancer can be configured with load-balancing rules. Load-balancing rules work at the port level. It accepts a source and destination ports and maps them together such that whenever a load balancer receives a request for the source port, the request is forwarded to a VM from a group of VMs attached to the load balancer on the destination port. This is shown in the following diagram:



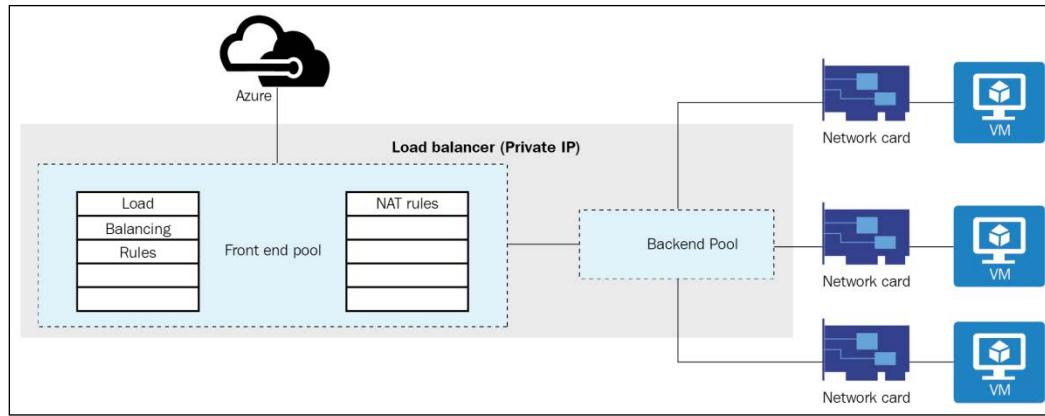
But how does this entire thing work? How is a public IP address assigned to a load balancer? What does the load balancer contain? How is it configured with load balancer rules? How does the load balancer send requests to the VMs? How does the VM know that it is attached to the load balancer? The answers to all of these questions are visible in the following diagram:



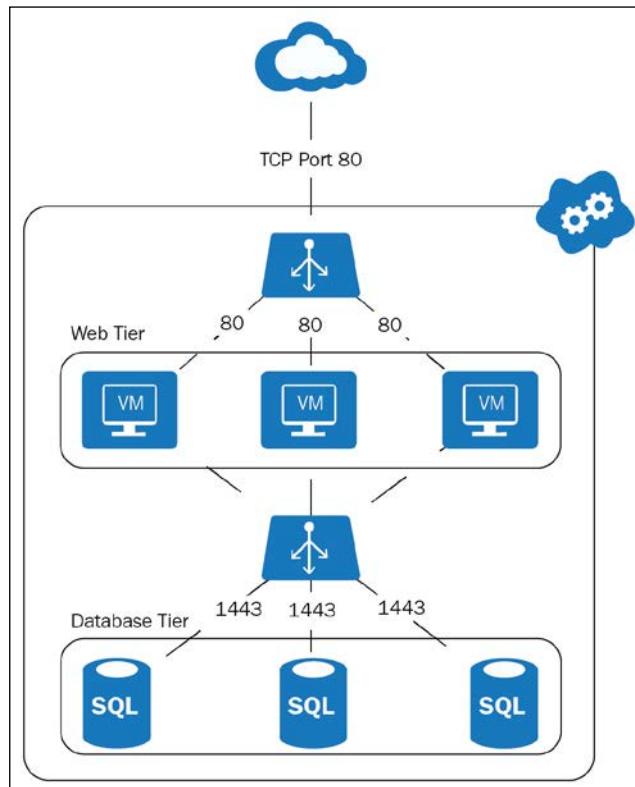
In this configuration, the load balancer is assigned a public IP address. The load balancer is accessible from the internet and can accept client requests. The load balancer can be configured with load-balancing and **Network Address Translation (NAT)** rules. Both NAT and load-balancing rules are part of the frontend configuration. The frontend configuration sends client requests to one of the IP addresses available in the backend pool. These IP addresses are assigned to the network interface that is associated to the VMs.

Internal load balancing

The following diagram shows the workings of an internal load balancer. You can see that the request comes from resources in Azure itself, since it is not accessible on the internet. In this configuration, the load balancer is assigned a private IP address. The load balancer is only accessible within the virtual network to which it is attached. It cannot be accessed through the internet. The remainder of its configuration is similar to a public load balancer. The load balancer can be configured with load-balancing and NAT rules:



The following diagram shows how multiple load balancers can be deployed to create solutions. In this way, there is a public load balancer that accepts client requests and an internal load balancer for the database tier. The database-tier VMs are not accessible on the internet, but only through the load balancer on port 1433:



Port forwarding

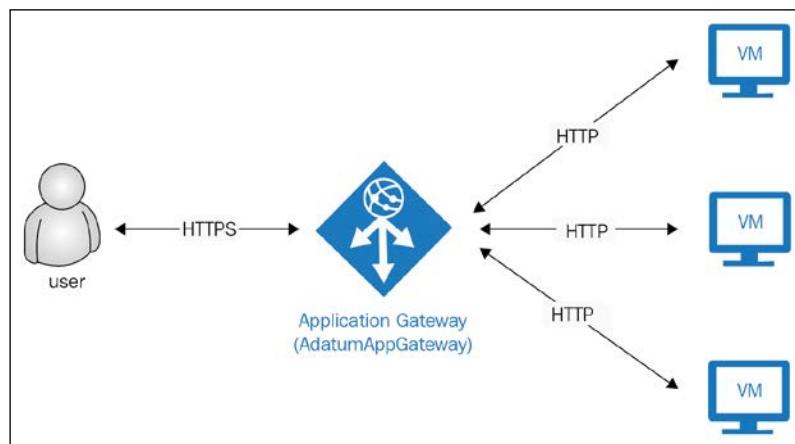
At times, there is a need for a request to always redirect to a VM. An Azure Load Balancer helps us to achieve this with NAT rules. NAT rules are evaluated after load-balancing rules are evaluated and found not to be satisfied. NAT rules are evaluated for each incoming request and, once it finds them, it forwards the request to that VM through a backend pool. It should be noted that a VM cannot register the same port for both port forwarding using NAT rules and load-balancing rules.

Azure Application Gateway

Azure Load Balancer helps us to enable solutions at the infrastructure level. However, there are times when advanced services and features are required using a load balancer. These advanced services include SSL termination, sticky sessions, advanced security, and more. An Azure Application Gateway provides these additional features; an Azure Application Gateway is a level 7 load balancer that works with the application and session payload in a TCP OSI stack.

Application gateways have more information than Azure load balancers to make decisions on request routing and load balancing between servers. Application gateways are managed by Azure and are highly available.

An application gateway sits in between the users and VMs, as shown in the following diagram:



Application gateways are a managed service. They use **Application Request Routing (ARR)** to route requests to different services and endpoints. Creating an application gateway requires a private or public IP address. The application gateway then routes the HTTP/HTTPS traffic to configured endpoints.

An application gateway is similar to an Azure load balancer from a configuration perspective, with additional constructs and features. Application Gateway can be configured with frontend IP address, certificate, port configuration, backend pool, session affinity, and protocol information.

Azure Traffic Manager

After gaining a good understanding of both Azure Load Balancer and Application Gateway, it's time to get into the details of Traffic Manager. Azure load balancers and application gateways are much-needed resources for high availability within a data center and region; however, to achieve high availability across regions and data centers, there is a need for another resource, and Traffic Manager helps us in this pursuit.

Traffic Manager helps us to create highly available solutions that span multiple geographies, regions, and data centers. Traffic Manager is not similar to load balancers. It uses **Domain Name Service (DNS)** to redirect requests to an appropriate endpoint determined by their health and configuration. Traffic Manager is not a proxy or a gateway, and it does not see the traffic passing between the client and the service. It simply redirects requests based on the most appropriate endpoints.

Azure Traffic Manager enables you to control the distribution of traffic across your application endpoints. An endpoint is any internet-facing service hosted inside or outside of Azure.

Endpoints are internet-facing, reachable public URLs. Applications are provisioned within multiple geographies and Azure regions. Applications deployed to each region have a unique endpoint referred by **DNS CNAME**. These endpoints are mapped to the Traffic Manager endpoint. When a Traffic Manager is provisioned, it gets an endpoint by default with a `.trafficmanager.net` URL extension.

When a request arrives at the Traffic Manager URL, it finds the most appropriate endpoint out of its list and redirects the request to it. In short, Azure Traffic Manager acts as a global DNS to identify the region that will serve the request.

However, how does Traffic Manager know which endpoints to use and redirect the client request to? There are two aspects that Traffic Manager implements to determine the most appropriate endpoint and region.

First, Traffic Manager actively monitors the health of all endpoints. It can monitor the health of VMs, cloud services, and app services. If it determines that the health of an application deployed to a region is not suitable for redirecting traffic, it redirects the requests to a healthy endpoint.

Second, Traffic Manager can be configured with routing information. There are six traffic routing methods available in Traffic Manager, which are as follows:

- **Priority:** This should be used when all traffic should go to a default endpoint, and backups are available in case the primary endpoints are unavailable.
- **Weighted:** This should be used to distribute traffic across endpoints evenly, or according to defined weights.
- **Performance:** This should be used for endpoints in different regions, and users should be redirected to the closest endpoint based on their location. This has a direct impact on network latency.
- **Geographic:** This should be used to redirect users from a specific geography to an endpoint (that is, Azure, external, or nested) available in that geography or nearest to that geography. Examples include complying with data sovereignty mandates, localization of content and user experience, and measuring traffic from different regions.
- **Subnet:** This is a new routing method added and it helps in providing clients with different endpoints based on their IP addresses. In this method, a range of IP addresses are assigned to each endpoint. These IP address ranges are mapped to the client IP address to determine an appropriate returning endpoint. Using this routing method, it is possible to provide different content to different people based on their originating IP address.
- **Multivalue:** This is also a new method added in Azure. In this method, multiple endpoints are returned to the client and any of them can be used. This ensures that if one endpoint is unhealthy, then other endpoints can be used instead. This helps in increasing the overall availability of the solution.

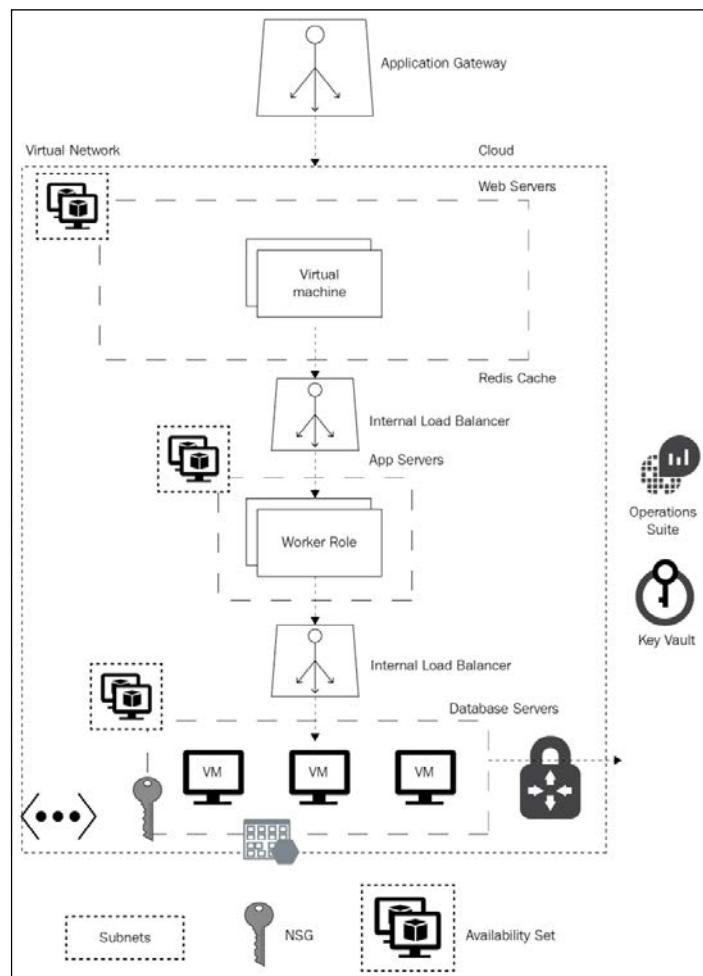
It should be noted that after Traffic Manager determines a valid healthy endpoint, clients connect directly to the application.

Architectural considerations for high availability

Azure provides high availability for various means and at various levels. High availability can be at data center level, region level, or across Azure. In this section, we will go through some of the architectures for high availability.

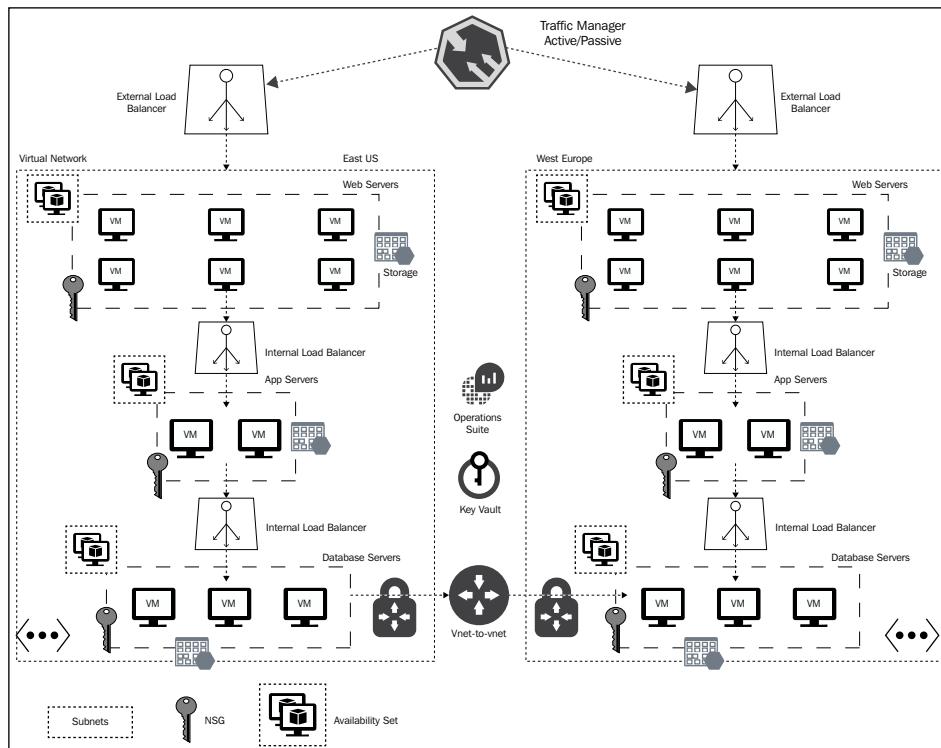
High availability within Azure regions

The architecture shown in the following diagram shows a high-availability deployment within a single Azure region. High availability is designed at the individual resource level. In this architecture, there are multiple VMs at each tier connected through either an application gateway or load balancer, and they are part of an availability set. Each tier is associated with an availability set. These VMs are placed on separate fault and update domains. While the web servers are connected to application gateways, the rest of the tiers, such as application and database tiers, have internal load balancers:



High availability across Azure regions

This architecture shows similar deployments on two different Azure regions. As shown in the following diagram, both regions have the same resources deployed. High availability is designed at the individual resource level within these regions. There are multiple VMs at each tier, connected through the load balancer, and they are part of the availability set. These VMs are placed on separate fault and update domains. While the web servers are connected to external load balancers, the rest of the tiers, such as application and database tiers, have internal load balancers. It should be noted that application load balancers can be used for web servers and application tiers instead of Azure load balancers if there is a need for advanced services, such as session affinity, SSL termination, advanced security using a **web application firewall (WAF)**, and path-based routing. Databases in both the regions are connected to each other using virtual network peering and gateways. This is helpful in configuring log shipping, SQL Server AlwaysOn, and other data synchronization techniques. The endpoints of load balancers from both of regions are used to configure Traffic Manager endpoints, and traffic is routed based on the priority load balancing method. Traffic Manager helps in routing all requests to the East US region and, after failover, to West Europe in the case of the non-availability of the first region:



Best practices

This section describes high-availability best practices. They have been categorized into application, deployment, data management, and monitoring.

Application high availability

An application should be built with high availability as one of the most important architectural concerns. Some of the important application-related, high-availability practices are outlined here:

- An application should implement appropriate exception handling to gracefully recover and inform stakeholders about the exception.
- An application should try to perform the same operation again for a fixed interval and a certain number of times before exiting in the event of an error or exception.
- An application should have the built-in timeout capability to decide whether an exception cannot be recovered from.
- Maintaining logs and writing logs for all errors, exceptions, and executions should be adopted within the application.
- Applications should be profiled to find their actual resource requirements in terms of compute, memory, and network bandwidth for a different number of users.

Please refer to <https://docs.microsoft.com/en-us/azure/architecture/checklist/availability> in order to learn more about applications and other high-availability best practices.

Deployment

A deployment strategy, to a large extent, affects the availability of the application and overall environment. Here are a number of strategies that you can deploy:

- Deploy multiple instances of Azure resources, including multiple instances for VMs, cloud services, and other resources.
- Deploy VMs on availability sets or availability zones – note that they cannot be used together.
- Deploy multiple instances of VMs across multiple regions.
- Create multiple environments and keep at least one of them in standby mode.

Data management

Some of the important data-related best practices for high availability include the following:

- If possible, store data on Azure-provided services, such as Azure SQL, Cosmos DB, and Table storage.
- Use storage accounts that are based on the georedundant type.
- Ensure that data is replicated to multiple regions and not only within a zone or data center.
- Take periodic backups and conduct restore tests frequently.
- If storing data in VMs, ensure that there are multiple VMs and that they are either on availability sets or availability zones.
- Use keys and secrets to data stored in Azure Key Vault.

Monitoring

Some of the important monitoring-related best practices for high availability are as follows:

- Use log analytics service to capture logs from the environment and enable auditing.
- Use application insights to capture telemetry information from the custom application and environment related to compute, storage and networks, and other log information.
- Ensure alerts are configured on OMS (log analytics) for issues related to the availability of the environment and application.

Scalability

Running applications and systems that are available to users for consumption is important for architects of any serious application. However, there is another equally important application feature that is one of the top priorities for architects, and this is the scalability of the application.

Imagine a situation in which an application is deployed and achieves great performance and availability with a few users, but both availability and performance decrease as the number of users begins to increase. There are times when an application under a normal load performs well, but decreases in performance with the increase in the number of users. This can happen if there is a sudden increase in the number of users and the environment is not built for such a large number of users.

To accommodate such spikes in the number of users, you might provision the hardware and bandwidth for handling spikes. The challenge with this is that the additional capacity is not used for a majority of the year, and so does not provide any return on investment. It is provisioned for use only during the holiday season or sales. I hope that by now you are becoming familiar with the problems that architects are trying to solve. All these problems are related to capacity sizing and the scalability of an application. The focus of this chapter is to understand scalability as an architectural concern and to check out the services that are provided by Azure for implementing scalability.

Capacity planning and sizing are a couple of the top priorities for architects for their applications and services. Architects must find a balance between buying and provisioning too many resources versus too few resources. Having fewer resources can lead to you not being able to serve all users, resulting in them turning to the competition. On the other hand, having more resources can hurt your budget and return on investment because most of the resources remain unused most of the time. Moreover, the problem is amplified with a varied level of demand during different times. It is almost impossible to predict the number of users for the application round the clock. However, it is possible to find an approximate number using past information and continuous monitoring.

Scalability can be defined as follows:

"Scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged to accommodate that growth. For example, a system is considered scalable if it is capable of increasing its total output under an increased load when resources (typically hardware) are added."

Scalability refers to the ability to handle a growing number of users and provide them with the same level of performance when there are fewer users in application deployment, processes, and technology. Scalability might refer to serving more requests without a decrease in performance, or it might refer to handling larger and more time-consuming work without any loss of performance in both cases.

Capacity planning and sizing exercises should be undertaken by architects at the very beginning of the project and during the planning phase to provide scalability to applications.

Some applications have stable demand patterns, while it is difficult to predict others. Scalability requirements are known for stable demand applications, while it is a more involved process for variable demand applications. Autoscaling, a concept we will review in the next section, should be used for such applications whose demands cannot be predicted.

Scalability versus performance

It is quite easy to get confused between scalability and performance architectural concerns, because scalability is all about ensuring that, no matter the number of users consuming the application, all receive the same predetermined level of performance.

Performance relates to application features that ensure that the application caters to predefined response times and throughput. Scalability refers to having provisions for more resources when needed in order to accommodate more users without sacrificing performance.

It is better to understand this using an analogy: the speed of a train determines the performance of a railway system. However, enabling more trains to run in parallel at the same or higher speed demonstrates the railways system's scalability.

Azure scalability

In this section, we will look at the features and capabilities provided by Azure to make applications highly scalable. Before we get into the architecture and configuration details, it is important to understand Azure's high-scalability concepts.

Concepts

The fundamental constructs provided by Azure to attain high availability are as follows:

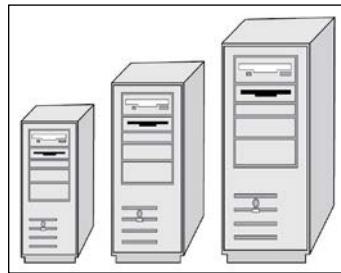
- Scaling
- Scaling up and down
- Scaling out and in
- Auto scaling
- Rolling updates

Scaling

Scaling refers to any transformation that either increases or decreases the units of resources that are used to serve requests from users. Scaling can be automatic or manual. Manual scaling requires an administrator to manually initiate the scaling process, while automatic scaling refers to an automatic increase or decrease in resources based on the events available from the environment and ecosystem, such as memory and CPU availability. Scaling can be effected up or down, or out and in, which will be explained later in this section.

Scaling up

Scaling up a VM or service refers to the adding of additional resources to existing servers, such as CPU, memory, and disks. Its aim is to increase the capacity of existing physical hardware and resources:

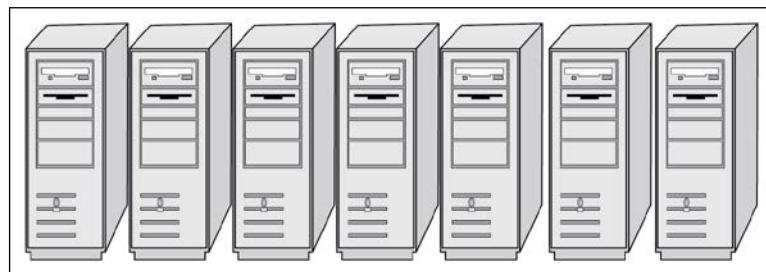


Scaling down

Scaling down a VM or service refers to the removal of existing resources from existing servers, such as CPU, memory, and disks. Its aim is to decrease the capacity of existing physical and virtual hardware and resources.

Scaling out

Scaling out refers to the process of adding additional hardware such as additional servers and capacity. This typically involves adding new servers, assigning them IP addresses, deploying applications on them, and making them part of the existing load balancers such that traffic can be routed to them. Scaling out can be automatic or manual as well. However, for better results, automation should be used:



Scaling in

Scaling in refers to the process of removing the existing hardware in terms of existing servers and capacity. This typically involves removing existing servers, deallocated their IP addresses, and removing them from the existing load balancer configuration such that traffic cannot be routed to them. In the same way as scaling out, scaling in can be automatic or manual.

Auto scaling

Auto scaling refers to the process of either scaling up/down or scaling out/in dynamically based on application demand, and this happens using automation. Auto scaling is useful because it ensures that deployment always consists of a correct and ideal number of server instances. Auto scaling helps in building applications that are fault tolerant. It not only helps in scalability, but also makes applications highly available. Finally, it provides the best cost management. Auto scaling helps in having the optimal configuration for server instances based on demand. It helps in not over-provisioning servers that are underutilized and removes servers that are no longer required after scaling out.

PaaS scalability

Azure provides app services for hosting managed applications. App services are a PaaS offering from Azure. They provide services to the web and mobile platforms. Behind the web and mobile platforms is a managed infrastructure that is managed by Azure on behalf of its users. Users do not see or manage the infrastructure; however, they have the capability to extend the platform and deploy their applications on top of it. In doing so, architects and developers can concentrate on their business problems instead of worrying about the base platform and infrastructure provisioning, configuration, and troubleshooting. Developers have the flexibility to choose any language, OS, and framework to develop their applications.

App services provide multiple plans and, based on the plans chosen, various capabilities of scalability are available. App services provide the following five plans:

- **Free:** This uses shared infrastructure. It means that multiple applications will be deployed on the same infrastructure from the same or multiple tenants. It provides 1 GB of storage free of cost. However, there is no scaling facility available in this plan.
- **Shared:** This also uses shared infrastructure and provides 1 GB of storage free of cost. Additionally, custom domains are also provided as an extra feature. However, there is no scaling facility available in this plan.
- **Basic:** This has three different **Stock Keeping Units (SKUs)** – B1, B2, and B3. They have increasing units of resources available to them in terms of CPU and memory. In short, they provide improved configuration of the VMs backing these services. Additionally, they provide storage, custom domains, and SSL support. The basic plan provides basic features for manual scaling. There is no automatic scaling available in this plan. A maximum of three instances can be used for scaling out the application.
- **Standard:** This also has three different SKUs – S1, S2, and S3. They have increasing units of resources available to them in terms of CPU and memory. In short, they provide improved configuration of the VMs backing these services. Additionally, they provide storage, custom domains, and SSL support that is similar to the basic plan. This plan also provides a traffic manager, staging slots, and one daily backup as an additional feature on top of the basic plan. The standard plan provides features for automatic scaling. A maximum of 10 instances can be used for scaling out the application.
- **Premium:** This also has three different SKUs – P1, P2, and P3. They have increasing units of resources available to them in terms of CPU and memory. In short, they provide improved configuration of the VMs backing these services. Additionally, they provide storage, custom domains, and SSL support that is similar to the basic plan. This plan also provides a traffic manager, staging slots, and 50 daily backups as an additional feature on top of the basic plan. The standard plan provides features for automatic scaling. A maximum of 20 instances can be used for scaling out the application.

PaaS – scaling up and down

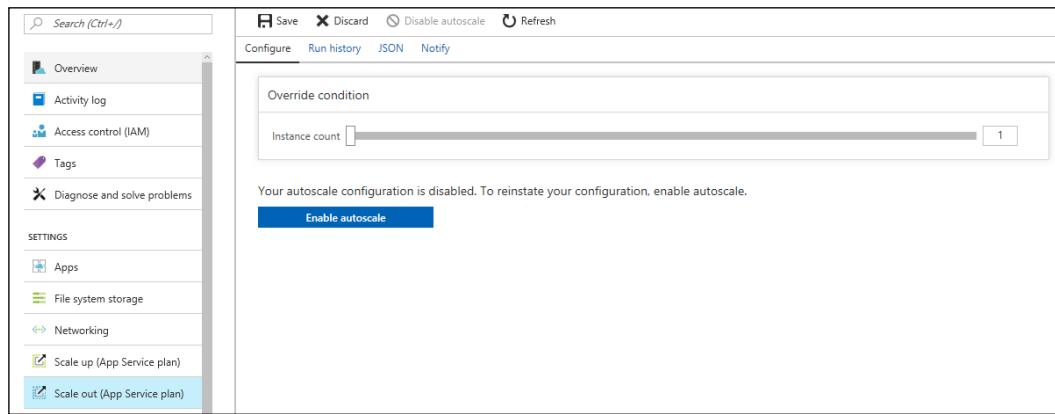
Scaling up and down services hosted in app services is quite simple. The Azure app services menu items to scale up, which opens a new blade with all plans and their SKUs listed. Choosing a plan and SKU will scale the service up or down, as shown in the following screenshot:

P1 Premium			P2 Premium			P3 Premium		
1 Core	2 Core	4 Core						
1.75 GB RAM	3.5 GB RAM	7 GB RAM						
250 GB Storage Custom domains / SSL SNI Incl & IP SSL Support Up to 20 instance(s) * Subject to availability 20 slots Web app staging 50 times daily Backup Traffic Manager Geo availability	250 GB Storage Custom domains / SSL SNI Incl & IP SSL Support Up to 20 instance(s) * Subject to availability 20 slots Web app staging 50 times daily Backup Traffic Manager Geo availability	250 GB Storage Custom domains / SSL SNI Incl & IP SSL Support Up to 20 instance(s) * Subject to availability 20 slots Web app staging 50 times daily Backup Traffic Manager Geo availability						
14,752.68 INR/MONTH (ESTIMATED)	29,505.37 INR/MONTH (ESTIMATED)	59,010.73 INR/MONTH (ESTIMATED)						
S1 Standard			S2 Standard			S3 Standard		
1 Core	2 Core	4 Core						
1.75 GB RAM	3.5 GB RAM	7 GB RAM						
50 GB Storage Custom domains / SSL SNI Incl & IP SSL Support Up to 10 instance(s) Auto scale Daily Backup 5 slots Web app staging Traffic Manager Geo availability	50 GB Storage Custom domains / SSL SNI Incl & IP SSL Support Up to 10 instance(s) Auto scale Daily Backup 5 slots Web app staging Traffic Manager Geo availability	50 GB Storage Custom domains / SSL SNI Incl & IP SSL Support Up to 10 instance(s) Auto scale Daily Backup 5 slots Web app staging Traffic Manager Geo availability						
Select								

PaaS – scaling out and in

Scaling out and in services hosted in app services is also quite simple. The Azure app services menu items to scale out, which opens a new blade with scaling configuration options.

By default, autoscaling is disabled for both premium and standard plans. It can be enabled using the **Scale Out** menu item and by clicking on the **Enable autoscale** button, as shown in the following screenshot:



Manual scaling does not require configuration but auto scaling helps in configuring with the aid of the following properties:

- **Mode of scaling:** This is based on either performance metric such as CPU or memory usage or users can simply specify count of instances for scaling.
- **When to scale:** Multiple rules can be added that determine when to scale out and in. Each rule can determine the criteria such as CPU or memory consumption, whether to increase or decrease instances, and how many instances to increase or decrease at a time. At least one rule for scale out and one rule for scale in should be configured. Threshold definitions help in defining the upper and lower limits that should trigger the auto scale - by either increasing or decreasing the number of instances.

Azure Solution Availability and Scalability

- **How to scale:** This specifies how many instances to create or remove in each scale-out or scale-in operation:

The screenshot shows the Azure portal interface for configuring an App Service plan's scaling settings. The left sidebar has 'Scale up (App Service plan)' selected under 'Scale out (App Service plan)'. The main panel shows the 'Configure' tab for an autoscale setting named 'destRG'. Under 'Default', it's set to 'Auto created scale condition'. In the 'Criteria' section, a rule is defined: 'Metric name: CPU Percentage', 'Time aggregation: Average', 'Time grain statistic: Average', 'Operator: Greater than', 'Threshold: 70', and 'Duration (in minutes): 10'. Under 'Action', the 'Operation' is set to 'Increase count by 1', and 'Cool down (minutes)' is set to 5. There are also sections for 'Rules' and 'Schedule'.

This is quite a good feature to enable in any deployment. However, readers should enable both scaling out and scaling in together to ensure that the environment is back to normal capacity after scaling out.

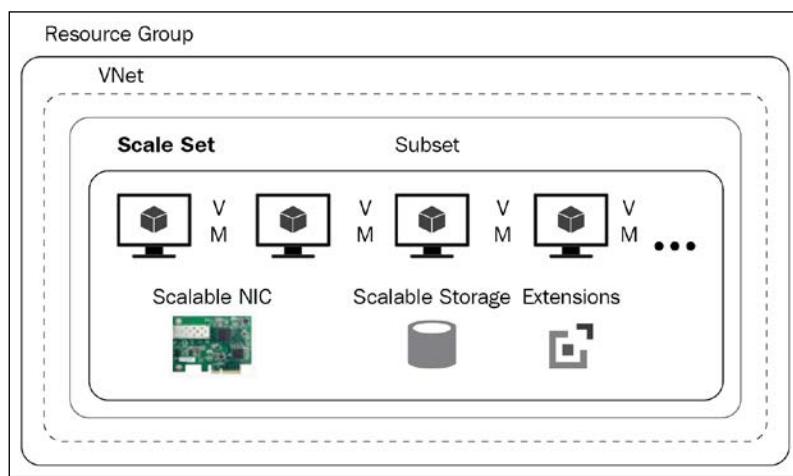
IaaS scalability

There are users who want to have complete control over the base infrastructure, platform, and application. They prefer to consume IaaS solutions compared to PaaS solutions. For such customers, when they create VMs, they are also responsible for capacity sizing and scaling. There is no out-of-the-box configuration for manually scaling or auto scaling VMs. These customers will have to write their own automation scripts, triggers, and rules to achieve auto scaling. With VMs comes the responsibility of maintaining them as well. The patching, updating, and upgrading of VMs is the responsibility of owners. Architects should think about both planned and unplanned maintenance. How these VMs should be patched, the order, grouping, and other factors must be thought through to ensure that both the scalability and availability of an application is not compromised. To help alleviate such problems, Azure provides **VM Scale Sets (VMSS)** as a solution.

VMSS

VMSS is an Azure compute resource that you can use to deploy and manage a set of identical VMs. With all VMs configured in the same way, scale sets are designed to support true auto scaling, and no preprovisioning of VMs is required. It helps in provisioning multiple identical VMs connected to each other through a virtual network and subnet.

VMSS consists of multiple VMs, but they are managed at the VMSS level. All VMs are part of this unit and any changes made are applied to the unit, which, in turn, applies it to those VMs that are using a predetermined algorithm:



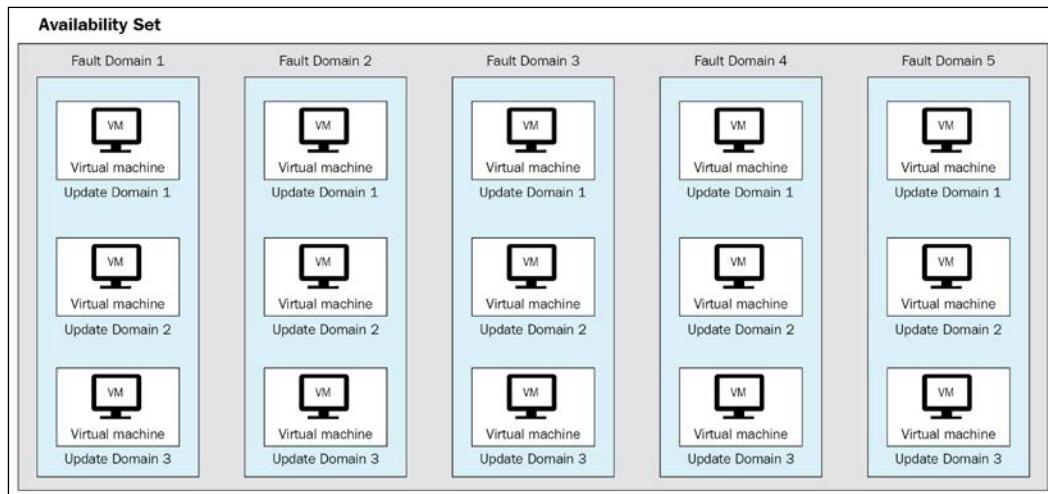
This allows these VMs to be load balanced using Azure Load Balancer or Application Gateway. All the VMs could be either Windows or Linux OSes. They can run automated scripts using a PowerShell extension and they can be managed centrally using a desired state configuration. They can be monitored as a unit, or individually using Log Analytics.

VMSS can be provisioned from the Azure portal, Azure **command-line interface (CLI)**, Azure Resource Manager templates, REST APIs, and PowerShell cmdlets. It is possible to invoke REST APIs and Azure CLI from any platform, environment, and OS, and in any language.

Many of Azure's services already use VMSS as their underlying architecture. Among them are Azure Batch, Azure Service Fabric, and Azure Container Services. Azure Container Services, in turn, provisions Kubernetes and DC/OS on VMSS.

VMSS architecture

VMSS allows for the creation of up to 1,000 VMs in a scale set when using a platform, images, and 100 VMs if using a custom image. If the amount of VMs is less than 100 in a scale set, they are placed in a single availability set; however, if they are more than 100, multiple availability sets are created (known as placement groups), and VMs are distributed among these availability sets. We know from Chapter 1, *Getting Started*, that VMs in an availability set are placed on separate fault and update domains. Availability sets related to VMSS have five fault and update domains by default. VMSS provides a model that holds metadata information for the entire set. Changing this model and applying changes impacts all VM instances. This metadata information includes maximum and minimum VM instances, the OS SKU and version, the current number of VMs, fault and update domains, and more. This is demonstrated in the following diagram:



VMSS scaling

Scaling refers to an increase or decrease in compute and storage resources. VMSS is a feature-rich resource that makes scaling easy and efficient. It provides auto scaling, which helps in scaling up or down based on external events and data such as CPU and memory usage. Some of the VMSS scaling features are explained in this section.

Horizontal versus vertical scaling

Scaling can be horizontal or vertical, or both. Horizontal scaling is another name for scaling out and in, while vertical scaling refers to scaling up and down.

Capacity

VMSS have a capacity property that determines the number of VMs in a scale set. VMSS can be deployed with zero as a value for this property. It will not create a single VM; however, if you provision VMSS by providing a number for the capacity property, that number of VMs is created.

Auto scaling

Automatic scaling of VMs in VMSS refers to the addition or removal of VM instances based on configured environments to meet the performance and scalability demands of an application. Generally, in the absence of VMSS, this is achieved using automation scripts and runbooks.

VMSS helps in this automation process with the help of configuration. Instead of writing scripts, VMSS can be configured for automated scaling up and down.

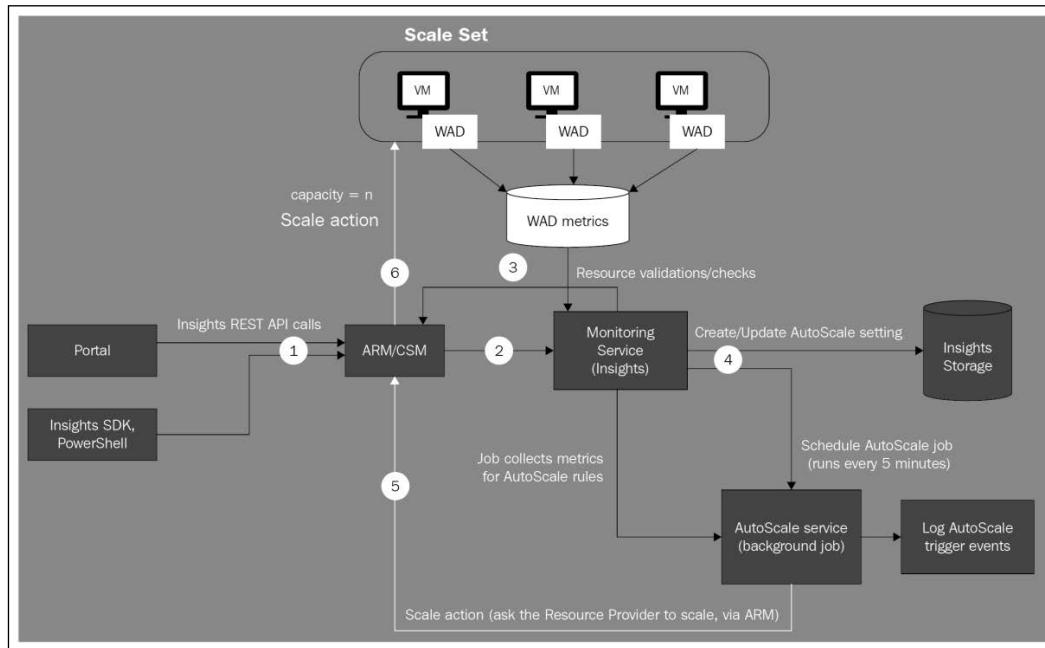
Auto scaling consists of multiple integrated components to achieve its end goal. Auto scaling continuously monitors VMs and collects telemetry data about them. It stores this data, combines it, and then evaluates it against a set of rules to determine whether it should trigger the auto scale. The trigger could be to scale out or scale in. It could also be to scale up or down.

Autoscale uses diagnostic logs for collecting telemetry data from VMs. These logs are stored in storage accounts as diagnostic metrics. Autoscale also uses the insight monitoring service, which reads these metrics, combines them together, and stores them in their own storage account.

Autoscale background jobs run continually to read the insights' storage data, evaluate them based on all the rules configured for auto scaling, and, if any of the rules or combination of rules return positive, execute the process of auto scaling. The rules can take into consideration the metrics from guest VMs and the host server.

The rules defined using the property descriptions are available at <https://docs.microsoft.com/en-us/azure/virtual-machine-scale-sets/virtual-machine-scale-sets-autoscale-overview>.

The autoscale architecture is shown in the following diagram:



Autoscale can be configured for scenarios that are more complex than general metrics available from environments. For example, scaling could be based on any of the following:

- Scaling on a specific day
- Scaling on a recurring schedule such as weekends
- Scaling differently on weekdays and weekends
- Scaling during holidays, that is, one of the events
- Scaling on multiple resource metrics

These can be configured using the `schedule` property of insight resources that helps in registering rules.

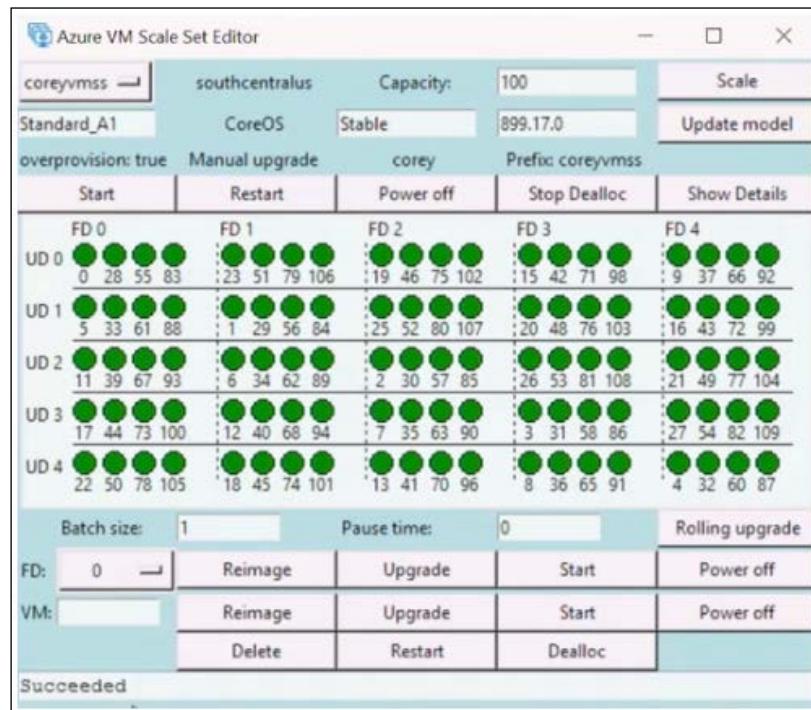
Architects should ensure that at least two actions – scale out and scale in – are configured together. Scaling in or scaling out configuration will not help in achieving the scaling benefits provided by VMSS.

Upgrades and maintenance

After VMSS and applications are deployed, they need to be actively maintained. Planned maintenance should be conducted periodically to ensure that both the environment and application are up to date with the latest features, and the environment is current from a security and resilience point of view.

Upgrades can be associated with applications, the guest VM instance, or the image itself. Upgrades can be quite complex because they should happen without affecting the availability, scalability, and performance of environments and applications. To ensure that updates can take place one instance at a time using rolling upgrade methods, it is important that VMSS supports and provides capabilities for these advanced scenarios.

There is a utility provided by the Azure team to manage updates for VMSS. It's a Python-based utility that can be downloaded at <https://github.com/gbowering/vmssdashboard>. It makes REST API calls to Azure to manage scale sets. This utility can be used for starting, stopping, upgrading, and re-imaging VMs on a fault domain or group of VMs, as shown in the following screenshot:



Application updates

Application updates in VMSS should not be executed manually. They must be executed as part of the release management and pipelines that are using automation. Moreover, the update should happen one application instance at a time, and not affect the overall availability and scalability of the application. Configuration management tools, such as the desired state configuration, should be deployed to manage application updates. The DSC pull server can be configured with the latest version of the application configuration and they should be applied on a rolling basis to each instance.

Guest updates

Updates to VM are the responsibility of the administrator. Azure is not responsible for patching guest VMs. Guest updates are in preview mode and users should control patching manually or use custom automation such as runbooks and scripts. However, rolling patch upgrades are in preview mode and can be configured in the Azure Resource Manager template using an upgrade policy, as follows:

```
"upgradePolicy": {  
    "mode": "Rolling",  
    "automaticOSUpgrade": "true" or "false",  
    "rollingUpgradePolicy": {  
        "batchInstancePercent": 20,  
        "maxUnhealthyUpgradedInstanceCount": 0,  
        "pauseTimeBetweenBatches": "PT0S"  
    }  
}
```

Image updates

VMSS can update the OS version without any downtime. OS updates involve changing the version or SKU of the OS or changing the URI of a custom image. Updating without downtime means updating VMs one at a time or in groups (such as one fault domain at a time) rather than all at once. By doing so, any VMs that are not being upgraded can keep running.

Best practices of scaling provided by VMSS

In this section, we will go through some of the best practices that applications should implement to take advantage of the scalability capability provided by VMSS.

The preference for scaling out

Scaling out is a better scaling solution compared to scaling up. Scaling up or down means resizing VM instances. When a VM is resized, it generally needs to be restarted, which has its own disadvantages. First, there is downtime for the machine. Second, if there are active users connected to the application on that instance, they might face the unavailability of the application, or they might even have lost transactions. Scaling out does not impact existing VMs, rather it provisions newer machines and adds them to the group.

Bare-metal versus dormant instances

Scaling new instances can take two broad approaches: creating the new instance from scratch, which means installing applications, configuring, and testing them; or starting the dormant, sleeping instances when they are needed due to scalability pressure on other servers.

Configuring the maximum and minimum number of instances appropriately

Setting a value of two for both the minimum and maximum instance count, with the current instance count being two, means no scaling action can occur. There should be an adequate margin between the maximum and minimum instance counts, which are inclusive. Auto scaling always scales between these limits.

Concurrency

Applications are designed for scalability to focus on concurrency. Applications should use asynchronous patterns to ensure that client requests do not wait indefinitely to acquire resources if resources are busy serving other requests. Implementing asynchronous patterns in code ensures that threads do not wait for resources and that systems are exhausted of all available threads. Applications should implement the concept of timeouts if intermittent failures are expected.

Stateless

Applications and services should be designed to be stateless. Scalability can become a challenge to achieve with stateful services, and it is quite easy to scale stateless services. With states comes the requirement for additional components and implementations, such as replication, centralized or decentralized repository, maintenance, and sticky sessions. All these are impediments on the path to scalability. Imagine a service maintaining an active state on a local server. No matter the number of requests on the overall application or on the individual server, the subsequent requests must be served by the same server. Subsequent requests cannot be processed by other servers. This makes scalability implementation a challenge.

Caching and the Content Distribution Network (CDN)

Applications and services should take advantage of caching. Caching helps eliminate multiple subsequent calls to either databases or filesystems. This helps in making resources available for more requests. The CDN is another mechanism that is used for caching static files, such as images and JavaScript libraries. They are available on servers across the globe. They also make resources available for additional client requests – this makes applications highly scalable.

N+1 design

N+1 design refers to building redundancy within the overall deployment for each component. It means to plan for some redundancy even when it is not required. It could mean additional VMs, storage, and network interfaces.

Summary

High availability and scalability are crucially important architectural concerns. Almost every application and every architect tries to implement high availability. Azure is a mature platform that understands the need for these architectural concerns in applications and provides resources to implement them at multiple levels. These architectural concerns are not an afterthought, and they should be part of the application development life cycle, starting from the planning phase itself. In next chapter, we will delve deeper into concepts related to security and monitoring in Azure.

3

Security and Monitoring

Security is, undoubtedly, the most important non-functional requirement for architects to implement. Enterprises put lots of emphasis on having their security strategy implemented correctly. In fact, security is one of the top concerns for almost every stakeholder in an application's development, deployment, and management. It becomes all the more important when the same application is built for deployment to the cloud.

The following topics will be covered in this chapter:

- Security
- Monitoring
- Azure monitoring
- Application Insights
- Log Analytics
- Executing runbooks on alerts
- Integrating PowerBI

Security

Security is an important element for any software or service. Adequate security should be built so that the application can only be used by people who are allowed to access it, and users should not be able to perform operations that they are not allowed to execute. Similarly, the entire request-response mechanism should be built using methods that ensure that only intended parties can understand the messages, and that it is easy to detect whether the messages have been tampered with or not. For the following reasons, security in Azure is even more important. Firstly, the organizations deploying their applications are not in full control of underlying hardware and networks. Secondly, security has to be built into each and every layer, including hardware, networks, operating systems, platforms, and applications. Any omissions or misconfigurations can render the application vulnerable to intruders.

Securing an application means that unknown and unauthorized entities are unable to access it. This also means that communication with the application is secure and not tampered with. This includes the following security measures:

- **Authentication:** Authentication checks the identity of a user and ensures that the given identity can access the application or service. Authentication is performed in Azure using OpenID Connect.
- **Authorization:** Authorization allows and establishes permissions that an identity can perform within the application or service. Authorization is performed in Azure using OAuth.
- **Confidentiality:** Confidentiality ensures that the communication between the user and the application remains secure. The payload exchange between entities is encrypted so that it will make sense only to the sender and receiver, but not otherwise. Confidentiality of messages is performed using symmetric and asymmetric encryption. Certificates are used to implement cryptography; that is, the encryption and decryption of messages.
- **Integrity:** Integrity ensures that the payload and message exchange between sender and receiver is not tampered with. The receiver receives the same message that's sent by the sender. Digital signatures and hashes are the implementation mechanisms to check the integrity of incoming messages.

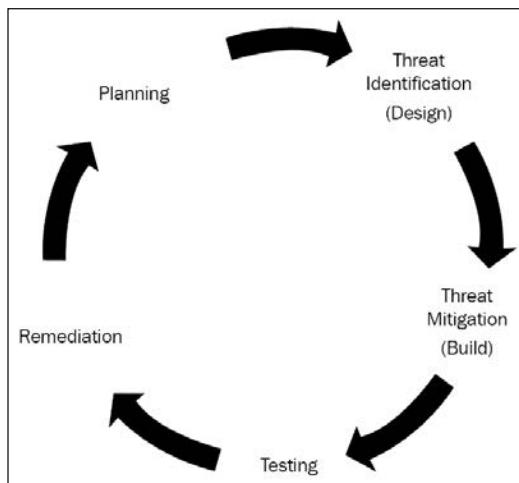
Security is a partnership between the service provider and the service consumer. Both parties have different levels of control over deployment stacks, and each should implement security best practices to ensure that all threats are identified and mitigated. We already know from *Chapter 1, Getting Started*, that the cloud broadly provides three paradigms – IaaS, PaaS, and SaaS – and each of these have different levels of collaborative control over the deployment stack. Each party should implement security practices for components under its control and within its ambit. Failure to implement security at any layer in the stack or by any party would make the entire deployment and application vulnerable to attack.

Security life cycle

Security is generally regarded as a non-functional requirement for a solution. However, with the growing number of cyber-attacks, it is considered a functional requirement these days.

Every organization follows some sort of application life cycle management for their applications. When security is treated as a functional requirement, it should follow the same process of application development. Security should not be an afterthought, it should be part of the application from the beginning. Within the overall planning phase for an application, security should also be planned. Depending on the nature of the application, different kinds and categories of threats should be identified, and, based on these identifications, they should be documented in terms of approach and scope to mitigate them. A threat modeling exercise should be undertaken to illustrate the threat each component could be subjected to. This will lead to designing security standards and policies for the application. This is typically the security design phase. The next phase is called the **threat mitigation** or **build** phase. In this phase, the implementation of security in terms of code and configuration is executed to mitigate the security threats and risks.

A system cannot be secure until it is tested. Appropriate penetration tests and other security tests should be performed to identify potential threat mitigation that has not been implemented or has been overlooked. The bugs from testing are remediated and the cycle continues throughout the life of the application. This process of application life cycle management, as shown in the following diagram, should be followed for security:



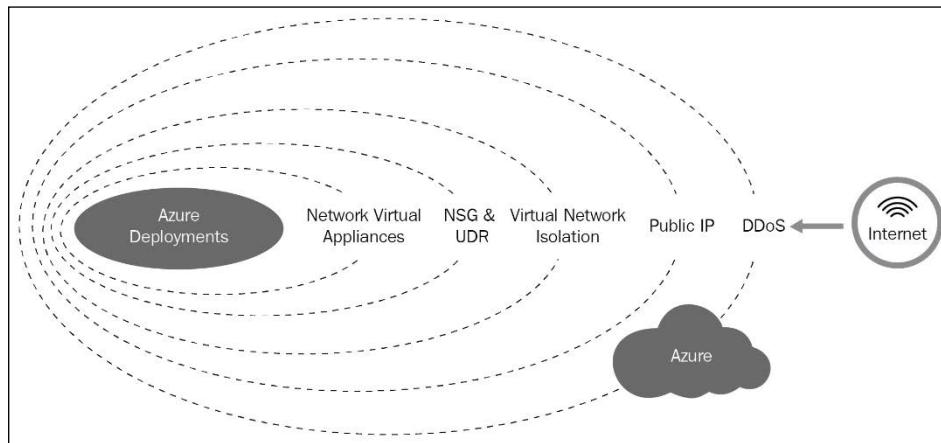
Planning, identification, mitigation, testing, and remediation are iterative processes that continue even when an application or service is operational. There should be active monitoring of entire environments and applications to proactively identify threats and mitigate them. Monitoring should also enable alerts and audit logs to help in reactive diagnosis, troubleshooting, and the elimination of threats and vulnerabilities.

The security life cycle of any application starts with the planning phase, which eventually leads to the design phase. In the design phase, the application's architecture is decomposed into granular components with discrete communication and hosting boundaries. Threats are identified based on their interaction with other components within and across hosting boundaries. Identified threats are mitigated by implementing appropriate security features within the overall architecture and testing to identify whether the identified vulnerability still exists. After the application is deployed to production and becomes operational, it is monitored for any security breaches and vulnerabilities, and either proactive or reactive remediation is conducted.

Microsoft provides complete guidance and information about the security life cycle, available at <https://www.microsoft.com/en-us/securityengineering/sdl/practices>.

Azure security

Azure provides all its services through data centers in multiple regions. These data centers are interconnected within regions, as well as across regions. Azure understands that it hosts mission-critical applications, services, and data for its customers. It must ensure that security is of the utmost importance for its data centers and regions. Customers deploy applications to the cloud based on their trust that Azure will protect their applications and data from vulnerabilities and breaches. Customers will not move to the cloud if this trust is broken, and so, Azure implements security at all layers, as seen in the next diagram, from the physical perimeter of data centers to logical software components. Each layer is protected, and even the Azure data center team does not have access to them:



Security is of paramount importance to both Microsoft and Azure. Azure is a cloud platform hosted by Microsoft. Microsoft ensures that trust is built with its customers, and it does so by ensuring that its customer deployment, solutions, and data are completely secure, physically and virtually. People will not use any cloud platform if it is not physically and digitally secure. To ensure that customers have trust in Azure, each activity in the development of Azure is planned, documented, audited, and monitored from a security perspective. The physical Azure data centers are protected for any intrusion and unauthorized access. In fact, even Microsoft personnel and operations teams do not have access to customer solutions and data. Some of the out-of-the-box security features provided by Azure are listed here:

- **Secure user access:** A customer's deployment, solution, and data can only be accessed by the customer. Even Azure data center personnel do not have access to customer artifacts. Customers can allow access to other people; however, that is at the discretion of the customer.

- **Encryption at rest:** Azure encrypts all its management data so that it cannot be read by anyone. It also provides this functionality to its customers, as well as those who can encrypt their data at rest.
- **Encryption at transit:** Azure encrypts all data that flows from its network. It also ensures that its network backbone is protected from any unauthorized access.
- **Active monitoring and auditing:** Azure monitors all its data centers actively on an ongoing basis. It actively identifies any breach, threat, or risk, and mitigates them.

Azure meets country-specific, local, international, and industry-specific compliance standards. They can be found at <https://www.microsoft.com/en-us/trustcenter/compliance/complianceofferings>.

IaaS security

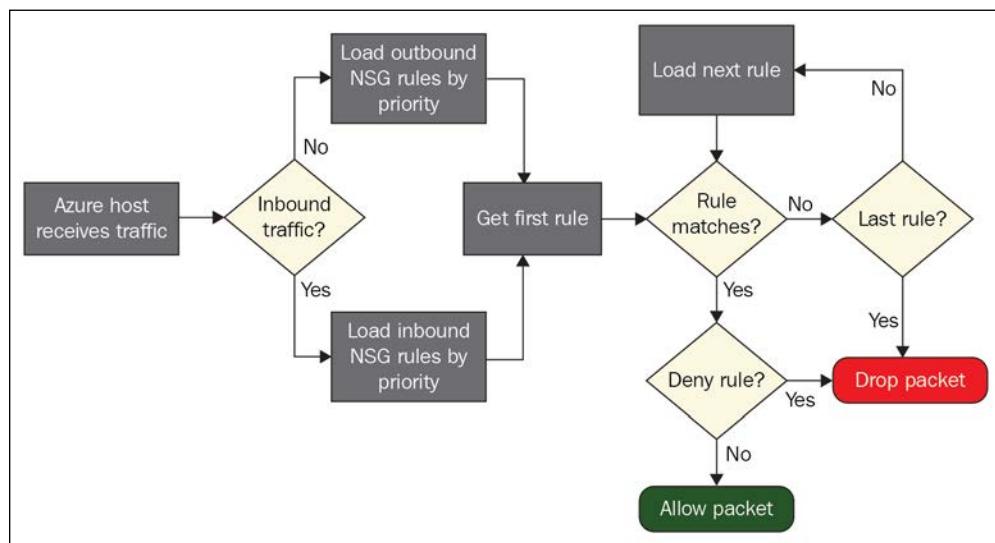
Azure is a mature platform for deploying IaaS solutions. There are lots of users of Azure who want complete control over their deployments, and they typically use IaaS for their solutions. It is important that these deployments and solutions are secure, by default and by design. Azure provides rich security features to secure IaaS solutions. In this section, some of the major features will be covered.

Network security groups

The bare minimum of IaaS deployment consists of virtual machines and virtual networks. The virtual machines might be exposed to the internet by applying a public IP to its network interface, or it might just be available only to internal resources. Some of those internal resources might, in turn, be exposed to the internet. In any case, virtual machines should be secured so that unauthorized requests should not even reach them. Virtual machines should be secured using facilities that can filter requests on the network itself, rather than the requests reaching a virtual machine and it having to take action on them. Ring-fencing is a mechanism that virtual machines use as one of its security mechanisms. This fence can allow or deny requests depending on their protocol, origin IP, destination IP, originating port, and destination port. This feature is deployed using the Azure network security groups (NSGs) resource. NSGs are composed of rules that are evaluated for both incoming and outgoing requests. Depending on the execution and evaluation of these rules, it is determined whether the requests should be allowed or denied access.

NSGs are flexible and can be applied to a virtual network subnet or individual network interfaces. When applying to a subnet, the security rules are applied to all virtual machines hosted on this subnet. On the other hand, applying to a network interface affects requests to only a particular virtual machine associated with that network interface. It is also possible to apply NSGs to both network subnets and network interfaces simultaneously. Typically, this design should be used to apply common security rules at the network subnet level, and unique security rules at the network interface level. It helps to design modular security rules and their applicability.

The flow for evaluating NSGs is shown in the following diagram:



When a request reaches a Azure host, based on whether its an inbound or outbound request, appropriate rules are loaded and each executed against the request/response. If the rule matches the request/response, either the request/response is allowed or denied. The rules matching consists of important request/response information, such as source IP address, destination IP address, source port, destination port, and protocol used.

NSG design

The first step in designing is to ascertain the security requirements of the resource. The following should be determined or considered:

- Is the resource accessible from the internet only?
- Is the resource accessible from both the internal resources and the internet?
- Is the resource accessible from the internal resource only?
- Determine the resources, load balancer, gateways, and virtual machines used.
- Configure a virtual network and its subnet.

Using the results of these investigations, an adequate NSG design should be created. Ideally, there should be multiple network subnets for each workload and type of resource. It is not recommended to deploy both load balancers and virtual machines on the same subnet.

Taking your requirements into account, rules should be determined that are common for different virtual machine workloads and subnets. For example, for a SharePoint deployment, the frontend application and SQL servers are deployed on separate subnets, so, rules for each subnet should be determined.

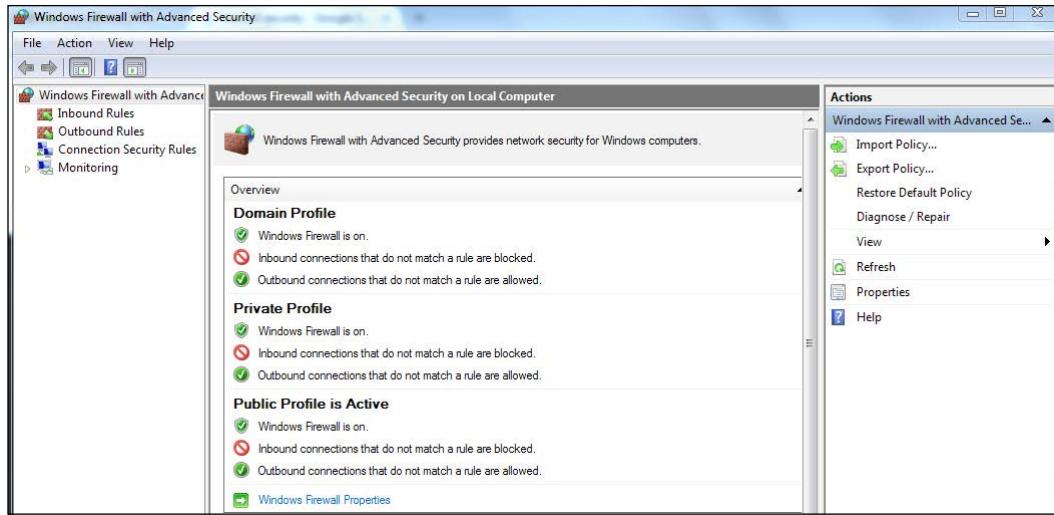
After common subnet-level rules are identified, rules for individual resources should be identified, and these should be applied to the network interface level. It is important to understand that if a rule allows an incoming request on a port, that port can also be used for outgoing requests without any configuration.

If resources are accessible from the internet, rules should be created with specific IP ranges and ports where possible. Careful functional and security testing should be executed to ensure that adequate and optimal NSG rules are opened and closed.

Firewalls

NSGs provide external security perimeters for requests. However, this does not mean that virtual machines should not implement additional security measures. It is always better to implement security both internally and externally. Virtual machines, whether in Linux or Windows, provide a mechanism to filter requests at the operating system level. This is known as a firewall in both Windows and Linux.

It is advisable to implement firewalls for operating systems. They help build a virtual security wall that allows only those requests that are considered trusted. Any untrusted requests are denied access. There are even physical firewall devices, but, on the cloud, operating system firewalls are used. The following screenshot shows the firewall configuration in the Windows operating system:



Firewalls filter network packets and identify incoming ports and IP addresses. Using the information from these packets, the firewall evaluates the rules and decides whether it should allow or deny access.

Firewall design

As a best practice, firewalls should be evaluated for individual operating systems. Each virtual machine has a distinct responsibility within the overall deployment and solution. Rules for these individual responsibilities should be identified and firewalls should be opened and closed accordingly.

While evaluating firewall rules, it is important to keep NSG rules at both the subnet and individual network interface level into consideration. If it's not done properly, it is possible that rules are denied at the NSG level, but left open at the firewall level, and vice versa. If a request is allowed at the NSG level and denied at the firewall level, the application will not work as intended, while security risks increase if a request is denied at the NSG level and allowed at the firewall level.

A firewall helps you build multiple networks isolated by its security rules. Careful functional and security testing should be executed to ensure that adequate and optimal firewall rules are opened and closed.

Reducing the attack surface area

NSGs and firewalls help you to manage authorized requests to the environment. However, the environment should not be overly exposed to security attacks. The surface area of the system should be optimally enabled to achieve its functionality, but disabled enough that attackers cannot find loopholes and access areas that are opened without any intended use, or opened but not adequately secured. Security should be adequately hardened, making it difficult for any attacker to break into the system.

Some of the configurations that should be done include the following:

- Remove all unnecessary users and groups from the operating system.
- Identify group membership for all users.
- Implement group policies using directory services.
- Block script execution unless it is signed by trusted authorities.
- Log and audit all activities.
- Install malware and anti-virus software, schedule scans, and update definitions frequently.
- Disable or shut down services that are not required.
- Lock down the filesystem so only authorized access is allowed.
- Lock down changes to the registry.
- A firewall must be configured according to the requirements.
- PowerShell script execution should be set to restricted or RemoteSigned.
- Enable enhanced protection through Internet Explorer.
- Restrict the ability to create new users and groups.
- Remove internet access and implement jump servers for RDP.
- Prohibit logging into servers using RDP through the internet. Instead, use site-to-site VPN, point-to-site VPN, or express routes to RDP into remote machines from within the network.
- Regularly deploy all security updates.
- Run the security compliance manager tool on the environment and implement all of its recommendations.
- Actively monitor the environment using the Security Center and Operations Management suite.

- Deploy virtual network appliances to route traffic to internal proxies and reverse proxies.
- All sensitive data, such as configuration, connection strings, and credentials, should be encrypted.

Implementing jump servers

It is a good idea to remove internet access from virtual machines. It is also a good practice to limit remote desktop services' accessibility from the internet, but then how do you access the virtual machines at all? One good way is to only allow internal resources to RDP into virtual machines using Azure VPN options. However, there is also another way – using **jump servers**.

Jump servers are servers that are deployed in the **demilitarized zone (DMZ)**. This means it is not on the network hosting the core solutions and applications. Instead, it is on a separate network or subnet. The primary purpose of the jump server is to accept RDP requests from users and help them log in to it. From this jump server, users can further navigate to other virtual machines using RDP. It has access to two or more networks: one that has connectivity to the outside world, and another internal to the solution. The jump server implements all the security restrictions and provides a secure client to connect to other servers. Normally, access to emails and the internet is disabled on jump servers.

An example of deploying a jump server with the VMSS is available at <https://azure.microsoft.com/en-in/resources/templates/201-vmss-windows-jumpbox/> using Azure Resource Manager templates.

PaaS security

Azure provides numerous PaaS services, each with their own security features. In general, PaaS services can be accessed using credentials, certificates, and tokens. PaaS services allow for the generation of short-lived security access tokens. Client applications can send this security access token to represent trusted users. In this section, we will cover some of the most important PaaS services that are used in almost every solution.

Log Analytics

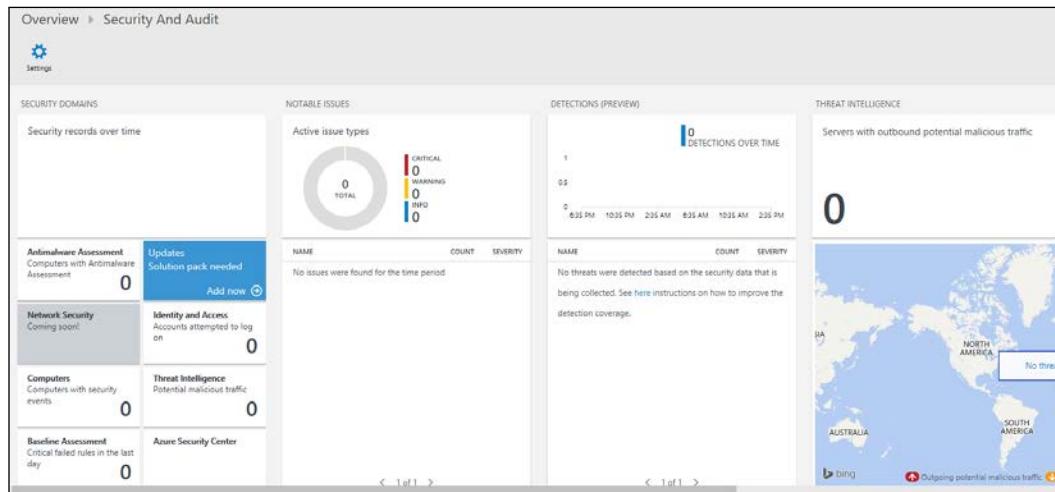
Log Analytics is a new analytics platform for managing cloud deployments, on-premises data centers, and hybrid solutions.

It provides multiple modular solutions – a specific functionality that helps to implement a feature. For example, security and audit solutions help to ascertain a complete view of security for an organization's deployment. Similarly, there are many more solutions, such as automation and change tracking, that should be implemented from a security perspective.

The Log Analytics security and audit provides information in the following five categories:

- **Security domains:** These provide the ability to view security records, malware assessments, update assessments, network security, identity and access information, and computers with security events. Access is also provided to the Azure Security Center dashboard.
- **Anti-malware assessment:** This helps to identify servers that are not protected against malware and have security issues. It provides an overall exposure to potential security problems and assesses their criticality. Users can take proactive actions based on these recommendations. Azure Security Center sub-categories provide information collected by Azure Security Center.
- **Notable issues:** This quickly identifies active issues and grades their severity.
- **Detections:** This category is in the preview mode. It enables the identification of attack patterns by visualizing security alerts.
- **Threat intelligence:** This helps to identify attack patterns by visualizing the total number of servers with outbound malicious IP traffic, the malicious threat type, and a map that shows where these IPs are coming from.

The preceding details, when viewed from the portal, are shown in the following screenshot:



Storage

Storage accounts play an important part in the overall solution architecture. Storage accounts can store important information, such as user PII data, business transactions, and data. It is of the utmost importance that storage accounts are secure and only allow access to authorized users. The data stored is encrypted and transmitted using secure channels. Storage, as well as the users and client applications consuming the storage account and its data, plays a crucial role in the overall security of the data. They should also keep data encrypted at all times. This also includes credentials and connection strings connecting to data stores.

Azure provides **role-based access control (RBAC)** to govern who can manage Azure Storage accounts. These RBAC permissions are given to users and groups in Azure **Active Directory (AD)**. However, when an application to be deployed on Azure is created, it will have users and customers that are not available in Azure AD. To allow users to access the storage account, Azure Storage provides storage access keys. There are two types of access keys at the storage account level – primary and secondary. Users possessing these keys can connect to the storage account. These storage access keys are used in the authentication step when accessing the storage account. Applications can access storage accounts using either primary or secondary keys. Two keys are provided so that if the primary key is compromised, applications can be updated to use the secondary key while the primary key is regenerated. This helps minimize application downtime. Moreover, it enhances security by removing the compromised key without impacting applications. The Storage key details, as seen on the Azure portal, is shown in the following screenshot:

The screenshot shows the 'Storage account name' field filled with a redacted value. Below it, the 'Default keys' section displays two entries:

NAME	KEY	CONNECTION STRING
key1	[REDACTED]	DefaultEndpointsProtocol=https;AccountName=[REDACTED]
key2	[REDACTED]	DefaultEndpointsProtocol=https;AccountName=[REDACTED]

Each key entry includes a copy icon and a refresh icon.

Azure Storage provides four services – blob, files, queues, and tables – in an account. Each of these services also provides infrastructure for securing themselves using secure access tokens. A **shared access signature (SAS)** is a URI that grants restricted access rights to Azure Storage services: blobs, files, queues, and tables. These SAS tokens can be shared with clients who should not be trusted with the entire storage account key to constrain access to certain storage account resources. By distributing an SAS URI to these clients, access to resources is granted for a specified period.

SAS tokens exist at both the storage account and the individual blob, file, table, and queue levels. A storage account-level signature is more powerful and has the right to allow and deny permissions at the individual service level. It can also be used instead of individual resource service levels:

Allowed services ⓘ

Blob File Queue Table

Allowed resource types ⓘ

Service Container Object

Allowed permissions ⓘ

Read Write Delete List Add Create Update Process

Start and expiry date/time ⓘ

Start
2017-07-30 10:39:25 AM

End
2017-07-30 6:39:25 PM

UTC - Coordinated Universal Time

Allowed IP addresses ⓘ
for example, 168.1.5.65 or 168.1.5.65-168.1.5.70

Allowed protocols ⓘ

HTTPS only HTTPS and HTTP

Signing key ⓘ

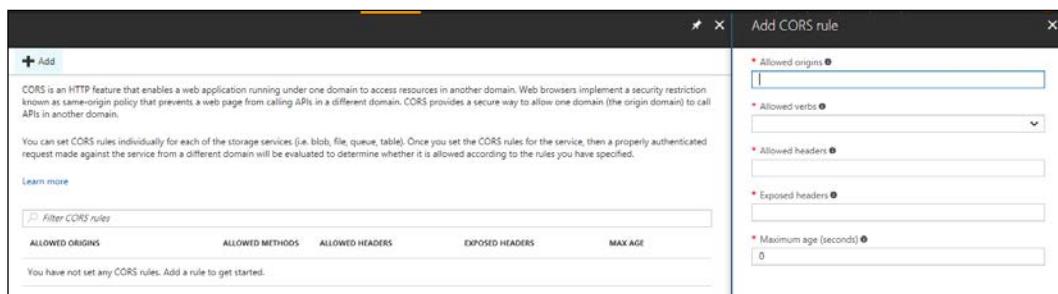
key1

Generate SAS

Generating and sharing SAS tokens is preferable to sharing storage account keys. SAS tokens provide granular access to resources and can be combined together as well. These tokens include **read**, **write**, **delete**, **list**, **add**, **create**, **update**, and **process**. Moreover, even access to resources can be determined while generating SAS tokens. It could be for blobs, tables, queues, and files individually, or a combination of them. Storage account keys are for the entire account and cannot be constrained for individual services. Neither can they be constrained from the permissions perspective. It is much easier to create and revoke SAS tokens than it is for storage access keys. SAS tokens can be created for use for a certain period of time, after which they automatically become invalid.

It is to be noted that if storage account keys are regenerated, then the SAS token based on them will become invalid and a newer SAS token should be created and shared with clients.

Cookie stealing, script injection, and denial of service attacks are common means used by attackers to disrupt an environment and steal data. Browsers and the HTTP protocol implement a built-in mechanism that ensures that these malicious activities cannot be performed. Generally, anything that is cross-domain is not allowed by either HTTP or browsers. A script running in one domain cannot ask for resources from another domain. However, there are valid use cases where such requests should be allowed. The HTTP protocol implements **Cross-Origin Resource Sharing (CORS)**. With the help of CORS, it is possible to access resources across domains and make them work. Azure Storage configures CORS rules for blobs, file, queue, and table resources. Azure Storage allows for the creation of rules that are evaluated for each authenticated request. If the rules are satisfied, the request is allowed to access the resource:



Data must not only be protected while in transit; it should also be protected while at rest as well. If data at rest is not encrypted, anybody who has access to the physical drive in the data center can read the data. Although the possibility is negligible, customers still should encrypt their data. Storage service encryption also helps protect data at rest. This service works transparently and injects itself without users knowing about it. It encrypts data when the data is saved in a storage account, and decrypts it automatically when it is read. This entire process happens without users performing any additional activity.

Azure account keys must be rotated periodically. This will ensure that an attacker is not able to breach access to storage accounts.

It is also a good idea to regenerate the keys; however, this must be evaluated in regard to its usage in existing applications. If it breaks the existing application, these applications should be prioritized for change management, and changes should be applied gradually.

Individual service-level SAS tokens with a limited timeframe should be generated and provided to users who should access the resources as much as possible. Permissions must be evaluated and optimum permissions must be provided.

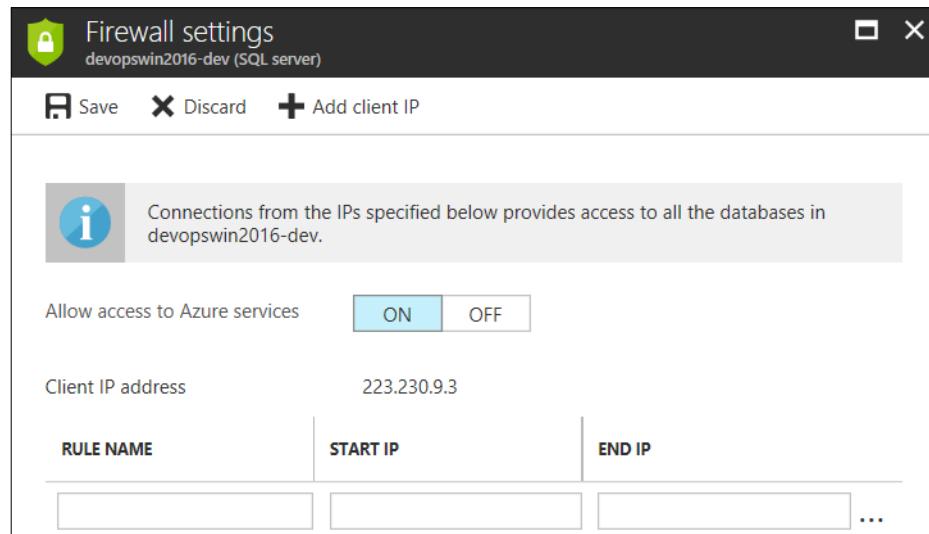
SAS keys and storage account keys should be stored in Azure Key Vault. This provides security storage and access to them. These keys can be read at runtime by applications from the key vault, instead of storing them in configuration files.

Azure SQL

SQL Server stores relational data on Azure. It is an SaaS that provides a highly available, scalable, performance-centric, and secure platform for storing data. It is accessible from anywhere, with any programming language and platform. Clients need a connection string comprising the server, database, and security information to connect to it.

SQL Server provides firewall settings that prevent access to anyone by default. IP addresses and ranges should be whitelisted to access SQL Server. Only IP addresses that architects are confident belong to customers or partners should be whitelisted. There are deployments in Azure for which either there are a lot of IP addresses or the IP addresses are not known, such as applications deployed in Azure Functions or Logic Apps. For such applications to access Azure SQL, Azure SQL allows whitelisting of all IP addresses to Azure services across subscriptions.

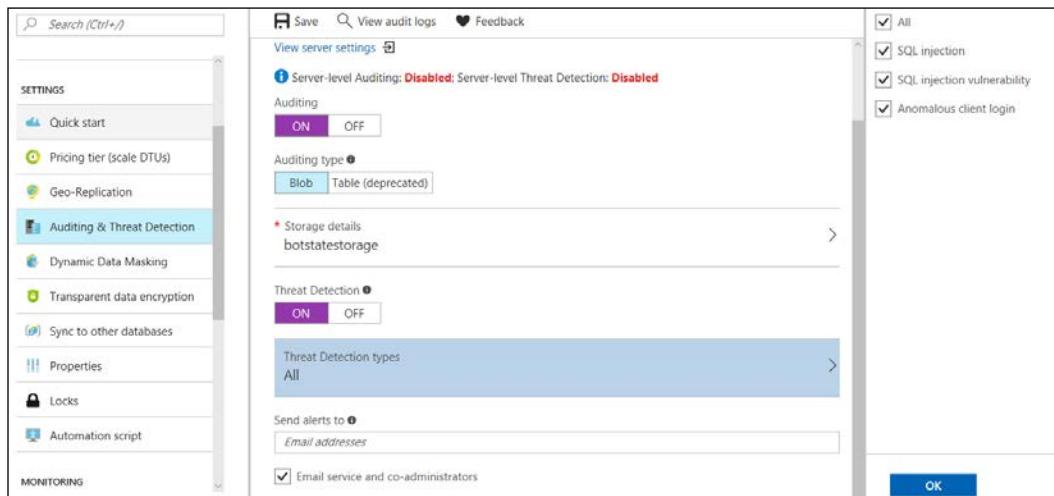
It is to be noted that firewall configuration is at the server level and not the database level. This means that any changes here affect all databases within a server:



Azure SQL also provides enhanced security by encrypting data at rest. This ensures that nobody, including the Azure data center administrators, can view the data stored in SQL Server. The technology used by SQL Server for encrypting data at rest is known as **Transparent Data Encryption (TDE)**. There are no changes required at the application level to implement TDE. SQL Server encrypts and decrypts data transparently when the user saves and reads data. This feature is available at the database level.

SQL Server also provides **dynamic data masking (DDM)**, which is especially useful for masking certain types of data, such as credit cards or user PII data. Masking is not the same as encryption. Masking does not encrypt data, but only masks, which ensures that data is not in a human-readable format. Users should mask and encrypt sensitive data in the Azure SQL server.

SQL Server also provides an **auditing and threat detection** service for all servers. There are advanced data collection and intelligence services running on top of these databases to discover threats and vulnerabilities and alert users to them. Audit logs are maintained by Azure in storage accounts and can be viewed by administrators to be actioned. Threads such as SQL injection and anonymous client logins can generate alerts that administrators can be informed about over email:



Data can be masked in Azure SQL. This helps us store data in a format that does not make sense:

The screenshot shows the Azure portal interface for managing Dynamic Data Masking rules. On the left, there's a sidebar with a search bar and a 'SETTINGS' section containing links like 'Quick start', 'Pricing tier (scale DTUs)', 'Geo-Replication', 'Auditing & Threat Detection', 'Dynamic Data Masking', and 'Transparent data encryption'. The 'Dynamic Data Masking' link is highlighted. The main content area has tabs for 'MASK NAME' and 'MASK FUNCTION'. A message says 'You haven't created any masking rules.' Below this is a section for 'SQL users excluded from masking (administrators are always excluded)'. On the right, there's a separate panel for creating a new masking rule, with fields for 'Mask name', 'Select what to mask' (Schema, Table, Column dropdowns), and 'Select how to mask' (Masking field format dropdown with 'Default value (0, xxxx, 01-01-1900)').

Azure SQL also provides **transparent data encryption** to encrypt data at rest, as shown in the following screenshot:

The screenshot shows the Azure portal interface for managing transparent data encryption. The left sidebar has a 'SETTINGS' section with 'Transparent data encryption' selected. The main content area has a summary message: 'Encrypts your databases, backups enable TDE, go to each database.' with a 'Learn more' link. Below this is a 'Data encryption' section with an 'ON' button (which is highlighted in blue) and an 'OFF' button. At the bottom, it shows 'Encryption status' with a green checkmark and the word 'Encrypted'.

Azure Key Vault

Securing resources using passwords, keys, credentials, certificates, and unique identifiers are an important element for any environment and application. They are important elements from the security perspective. They need to be protected, and to ensure that these resources remain secure and do not get compromised is an important pillar of security architecture. Management and operations that keep the secrets and keys secure, while making them available when needed, are important aspects that cannot be ignored. Typically, these secrets are used all over the place – within the source code, inside configuration files, on pieces of paper, and in other digital formats. To overcome these challenges and store all secrets uniformly in a centralized secure storage, Azure Key Vault should be created.

Azure Key Vault is well integrated with other Azure services. For example, it would be easy to use a certificate stored in Azure Key Vault and deploy it to an Azure virtual machine's certificate store. All kinds of keys, including storage keys, IoT and event keys, and connection strings can be stored as secrets in Azure Key Vault. They can be retrieved and used transparently without anyone viewing them or storing them temporarily anywhere. Credentials for SQL Server and other services can also be stored in Azure Key Vault.

Azure Key Vault works on a per-region basis. What this means is that an Azure Key Vault resource should be provisioned at the same region where the application and service is deployed. If a deployment consists of more than one region and needs services from Azure Key Vault, multiple Azure Key Vault instances should be provisioned.

An important feature of Azure Key Vault is that the secrets, keys, and certificates are not stored in the general storage. This sensitive data is backed up by the hardware security module. This means that this data is stored in separate hardware on Azure that can only be unlocked by keys owned by users.

Security monitoring and auditing

Azure provides the following two important security resources to manage all security aspects of the Azure subscription, resource groups, and resources:

- Azure Monitor
- Azure Security Center

Azure Monitor

Azure Monitor is a one-stop place for monitoring Azure resources. It provides information about Azure resources and their state. It provides a rich query interface, using information that can be sliced and diced using data at the levels of subscription, resource group, individual resource, and resource type.

Azure Monitor can be used through the Azure portal, PowerShell, CLI, and REST API:

The screenshot shows the Azure Monitor - Activity log interface. On the left, there's a sidebar with options like 'Activity log', 'Operation log (classic)', 'Metrics', 'Diagnostics settings', and 'Log search'. The main area has a search bar at the top. Below it are several filter dropdowns: 'Subscription' (selected 'Resource group'), 'Resource group' (selected 'All resource groups'), 'Resource' (selected 'All resources'), 'Resource type' (selected 'All resource types'), 'Operation' (selected '0 selected'), 'Event category' (selected 'Event severity'), 'Event severity' (selected '0 selected'), 'Timestamp' (selected 'Last 6 hours'), 'Category' (selected 'All categories'), 'Event severity' (selected '4 selected'), and 'Email or name or service principal' (empty). There are 'Apply' and 'Reset' buttons. Below the filters is a table with columns: OPERATION NAME, STATUS, TIME, TIME STAMP, SUBSCRIPTION, and EVENT INITIATED BY. A message says 'No results to display'.

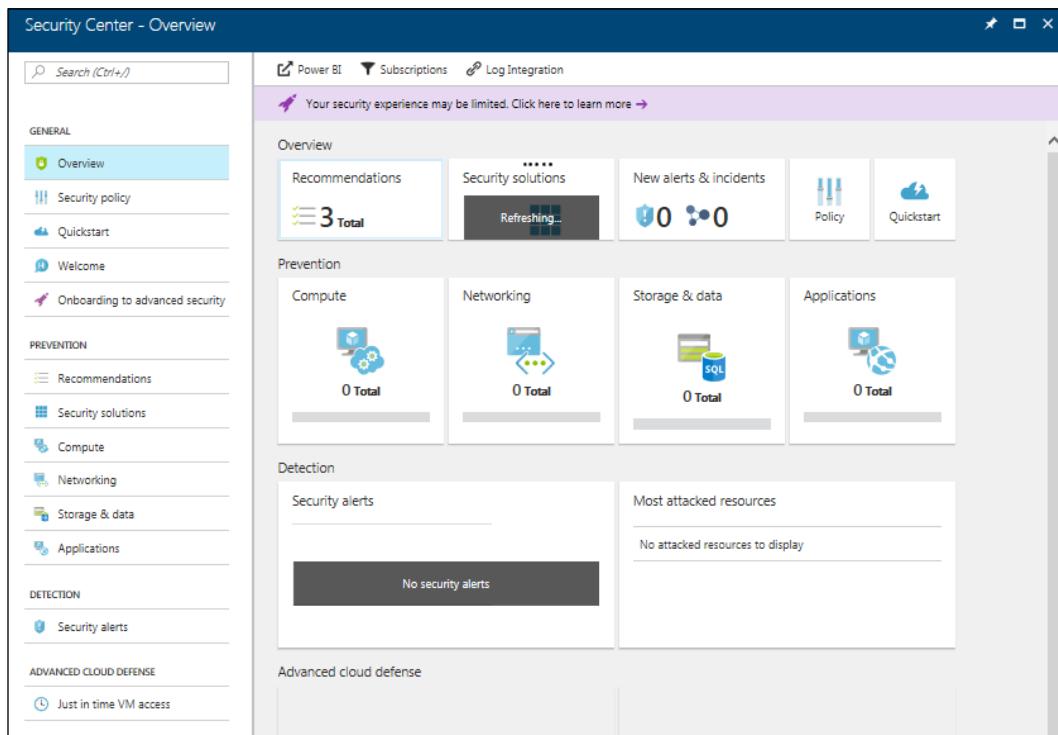
The following logs are those provided by Azure Monitor:

- **Activity log:** This provides all management-level operations performed on resources. It provides details about the creation time, creator, resource type, and status of resources.
- **Operation log (classic):** This provides details of all operations performed on resources within a resource group and subscription.
- **Metrics:** This gets performance information for individual resources and sets alerts on them.
- **Diagnostic settings:** This helps us to configure the effects logs by setting up Azure Storage for storing logs, streaming logs in real time to Azure Event Hubs, and sending them to Log Analytics.
- **Log search:** This helps integrate Log Analytics with Azure Monitor.

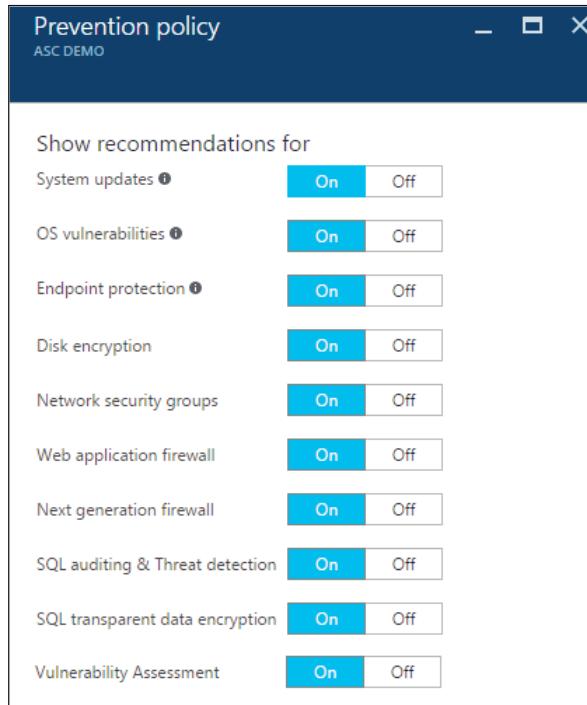
Azure Monitor can help identify security-related incidents and take appropriate actions. It is important that only authorized individuals should be allowed to access Azure Monitor, since it might contain sensitive information.

Azure Security Center

Azure Security Center, as the name suggests, is a one-stop place for all security needs. There are generally two activities related to security – implementing security and monitoring for any threats and breaches. Security Center has been built primarily to help with both these activities. Azure Security Center enables users to define their security policies and get them implemented on Azure resources. Based on the current state of Azure resources, Azure Security Center provides security recommendations to harden the solution and individual Azure resources. The recommendations include almost all Azure security best practices, including the encryption of data and disks, network protection, endpoint protection, access control lists, the whitelisting of incoming requests, and the blocking of unauthorized requests. The resources range from infrastructure components such as load balancers, network security groups, and virtual networks, to PaaS resources such as Azure SQL and Storage:



Azure Security Center is a rich platform, and provides recommendations for multiple services, as shown in the following screenshot:



Monitoring

Monitoring is an important architectural concern that should be part of any solution, big or small, mission-critical or not, cloud or not, and it should not be neglected.

Monitoring refers to the act of keeping track of solutions and capturing various telemetry information, processing it, identifying the information that qualifies for alerts based on rules, and raising them. Generally, an agent is deployed within the environment and monitors it, sending telemetry information to a centralized server, where the rest of the processing of generating alerts and notifying stakeholders happens.

Monitoring takes both proactive and reactive actions and measures on the solution. It is also the first step toward the auditability of the solution. Without the availability of monitoring log records, it is difficult to audit the system from various perspectives, such as security, performance, and availability.

Monitoring helps us identify availability, performance, and scalability issues before they happen. Hardware failure, software misconfiguration, and patch update challenges can be discovered well before they impact users by using monitoring and performance degradation can be fixed before it happens.

Monitoring reactively logs pinpoint areas and locations that are causing issues, identifies the issues, and enables faster and better repairs.

Teams can identify patterns of issues using monitoring telemetry information and eliminate them by innovating new solutions and features.

Azure is a rich cloud environment that provides multiple rich monitoring features and resources to monitor not only cloud-based deployment, but also on-premise deployment.

Azure monitoring

The first question that should be answered is, *what must we monitor?* This question becomes more important for solutions that are deployed on the cloud because of the constrained control over it.

There are some important components that should be monitored. They include the following:

- Custom applications
- Azure resources
- Guest operating systems (virtual machines)
- Host operating systems (Azure physical servers)
- Azure infrastructure

There are different Azure logs and monitoring for these components.

Azure activity logs

Previously known as audit logs and operational logs, activity logs are control-plane events on the Azure platform. They provide information and telemetry information at the subscription level, instead of the individual resource level. They track information about all changes that happen at the subscription level, such as creating, deleting, and updating resources using **Azure Resource Manager (ARM)**. They help us discover the identity (such as service principal, users, or groups) and perform an action (such as write or update) on resources (for example, storage, virtual machines, or SQL) at any given point of time. They provide information about resources that are modified in their configuration, but not their inner workings and execution.

Azure diagnostic logs

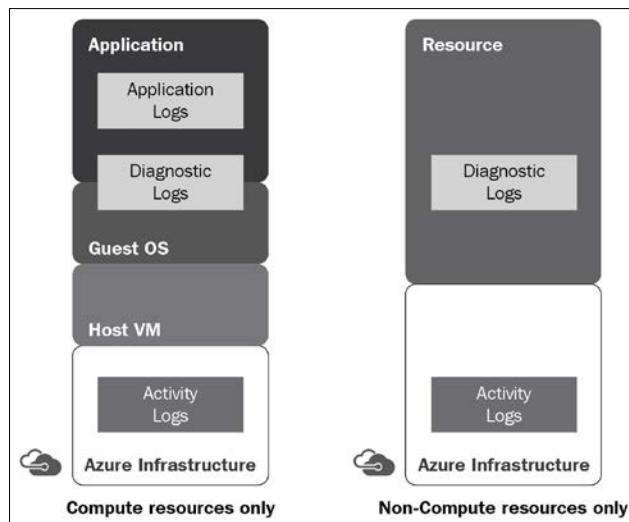
The information originating within the inner workings of Azure resources is captured in what are known as **diagnostic logs**. They provide telemetry information about the operations of resources that are inherent to the resource. Not every resource provides diagnostic logs, and resources that provide logs on their own content are completely different from other resources. Diagnostic logs are configured individually for each resource. Examples of diagnostic logs include storing a file in a container in a blob service in a storage account.

Azure application logs

The application logs can be captured by Application Insights resources and can be managed centrally. They get information about the inner workings of custom applications, such as their performance metrics and availability, and users can get insights from them in order to manage them better.

Guest and host operating system logs

Both guest and host operating system logs are surfaced to users using Azure Monitor. They provide information about the status of host and guest operating systems:



The important resources from Azure related to monitoring are as follows:

- Azure Monitor
- Azure Application Insights
- Log Analytics, previously known as **Operational Insights**

There are other tools, such as **System Center Operations Manager (SCOM)**, that are not part of the cloud feature but can be deployed on IaaS-based virtual machines to monitor any workload on Azure or an on-premises data center.

Azure Monitor

Azure Monitor is a central tool and resource that provides complete management features that allow you to monitor an Azure subscription. It provides management features for activity logs, diagnostic logs, metrics, Application Insights, and Log Analytics. It should be treated as a dashboard and management resource for all other monitoring capabilities.

Azure Application Insights

Azure Application Insights provides centralized, Azure-scale monitoring, logs, and metrics capabilities to custom applications. Custom applications can start sending metrics, logs, and other telemetry information to Azure Application Insights. It also provides rich reporting, dashboarding, and analytics capabilities to get insights from incoming data and act on them.

Azure Log Analytics

Azure Log Analytics provides centralized processing of logs and generates insights and alerts from them. Activity logs, diagnostic logs, application logs, event logs, and even custom logs can send information to Log Analytics, which can further provide rich reporting, dashboarding, and analytics capabilities to get insights from incoming data and act on them.

Application Insights

As the name suggests, Azure Application Insights provides insights into the workings of an application. The insights relevant for a web application include the number of incoming requests per second, requests failed per second, hardware usage in terms of CPU utilization, and memory availability. Application Insights provides a dashboard, reports, and charts to view various metrics related to the application. This allows us to view and better understand the trends in terms of usage of the application, its availability, the number of requests, and more, to take both precautionary and reactive actions on the application. Trend information can be used to find out things that are not working in favor of the application and things that are working fine over a specific period.

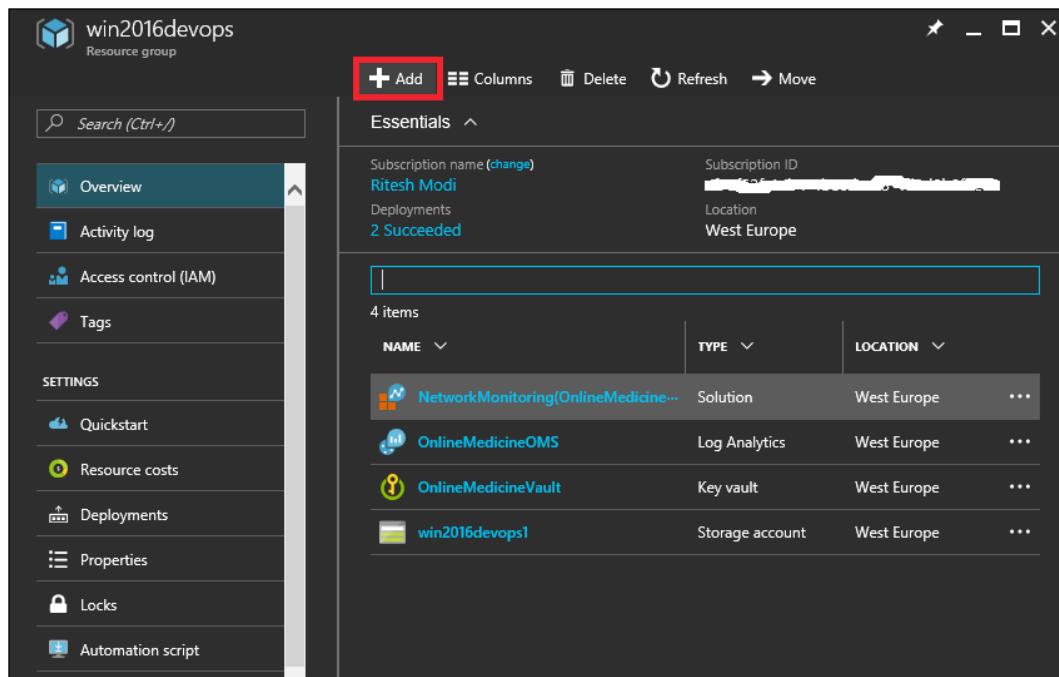
The first step in working with Application Insights is to provision this service on Azure in a resource group that has all general and management services consumed by all applications. If you remember, we created a similar resource group named `Win2016devOps` that houses all common services, such as Azure Key Vault, Application Insights, Operational Insights, and Storage, to hold scripts and templates used across environments and applications.

Provisioning

As mentioned before, the first step in consuming an Application Insights service is to provision it on Azure.

Application Insights can be provisioned manually using the Azure portal, Azure REST APIs, PowerShell, and ARM templates. Here we will use the Azure portal:

1. Log in to the Azure portal using your credentials and navigate to an existing resource group, or create a new resource group. Click on the **Add** button:

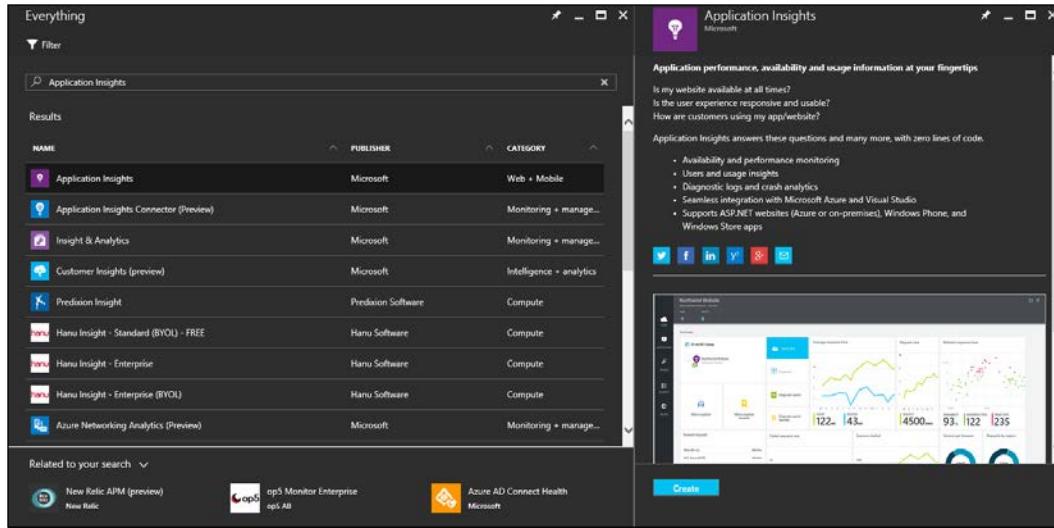


The screenshot shows the Azure portal interface for a resource group named `win2016devops`. The left sidebar contains navigation links like Overview, Activity log, Access control (IAM), Tags, Quickstart, Resource costs, Deployments, Properties, Locks, and Automation script. The main area displays resource details for the current group, including the subscription name (`Ritesh Modi`), subscription ID, location (West Europe), and a list of resources. The 'Add' button at the top of the main content area is highlighted with a red box.

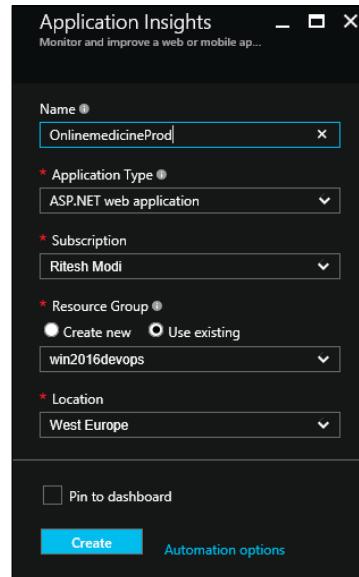
NAME	TYPE	LOCATION	...
NetworkMonitoring(OnlineMedicine...)	Solution	West Europe	...
OnlineMedicineOMS	Log Analytics	West Europe	...
OnlineMedicineVault	Key vault	West Europe	...
win2016devops1	Storage account	West Europe	...

Security and Monitoring

2. Type Application Insights in the search box in the next blade. The first link should refer to Application Insights. Click on it to create a new Application Insights service instance. Click on the **Create** button to get started:



3. The next blade will ask for the Application Insights service instance name, the type of application, the subscription name, the resource group name, and the location of the service. Provide the appropriate details and click on the **Create** button. This will provision the service:



- Now, navigate to the service that shows the essential properties, such as its **Instrumentation Key** as highlighted in the following screenshot. The key will be different in every instance, and is generally copied and used in Visual Studio. Please note that some of the information has been blacked out for security reasons:

The screenshot shows the Azure Application Insights Overview page for the service 'OnlinemedicineProd'. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, INVESTIGATE (Application map, Smart Detection, Live Metrics Stream, Availability, Failures, Performance, Servers, Browser, Usage), and CONFIGURE. The main content area displays the 'Essentials' section with details: Resource group (win2016devops), Location (West Europe), Subscription name (Ritesh Modi), and Type (ASP.NET). The 'Instrumentation Key' field is highlighted with a red box and contains the value 'befc0dbe-c068-46f8-8443-01...'. Below this, there are sections for Health (Overview timeline, SERVER RESPONSE TIME), Alerts (0), Click to configure, Smart Detection (ONLINEMEDICINEPROD, 0 detections last 24h), Web tests (0), and App map.

Log Analytics

The Log Analytics service helps in providing information about these environments using its rich log storage and analytic features. Log Analytics is used for capturing logs from multiple sources including applications. However, it is equally important to monitor the environment on which an application is hosted and running. This involves the infrastructure, such as virtual machines, Docker containers, and related components.

Provisioning

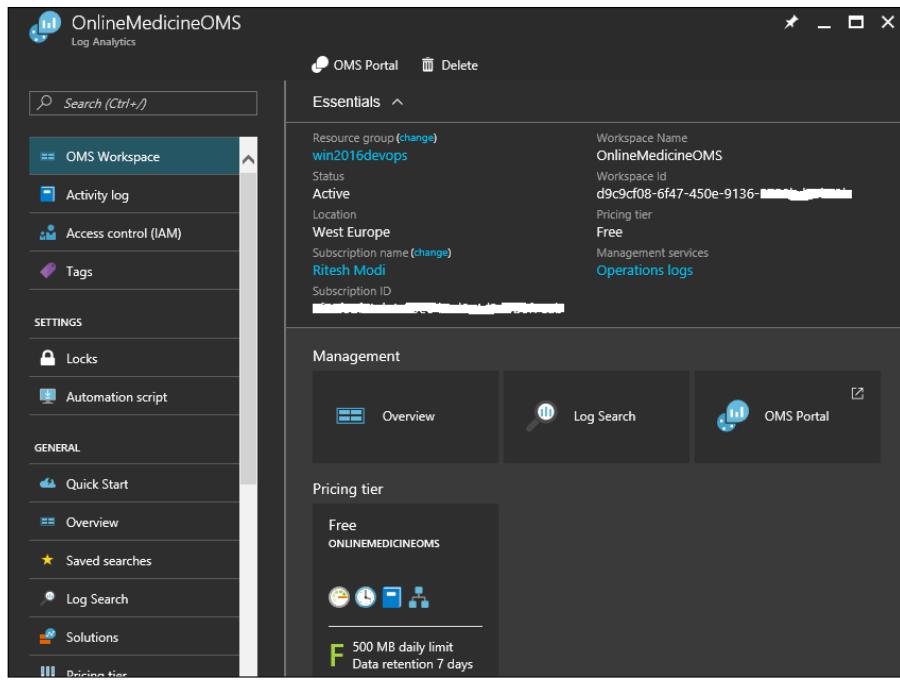
Log Analytics must be provisioned on Azure before it can be used to monitor virtual machines and containers. Again, similar to Application Insights, Log Analytics can be provisioned through the Azure Portal, PowerShell, REST APIs, and resource group manager templates. A Log Analytics workspace is a security boundary that can be accessed by certain users. Multiple workspaces should be created to isolate users and their corresponding access to environment telemetry data.

The JSON script used to provision a Log Analytics workspace is shown here:

```
{  
    "apiVersion": "2015-11-01-preview",  
    "type": "Microsoft.OperationalInsights/workspaces",  
    "name": "[parameters('workspaceName')]",  
    "location": "[parameters('deployLocation')]",  
    "properties": {  
        "sku": {  
            "Name": "[parameters('serviceTier')]"  
        }  
    }  
}
```

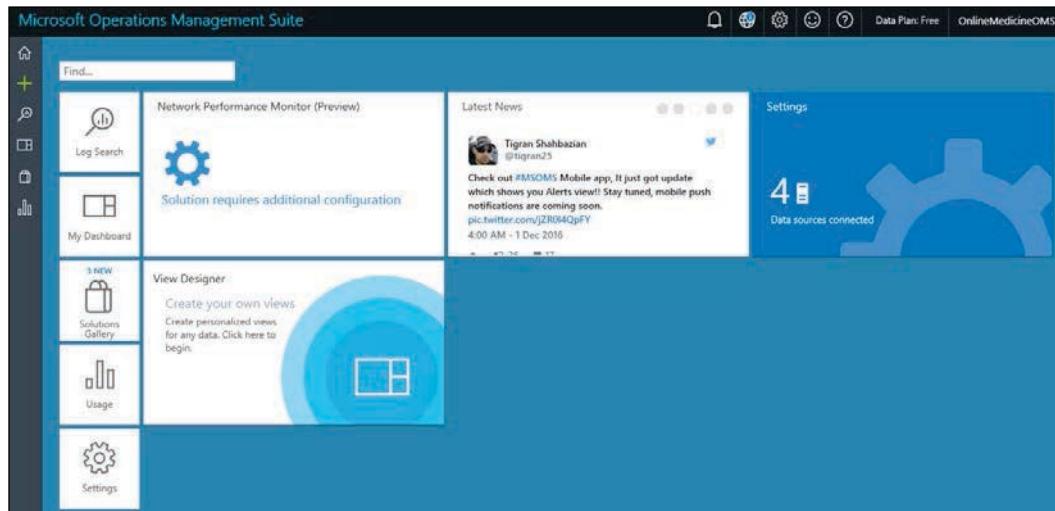
The name, location, and sku information is needed to provision a workspace, and their values are provided using parameters.

The workspace after provisioning is shown in the following screenshot:



Click on the Log Analytics portal section to open the workspace portal. This portal is used to view all telemetry information captured by Operational Insights, configure Operational Insights, and provide dashboard features and functionality.

The home screen of Log Analytics is shown here:



The **Solutions** section shows that two solutions were deployed. A different strategy of having a separate workspace for each environment can be adopted, and it is left to readers to decide what's best for their applications and solutions. Log Analytics can be configured using the **Settings** tile:

The screenshot shows the Microsoft Operations Management Suite interface. On the left, there is a navigation sidebar with icons for Home, Plus, Overview, Settings, Solutions (which is selected and highlighted in blue), Connected Sources, Data, Computer Groups, Accounts, Alerts, and Preview Features. The main content area is titled "Installed Solutions: 2". It lists "Log Search" and "Network Performance Monitor (Preview)". There is a "Remove" button next to the Network Performance Monitor entry.

Log Analytics agents

You may have noticed that there are no code changes required at the application level to consume Log Analytics. Log Analytics depends on the installation of an agent on virtual machines. These agents collect telemetry data and send them to the Log Analytics workspace, where they are stored for a specified period of time depending upon the chosen SKU. These agents can be installed manually on virtual machines or automatically using ARM templates immediately after provisioning the virtual machines. The JSON code for the provisioning of an agent on a virtual machine is as follows:

```
{  
    "apiVersion": "2015-06-15",  
    "type": "Microsoft.Compute/virtualMachines/extensions",  
    "name": "[concat(variables('vmName'),copyIndex(1),'/omsscript')]",  
    "location": "[resourceGroup().location]",  
    "dependsOn": [  
        ...  
    ]  
}
```

```
[concat('Microsoft.Compute/virtualMachines/',variables('vmName')  
,copyIndex(1))],  
[resourceId('Microsoft.Compute/virtualMachines/extensions', conc  
at(variables('vmName'),copyIndex(1),'powershellscript'))]  
,  
"copy": {  
    "count": "[parameters('countVMs')]",  
    "name": "omsloop"  
},  
"properties": {  
    "publisher": "Microsoft.EnterpriseCloud.Monitoring",  
    "type": "MicrosoftMonitoringAgent",  
    "typeHandlerVersion": "1.0",  
    "settings": {  
        "workspaceId": "[parameters('WorkspaceID')]"  
    },  
    "protectedSettings": {  
        "workspaceKey": "[listKeys(variables('accountid'),'2015-11-01-  
preview').primarySharedKey]"  
    }  
}  
}
```

The workspaceId and accountid elements are available from the **Settings** tile of the Log Analytics workspace, and the copy element is used to deploy it to multiple virtual machines. This resource is a child resource of the virtual machine resource, ensuring that this extension is executed whenever a virtual machine is provisioned or updated.

Security and Monitoring

The configuration related to workspace ID and account ID is shown next. The primary key is used as an account ID for configuring the agents using ARM templates:

The screenshot shows the Microsoft Operations Management Suite interface. The left sidebar has a 'Connected Sources' section highlighted. The main pane shows 'Windows Servers' with options to download the Windows Agent (64-bit or 32-bit) and enter a 'WORKSPACE ID' and 'PRIMARY KEY'. It also includes sections for 'SECONDARY KEY' and 'OMS Gateway'.

Search

The Log Analytics workspace provides search capabilities to search for specific log entries, export all telemetry data to Excel and/or PowerBI, and search query language similar to SQL.

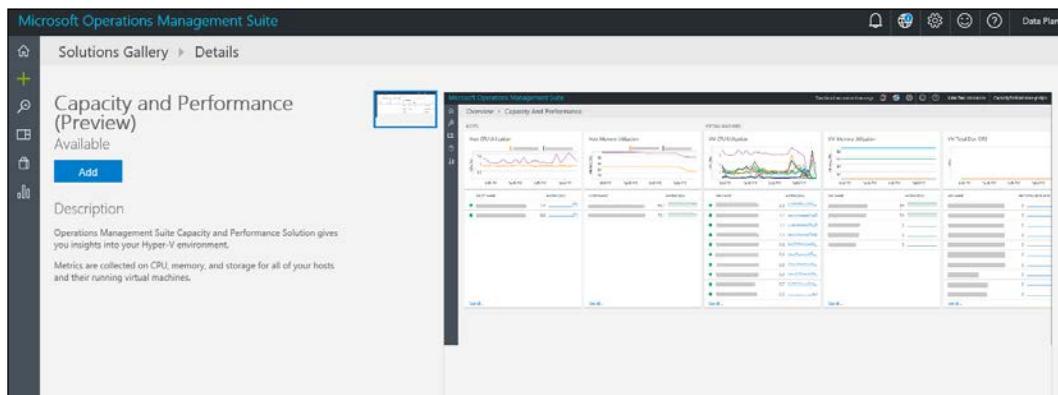
The **Log Search** screen is shown here:

The screenshot shows the Microsoft Operations Management Suite Log Search interface. It features a search bar at the top with placeholder text 'Begin searching here...'. Below it is a 'Let's start searching!' section with a sample query 'Type=Perf CounterName="% Processor Time"'. To the right, there is a preview of the results: 'measure avg(CounterValue) by Computer interval 30MINUTE'. The interface also includes sections for 'A few more queries to try' and 'For more information'.

Solutions

Solutions in Log Analytics are additional capabilities that can be added to the workspace, capturing additional telemetry data that is not captured by default. When these solutions are added to the workspace, appropriate management packs are sent to all the agents connected to the workspace in the context of configuring themselves for capturing solution-specific data from virtual machines and containers, and then start sending it to the Log Analytics workspace.

The following screenshot shows the solution gallery and the capacity and performance solution on the Log Analytics workspace. Clicking on any solution and subsequently clicking on the **Add** button adds the solution to the workspace, as follows:

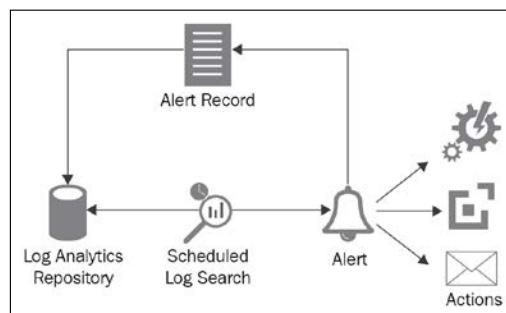


Azure provides lots of Log Analytics solutions for tracking and monitoring different aspects of environments and applications. At a minimum, a set of solutions that are generic and applicable to almost any environment should be added to the workspace:

- Capacity and performance
- Agent health
- Change tracking
- Containers
- Security and audit
- Update management
- Network performance monitoring

Alerts

Log Analytics allows us to generate alerts on the ingested data. It does so by running a pre-defined query composed of conditions on the incoming data. If it finds any records that fall within the ambit of the query results, it generates an alert. Log Analytics provides a highly configurable environment for determining the conditions for generating alerts, time windows in which the query should return the records, time windows in which the query should be executed, and actions to be taken when the query returns results as alerts:



The first step in configuring an alert is to create a saved search. A saved search is simply a search query against Log Analytics, as follows:

Save the query by giving it a name. After saving the query, click on the **Alert** button from the **Log Search** menu. It provides the user interface to define and add a new alert rule:

The screenshot shows the configuration of an alert rule. In the General section, the alert is named "eventsalert" and its description is "this alert is raised whenever events arrives". The Severity is set to Critical. The Search query is defined as "search * | where (Type == "Event")". The Schedule section specifies an alert frequency of "Check for this alert every 15 Minutes". It also includes a "Generate alert based on" section with "Number of results" set to "Greater than 2". A "Suppress alerts" checkbox is present with a note about reducing alert noise. The Actions section contains several configuration options: Email notification (Yes), Webhook (No), Runbook (Yes), Automation account (datacenterautomation), Select a runbook (TestRunbook), Run on (Hybrid worker), and ITSM Actions (Yes).

Within this single screen, all configurations related to an alert rule can be performed. You need to provide the **Name**, **Description**, **Severity**, and the query to be executed as part of the rule evaluation within their respective fields.

The time window specifies the data interval in which the query should be executed. In the screenshot, you can see that whenever the rule is executed, it processes data from the last 15 minutes.

The schedule section helps you configure the frequency of the rule execution. It answers the question, *how frequently should the query run?* In the previous screenshot, the rule is executed every 15 minutes. The time window should be more than the **Alert frequency**. The alerts can be further configured based on the number of results found. It is not necessary for an alert to be generated for every instance of data found based on the query. This data can be further accumulated and calculations performed on these groups of data can be used to raise alerts. The alerts can also be generated based on resource metric measurements. Metrics are generally associated with performance data provided by a resource such as CPU utilization, network usage, and memory availability. You can also create a time interval that should elapse before the action is executed. In this case, alerts are generated but the action is executed only after the configured interval has elapsed.

The **Actions** section allows us to configure things that should follow an alert. Generally, there should be a remedial and/or notification action. Log Analytics provides four different ways to create a new action. They can be combined in any way. An alert will execute any and all the following configured actions:

- **Email notification:** This sends an email to the configured recipients:

The screenshot shows a configuration dialog titled "Actions". Under the "Email notification" section, the "Yes" button is selected. Below it, the "Subject" field contains "Email subject" and the "Recipients" field contains "callritz@hotmail.com".

- **Webhook:** A webhook executes an arbitrary external process using a HTTP POST mechanism. For example, a REST API can be executed, or the Service Manager/ServiceNow APIs can be invoked to create a ticket:

The screenshot shows a configuration dialog titled "Webhook". The "Yes" button is selected. The "Webhook URL" field contains "Invoke webhook when alert fires". The "Include custom JSON payload" checkbox is checked. Below it, a text area shows a sample JSON payload:

```
Custom JSON payload with Alert parameters e.g.  
{ "incidentname": "#alertrulename",  
  "linktoresults": "#linktosearchresults",  
  "searchquery": "#searchquery"}
```

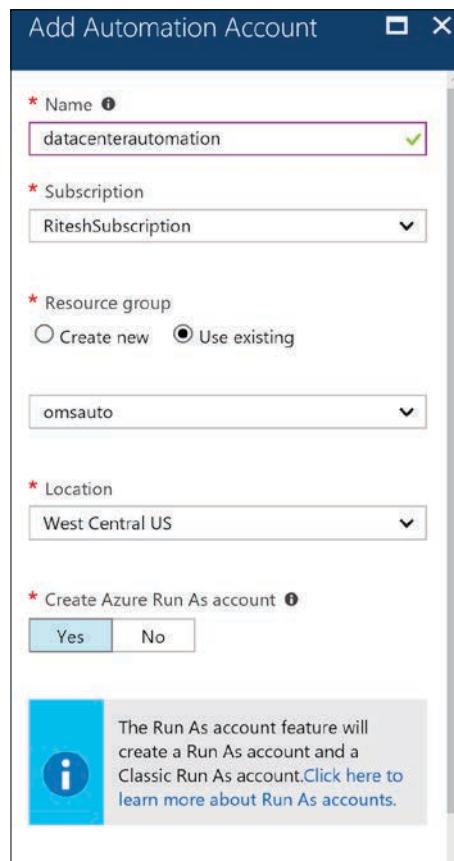
 At the bottom, a red message says "Url is invalid" and a "Test webhook" button is visible.

- **Runbooks:** This action executes Azure Automation runbooks. In the next section, we will see the entire process of executing an Azure Automation runbook.
- **ITSM actions:** ITSM solutions should be provisioned before using this option. It helps with connecting and sending information to ITSM systems.

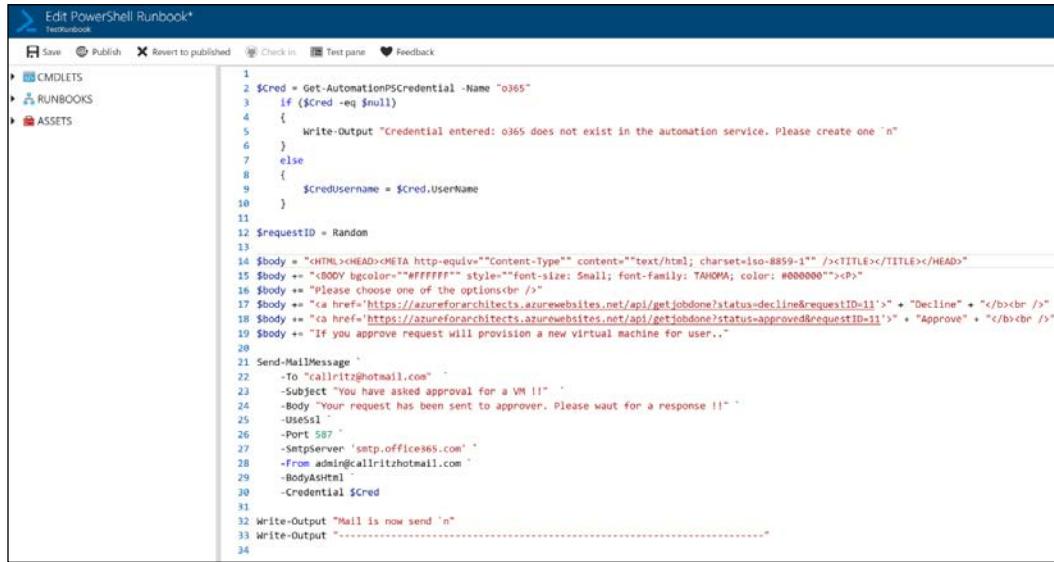
Executing runbooks on alerts

One of the actions provided by a Log Analytics alert is the execution of an Azure Automation runbook. This facility of executing runbooks on an alert provides the opportunity to act on the alert to remedy it and inform the relevant stakeholders using notifications.

1. The first step in executing a runbook in response to an alert is to create an Azure Automation account:

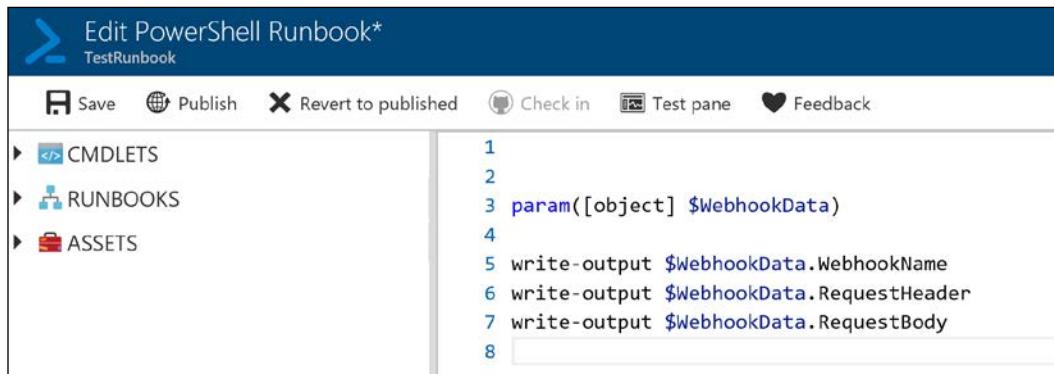


2. After the account has been created, create a runbook, as shown in the following screenshot, to prove that it can be executed as part of the alert generation. In this case, the runbook sends an email as part of the notification. It uses Azure Automation credentials to send an email using the O365 SMTP server. Users should have a valid O365 account before sending an email using Azure Automation:



```
1 $cred = Get-AutomationPSCredential -Name "O365"
2 if ($cred -eq $null)
3 {
4     Write-Output "Credential entered: O365 does not exist in the automation service. Please create one `n"
5 }
6 else
7 {
8     $credUsername = $cred.UserName
9 }
10
11 $requestID = Random
12
13 $body = "<HTML><HEAD><META http-equiv=""Content-Type"" content=""text/html; charset=iso-8859-1"" /><TITLE></TITLE></HEAD>
14 <BODY bgcolor=""#FFFFFF"" style=""font-size: Small; font-family: TAHOMA; color: #000000""><p>
15 $body += "Please choose one of the options<br />
16 $body += "<a href=""https://azureforarchitects.azurewebsites.net/api/getjobdone?status=decline&requestID=11">" + "Decline" + "</a><br />
17 $body += "<a href=""https://azureforarchitects.azurewebsites.net/api/getjobdone?status=approved&requestID=11">" + "Approve" + "</a><br />
18 $body += "If you approve request will provision a new virtual machine for user.."
19
20
21 Send-MailMessage `
22     -To "callritz@hotmail.com" `
23     -Subject "You have asked approval for a VM !!" `
24     -Body "Your request has been sent to approver. Please wait for a response !!" `
25     -UseSsl `
26     -Port 587 `
27     -SmtpServer 'smtp.office365.com' `
28     -From "admin@callritz@hotmail.com" `
29     -BodyAsHtml `
30     -Credential $cred
31
32 Write-Output "Mail is now send `n"
33 Write-Output "-----"
```

3. It has to be noted that this is just a demonstration. The runbook can also accept parameters and Log Analytics alerts and send a single object parameter. This parameter contains all the data pertaining to the source of the alert, details about the alert, and information that is available with Log Analytics:

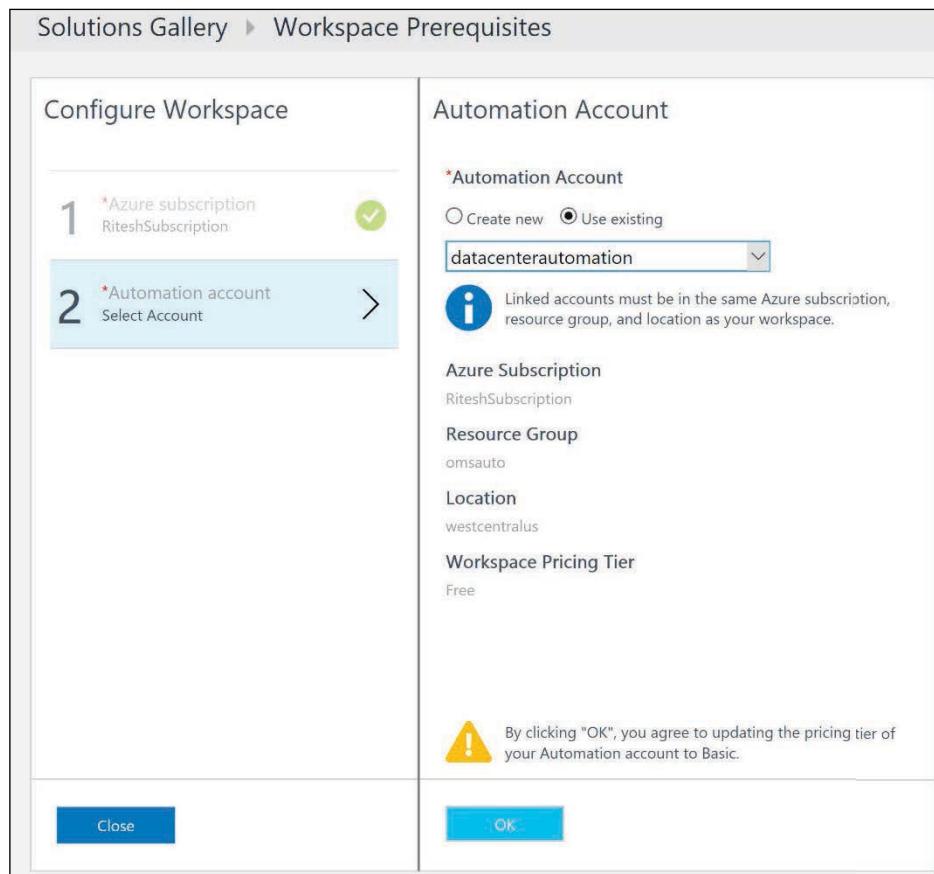


```
1
2
3 param([object] $WebhookData)
4
5 write-output $WebhookData.WebhookName
6 write-output $WebhookData.RequestHeader
7 write-output $WebhookData.RequestBody
8
```

4. The data is in a JSON format and a `ConvertFrom-JSON` cmdlet can be used to create PowerShell objects.
5. The next step is to configure a Log Analytics configuration so that it can connect to the Azure Automation account. For this, an **Automation & Control** solution needs to be enabled and deployed.
6. Clicking on this tile will navigate to the **Solutions Gallery** configuration window. Click on **Configure Workspace** to deploy it:



7. Select the newly created Azure **Automation Account** as part of the deployment of the solution:



8. After deploying the solution, navigate to the **Settings** window within the Log Analytics workspace and ensure that the Azure Automation settings shows details about the Azure **Automation Account**, as shown in the following screenshot. This ensures that the Log Analytics workspace is connected to the Azure **Automation Account**:

The screenshot shows the Azure Portal's Settings page. On the left, there is a navigation menu with the following items:

- Solutions
- Connected Sources
- Data
- Computer Groups
- Accounts** (highlighted in blue)
- Alerts
- Preview Features
- Upgrade Summary

On the right, there are three sections:

- Workspace Information**: Includes Manage Users.
- Azure Subscription & Data Plan**
- Automation Account** (highlighted in blue): Shows details for the Automation Account named "datacenterautomation". The details are as follows:
 - Subscription ID: 9755ffce-e94b-4332-9be8-1ade15e78909
 - Resource Group Name: omsauto
 - Automation Account Name: datacenterautomation

- Now, the runbook should be available while configuring the alert action runbook, as shown in the following screenshot:

The screenshot shows a modal dialog titled "Runbook". It has a "Yes" button (highlighted in blue) and a "No" button. Below the buttons, it says "Automation account" followed by the name "datacenterautomation". Underneath that, it says "Select a runbook" and shows a dropdown menu with "TestRunbook" selected. At the bottom, it says "Run on" with two options: "Azure" (highlighted in blue) and "Hybrid worker".

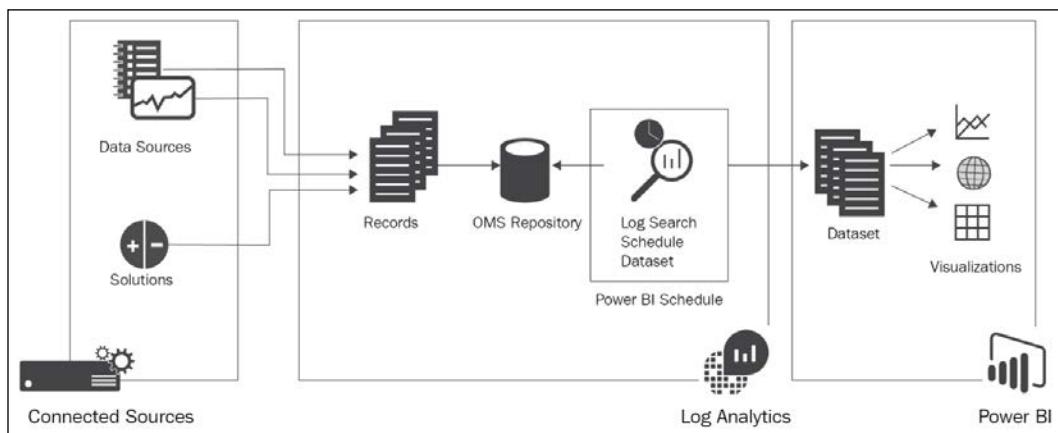
Integrating PowerBI

Gathering data and storing it in a central repository is an important aspect. However, there should be tools and utilities to process the data and generate insights out of it. PowerBI is a Microsoft-provided service specifically created to visualize and generate insights from raw data.

PowerBI can be enabled using the **Settings** menu, just like the configuration for Azure Automation. The connection to PowerBI should be made from the **Settings** menu. Once this connection is made, it can be used to send Log Analytics data to PowerBI.

Log Analytics provides two different ways to interact with PowerBI. First, it needs to be enabled from the **Settings** menu.

Like alerts, the PowerBI menu option is available from the top-level log search menu. Clicking on it allows us to configure the PowerBI connectivity. A scheduler runs periodically to execute search queries and send the resulting data to PowerBI. The data is stored as datasets in PowerBI and can be used to generate charts, reports, and dashboards as shown in the following diagram:



The other way to get data into PowerBI from Log Analytics is to use the **Power Query** language within PowerBI.

The exported **Power Query Formula Language (M Language)** can be used with Power Query in Microsoft Excel and PowerBI Desktop.

To use PowerBI Desktop, follow the next instructions to download and generate reports using queries:

1. Download PowerBI Desktop from <https://powerbi.microsoft.com/en-us/desktop/>.
2. In PowerBI Desktop, select **Get Data | Blank Query | Advanced Query Editor**.

3. Paste the following M Language script into the **Advanced Query Editor** and select **Done**. This will result in the execution of the query and bring the Log Analytics data into PowerBI. Provide a name and add a new dataset by clicking on the **Apply and Close** button in the PowerBI query editor:

```

let AnalyticsQuery =
let Source = Json.Document(Web.Contents("https://management.
azure.com/subscriptions/9755ffce-e94b-4332-9be8-1ade15e78909/
resourceGroups/omsauto/providers/Microsoft.OperationalInsights/
workspaces/data_centermonitoring/api/query?api-version=2017-01-01-
preview",
[Query=[#"query"="search * | where ( Type == ""Event"" )
",#"x-ms-app"="OmsAnalyticsPBI",#"timespan"="PT24H",#"prefer"="ai.
response-thinning=true"],Timeout=#duration(0,0,4,0)])),
TypeMap = #table(
{ "AnalyticsTypes", "Type" },
{
{
  { "string", Text.Type },
  { "int", Int32.Type },
  { "long", Int64.Type },
  { "real", Double.Type },
  { "timespan", Duration.Type },
  { "datetime", DateTimeZone.Type },
  { "bool", Logical.Type },
  { "guid", Text.Type }
}),
DataTable = Source[tables]{0},
Columns = Table.FromRecords(DataTable[columns]),
ColumnsWithType = Table.Join(Columns, {"type"}, TypeMap ,
{"AnalyticsTypes"}),
Rows = Table.FromRows(DataTable[rows], Columns[name]),
Table = Table.TransformColumnTypes(Rows, Table.
ToList(ColumnsWithType, (c) => { c{0}, c{3}}))
in
Table
in AnalyticsQuery

```

Summary

Security is always an important aspect for any deployment and solution. It has become much more important and relevant because of deployment onto the cloud. Moreover, there is an increasing threat of cyber attacks. In these circumstances, security has become a focal point for organizations. No matter the type of deployment or solution, whether it's IaaS, PaaS or SaaS, security is needed across all of them. Azure data centers are completely secure, and they have a dozen international security certifications. They are secure by default. They provide IaaS security resources, such as NSGs, network address translation, secure endpoints, certificates, key vaults, storage, and virtual machine encryption, and PaaS security features for individual PaaS resources. Security has a complete life cycle of its own and it should be properly planned, designed, implemented, and tested, just like any other application functionality.

Monitoring is an important architectural aspect of any solution. It is also the first step toward auditability. It enables operations to manage a solution, both reactively and proactively. It provides the necessary records for troubleshooting and fixing the issues that might arise from platforms and applications. There are many resources in Azure that are specific to implementing monitoring for Azure, other clouds, and on-premises data centers. Application Insights and Log Analytics are some of the most important resources in this regard. Needless to say, it is a must for making your solutions and products better by innovating based on insights from monitoring data.

In next chapter, we will be looking into a completely separate aspect of Azure related to the provisioning of resources using Azure Resource Manager Templates.

4

Cross-Subscription Deployments Using ARM Templates

Azure Resource Manager (ARM) templates are the preferred mechanism for provisioning resources and configuring them on Azure.

ARM templates help to implement a relatively new paradigm known as **Infrastructure as Code (IaC)**. ARM templates convert the infrastructure and its configuration into code. Converting infrastructure into code has numerous advantages. IaC brings a high level of consistency and predictability in deployments across environments. It also ensures that environments can be tested before going to production, and, finally, it gives a high level of confidence in the deployment process, maintenance, and governance.

The following topics will be covered in this chapter:

- ARM templates
- Deploying resource groups with ARM templates
- Deploying resources across subscriptions and resource groups
- Deploying cross-subscription and resource-group deployments using linked templates

ARM templates

A prominent advantage of IaC is that it can be version-controlled. It can be reused across environments, which provides a high degree of consistency and predictability in deployments. It also ensures that the impact and result of deploying an ARM template is the same no matter the number of times the template is deployed. This feature is known as an **idempotent template**.

ARM templates debuted with the introduction of the ARM specification and have been getting richer in features and more mature since then. It's important to understand that there's generally a feature gap of a few weeks to a couple of months between the actual resource configuration and the availability of this configuration in ARM templates.

The resource has its own configuration. This configuration can be affected in a multitude of ways, including using Azure PowerShell, Azure CLI, Azure SDKs, REST API, and ARM templates.

Each of these ways has its own development and release life cycle, which is different from the actual resource development. Let's try to understand this with the help of an example.

The Azure Databricks resource has its own cadence and development life cycle. The consumers of this resource have their own development life cycle, which is different from the actual resource development. If Databricks gets its first release on December 31, the Azure PowerShell cmdlets for it might not be available on the same date and they might get released January 31 of the next year; similarly, the availability of these features in the REST API and ARM templates might be around January 15th.

ARM templates are JSON-based documents that, when executed, invoke REST API on the Azure management plane and submit the entire document to it. The REST API has its own development life cycle, and the JSON schema for the resource has its own life cycle.

This means Azure development of a feature within a resource should happen in at least three different components before they can be consumed from ARM templates. These include the following:

- The resource itself
- The REST API for the resource
- The ARM template resource schema

Each resource in the ARM template has the `apiVersion` property. This property helps to decide the REST API version that should be used to provision and deploy the resource. The next diagram shows the flow of requests from the ARM template to resource APIs that are responsible for the creation, updation, and deletion of resources:



A resource configuration, such as a storage account in ARM template, looks as follows:

```
{
  "type": "Microsoft.Storage/storageAccounts",
  "apiVersion": "2017-06-01",
  "name": "[variables('storage2')]",
  "location": "[resourceGroup().location]",
  "kind": "Storage",
  "sku": {
    "name": "Standard_LRS"
  }
}
```

In this code listing, the availability of this schema for defining SKU is based in the development of the ARM template schema. The availability of the REST API and its version number is determined by `apiVersion`, which happens to be `2017-06-01`. The actual resource is determined by the `type` property, which has the following two parts:

- **Resource-provider namespace:** Resources in Azure are hosted within namespaces and related resources are hosted within the same namespace.
- **Resource type:** Resources are referenced using their type name.

In this case, the resource is identified by its provider name and type, which happens to be `Microsoft.Storage/storageaccounts`.

Until recently, ARM templates expected resource groups to be available prior to their deployment. They were also limited to deploying to a single resource group within a single subscription.

This means that, until recently, an ARM template could deploy all resources within a single resource group. This was the practice for years until Azure recently announced that a single ARM template could be used for deploying resources to multiple resource groups within the same subscription or across multiple subscriptions simultaneously. It's now possible to create resource groups as part of ARM templates, which means it's now possible to deploy resources in multiple regions into different resource groups.

Why would we need to create resource groups from within ARM templates, and why would we need to have cross-subscription and resource-group deployments simultaneously?

To appreciate the value of creating a resource group and cross-subscription deployments, we need to understand how deployments were carried out prior to these features being available.

To deploy an ARM template, a resource group is a prerequisite. Resource groups should be created prior to the deployment of a template. Developers use PowerShell, the Azure CLI, or the REST API to create resource groups and then initiate the deployment of ARM templates. This means that any end-to-end deployment consists of multiple steps. The first step is the provision of the resource group and the next step is the deployment of the ARM template to the newly created resource group. These steps could be executed using a single PowerShell script or individual steps from the PowerShell command line. The PowerShell script should implement exception handling, compensating code, rollback at a minimum until it's enterprise-ready. It is important to note that resource groups can be deleted from Azure and the next time the script runs they might be expected to be available. It would fail because it might assume that the resource group exists. In short, the deployment of the ARM template to a resource group should be an atomic step rather than multiple steps.

Compare this with the ability to create resource groups and its constituent resources together within the same ARM templates. Whenever you deploy the template, it ensures that the resource groups are created if they don't yet exist and continues to deploy resources to them after creation.

Let's also look at how these new features can help to remove some of the technical constraints related to disaster-recovery sites.

Prior to these features, if you had to deploy a solution that was designed with disaster recovery in mind, there were two separate deployments: one deployment for the primary region and another deployment for the secondary region. For example, if you were deploying an ASP.NET MVC application using App Services, you would create an app service and configure it for the primary region, and then you would conduct another deployment with the same template to another region with the same configuration.

With the availability of cross-subscription and resource-group deployment, it's possible to create the disaster-recovery site at the same time as the primary site. This eliminates two deployments and ensures that the same configuration can be used on multiple sites.

Deploying resource groups with ARM templates

In this section, an ARM template will be authored and deployed, which will create a couple of resource groups within the same subscription.

To use PowerShell to deploy templates that contain resource groups and cross-subscription resources, the latest version of PowerShell should be used. At the time of writing, Azure module version 6.6.0 was used:

PS C:\Users\rites> Get-Module azurerm.resources			
ModuleType	Version	Name	ExportedCommands
Script	6.6.0	AzureRM.Resources	{Add-AzureRmADGroupMember,

If the latest Azure module is not installed, it can be installed using the following command:

```
install-module -Name Azurerm -Force
```

Now, it's time to create an ARM template that will create multiple resource groups within same subscription. The code for the ARM template is shown next:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "resourceGroupInfo": {
      "type": "array"
    },
    "multiLocation": {
      "type": "array"
    }
  },
  "resources": [
    {
      "type": "Microsoft.Resources/resourceGroups",
      "name": "[parameters('resourceGroupInfo')[0].name]",
      "location": "[parameters('multiLocation')[0]]",
      "properties": {
        "tags": {
          "tag1": "value1"
        }
      }
    },
    {
      "type": "Microsoft.Resources/resourceGroups",
      "name": "[parameters('resourceGroupInfo')[1].name]",
      "location": "[parameters('multiLocation')[1]]",
      "properties": {
        "tags": {
          "tag2": "value2"
        }
      }
    }
  ]
}
```

```
    "location": "[parameters('multiLocation')[copyIndex()]]",
    "name": "[parameters('resourceGroupInfo')[copyIndex()]]",
    "apiVersion": "2018-05-01",
    "copy": [
        {
            "name": "allResourceGroups",
            "count": "[length(parameters('resourceGroupInfo'))]"
        },
        "properties": {}
    ],
    "outputs": {}
}
```

The first section of the code listing is about parameters that the ARM templates expect. These are mandatory parameters, and anybody deploying these templates should provide values for them. Array values must be provided for both the parameters.

The second major section is the `resources` JSON array, which can contain multiple resources. In this example, we are creating resource groups, so it is declared within the `resources` section. Resource groups are getting provisioned in a loop because of the use of the `copy` element. The `copy` element ensures that the resource is run for a specified number of times and creates a new resource in every iteration. If we send two values for the `resourceGroupInfo` array parameter, the length of the array would be two and the `copy` element will ensure that the `resourceGroup` resource is executed twice.

All resource names within a template should be unique for a resource type, so the `copyIndex` function is used to provide a current iteration number and assigned to the name to make it unique. Also, we want the resource groups to be created in different regions using distinct regions names sent as a parameter. The assignment of a name and location for each resource group is done using the `copyIndex` function.

The code for the `parameters` file is shown next. This code is pretty straightforward and provides array values to the two parameters expected by the previous template. The values in this file should be changed for all parameters according to the reader's environment:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/
deploymentParameters.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "resourceGroupInfo": {
```

```

        "value": [ "firstResourceGroup", "SeocndResourceGroup" ]
    },
    "multiLocation": {
        "value": [
            "West Europe",
            "East US"
        ]
    }
}
}

```

To deploy this template using PowerShell, log into Azure with valid credentials using the following command:

```
Login-AzureRmAccount
```

The valid credentials could be a user account or a service principal. Then, use a newly released `New-AzureRmDeployment` cmdlet to deploy the template. The deployment script is available in the `multipleResourceGroups.ps1` file:

```
New-AzureRmDeployment -Location "West Europe" -TemplateFile "c:\users\rites\source\repos\CrossSubscription\CrossSubscription\multipleResourceGroups.json" -TemplateParameterFile "c:\users\rites\source\repos\CrossSubscription\CrossSubscription\multipleResourceGroups.parameters.json" -Verbose
```

It's important to understand that the `New-AzureRMResourceGroupDeployment` cmdlet can't be used here. It can't be used because the scope of the `New-AzureRMResourceGroupDeployment` cmdlet is a resource group and it expects a resource group to be available as a prerequisite. For deploying resources at subscription level, Azure had released a new cmdlet that can work above the resource group scope. The new cmdlet, `new-AzureRmDeployment`, works at the subscription level.

The same template can also be deployed using the Azure CLI. Here are the steps to deploy it using the Azure CLI:

1. Use the latest version of the Azure CLI to create resource groups using ARM template. At the time of writing, version 2.0.43 was used for deployment, as shown here:

```
C:\Users\rites>az --version
azure-cli (2.0.43)
```

2. Log into Azure using the following command and select the right subscription for use:

```
C:\Users\rites>az login
```

3. If the login has access to multiple subscriptions, select an appropriate subscription using the following command:

```
C:\Users\rites>az account set --subscription xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

4. Execute the deployment using the following command. The deployment script is available in the `multipleResourceGroupsCLI.txt` file:

```
C:\Users\rites>az deployment create --location WestUS --template-file "c:\users\rites\source\repos\CrossSubscription\CrossSubscription\azuredeploy.json" --parameters @"c:\users\rites\source\repos\CrossSubscription\CrossSubscription\azuredeploy.parameters.json" --verbose
```

Deploying resources across subscriptions and resource groups

In the last section, resource groups were created as part of ARM templates, which is a relatively new Azure feature. Another new Azure feature is the provision of resources into multiple subscriptions simultaneously from a single deployment using a single ARM template. In this section, we will provide a new storage account into two different subscriptions and resource groups. The person deploying the ARM template would select one of the subscriptions as the base subscription, using which they would initiate the deployment and provision the storage account into the current and another subscription. The prerequisite for deploying this template is that the person doing the deployment should have access to at least two subscriptions and that they have contributor rights on these subscriptions. The code listing is shown here and is available in the `CrossSubscriptionStorageAccount.json` file within the accompanying code:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "storagePrefix1": {
```

```

    "type": "string",
    "defaultValue": "st01"
  ...
    "type": "string",
    "defaultValue": "rg01"
  },
  "remoteSub": {
    "type": "string",
    "defaultValue": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
  }
}
...
}
]
,
"outputs": {}
}
}
],
"outputs": {}
}

```

It is important to note that the names of the resource group used within the code should already be available in respective subscriptions. The code will throw an error if the resource groups are not available. Moreover, the names of the resource group should match exactly in the ARM template.

The code for deploying this template is shown next. In this case, we use `New-AzureRmResourceGroupDeployment`, because the scope of deployment is a resource group. The deployment script is available in the `CrossSubscriptionStorageAccount.ps1` file:

```

New-AzureRmResourceGroupDeployment -TemplateFile "<< path to your
CrossSubscriptionStorageAccount.json file >>" -ResourceGroupName
"<<provide your base subscription resource group name>>"
-storagePrefix1 <<provide prefix for first storage account>>
-storagePrefix2 <<provide prefix for first storage account>> -verbose

```

Another example of cross-subscription and resource-group deployments

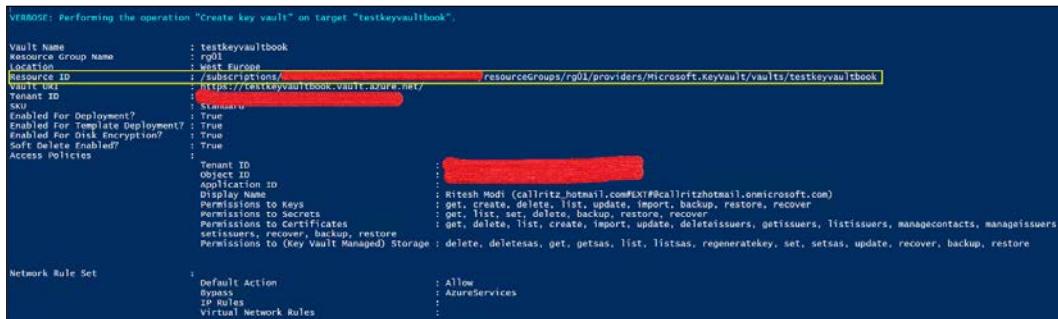
In this section, we create two storage accounts in two different subscriptions, resource groups, and regions from one ARM template and a single deployment. It uses the nested templates approach along with the `copy` element to provide different names and locations to these resource groups in different subscriptions.

However, before we can execute the next set of ARM templates, an Azure Key Vault instance should be provisioned as a prerequisite and a secret should be added to it. This is because the names of the storage accounts are retrieved from Azure Key Vault and passed as parameters to ARM templates for provisioning the storage account.

To provision Azure Key Vault using Azure PowerShell, the next set of commands can be executed. The code for the following commands is available in the `CreateKeyVaultAndSetSecret.ps1` file with the accompanying code:

```
New-AzureRmResourceGroup -Location <>replace with location of your key vault>> -Name <>replace with name of your resource group for key vault>> -verbose  
New-AzureRmKeyVault -Name <>replace with name of your key vault>> -ResourceGroupName <>replace with name of your resource group for key vault>> -Location <>replace with location of your key vault>> -EnabledForDeployment -EnabledForTemplateDeployment -EnabledForDiskEncryption -EnableSoftDelete -EnablePurgeProtection -Sku Standard -Verbose
```

Readers should note that the `ResourceID` value should be noted from the result of the `New-AzureRmKeyVault` cmdlet. This value will need to be replaced in the parameters file. See the following screenshot for details:



```
VERBOSE: Performing the operation "Create key vault" on target "testkeyvaultbook".  
Vault Name : testkeyvaultbook  
Resource Group Name : rg01  
Location : West Europe  
Resource ID : /subscriptions/.../resourceGroups/rg01/providers/Microsoft.KeyVault/vaults/testkeyvaultbook  
Vault URL : https://testkeyvaultbook.vault.azure.net/  
Tenant ID :  
SKU : Standard  
Enabled For Deployment? : True  
Enabled For Template Deployment? : True  
Enabled For Disk Encryption? : True  
Soft Delete Enabled? : True  
Access Policies :  
    Tenant ID :  
    Object ID :  
    Application ID :  
    Display Name : Ritesh Modi (callritz@hotmail.com#EXT#@callritz@hotmail.onmicrosoft.com)  
    Permissions to Keys : get, create, delete, list, update, import, backup, restore, recover  
    Permissions to Secrets : get, list, set, delete, backup, restore, recover  
    Permissions to Certificates : get, delete, list, create, import, update, deleteissuers, getissuers, listissuers, managecontacts, manageissuers, setissuers, recover, backup, restore  
    Permissions to (Key Vault Managed) Storage : delete, deletesas, get, getsas, list, listsas, regeneratekey, set, setsas, update, recover, backup, restore  
Network Rule Set :  
    Default Action : Allow  
    Bypass :  
    IP Rules :  
    Virtual Network Rules :  
...
```

Execute the following command to add a new secret to the newly created Azure Key Vault instance:

```
Set-AzureKeyVaultSecret -VaultName <>replace with name of your key vault>> -Name <>replace with name of your secret>> -SecretValue $(ConvertTo-SecureString -String <>replace with value of your secret>> -AsPlainText -Force ) -Verbose
```

The code listing is available in the `CrossSubscriptionNestedStorageAccount.json` file from within the accompanying code:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/
deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "hostingPlanNames": {
      "type": "array",
      "minLength": 1
    },
    ...
    "type": "Microsoft.Resources/deployments",
    "name": "deployment01",
    "apiVersion": "2017-05-10",
    "subscriptionId": "[parameters('subscriptions')[copyIndex()]]",
    "resourceGroup": "[parameters('resourceGroups')[copyIndex()]]",
    "copy": {
      "count": "[length(parameters('hostingPlanNames'))]",
      "name": "mywebsites", "mode": "Parallel"
    },
    ...
    "kind": "Storage",
    "properties": {
    }
  }
}
...
}
```

Here's the code for the parameters file. It is available in the `CrossSubscriptionNestedStorageAccount.parameters.json` file:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/
deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "hostingPlanNames": {
      ...
      "storageKey": {
        "reference": {
          "keyVault": { "id": "<>replace it with the value of Key vault
ResourceId noted before>>" }
        }
      }
    }
  }
}
```

```
        "secretName": "<<replace with the name of the secret available  
in Key vault>>"  
    }  
}  
}  
}  
}
```

Here's the PowerShell code for deploying the previous template. The deployment script is available in the `CrossSubscriptionNestedStorageAccount.ps1` file:

```
New-AzureRmResourceGroupDeployment -TemplateFile "c:\users\rites\source\repos\CrossSubscription\CrossSubscription\CrossSubscriptionNestedStorageAccount.json" -ResourceGroupName rg01 -TemplateParameterFile "c:\users\rites\source\repos\CrossSubscription\CrossSubscription\CrossSubscriptionNestedStorageAccount.parameters.json" -Verbose
```

Deploying cross-subscription and resource-group deployments using linked templates

The previous example used nested templates to deploy to multiple subscriptions and resource groups. In the next example, we'll deploy multiple app service plans in separate subscriptions and resource groups using linked templates. The linked templates are stored in Azure Blob Storage, which is protected using policies. This means that only the holder of the storage account key or a valid shared access signature can access this template. The access key is stored in Azure Key Vault and is accessed from the `parameters` file using references under the `storageKey` element. Readers should upload the `website.json` file within a container of Azure Blob Storage. The `website.json` file is a linked template responsible for provisioning an App Service plan and an App Service. The file is protected using the **Private (no anonymous access)** policy, as shown in the next screenshot. A privacy policy ensures that anonymous access is not allowed. I have created a container named `armtemplates` and set it with a private policy:

IDENTIFIER	START TIME	EXPIRY TIME	PERMISSIONS
armtemplates			No results

This file can only be accessed using the SAS keys. The SAS keys can be generated from the Azure portal for a storage account using the **Shared access signature** item on the left menu as shown next. Readers should click on the **Generate SAS and connection string** button to generate the SAS token. It is to be noted that an SAS token is displayed once and not stored with Azure. So, copy it and store it somewhere so that it can be uploaded to Azure Key Vault. The next screenshot shows the generation of the SAS token:

We will use the same key vault that was created in the previous section. We just have to ensure that there are two secrets available within the key vault. The first secret is `StorageName` and the other one is `StorageKey`. The commands to create these secrets in the key vault are as follows:

```
Set-AzureKeyVaultSecret -VaultName "testkeyvaultbook" -Name  
"storageName" -SecretValue $(ConvertTo-SecureString -String  
"uniquename" -AsPlainText -Force ) -Verbose  
  
Set-AzureKeyVaultSecret -VaultName "testkeyvaultbook" -Name  
"storageKey" -SecretValue $(ConvertTo-SecureString -String "?sv=2018-  
03-28&ss=bfqt&srt=sco&sp=rwdlacup&se=2019-03-29T21:51:03Z&st=2019-  
01-30T14:51:03Z&spr=https&sig=gTynGhj20er6pDl7Ab%2Bpc29WO3%2BJhvi%2  
Bff%2F6rHYWp4g%3D" -AsPlainText -Force ) -Verbose
```

Readers are advised to change the names of the key vault and the secret key value based on their storage account.

After ensuring that the key vault has the necessary secrets, the ARM template file code can be used for deploying the nested template across subscriptions and resource groups.

The ARM template code is available in the `CrossSubscriptionLinkedStorageAccount.json` file and is also shown here. Readers are advised to change the value of the `templateUrl` variable within this file. It should be updated with a valid Azure Blob Storage file location:

```
{  
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/  
deploymentTemplate.json#",  
    "contentVersion": "1.0.0.0",  
    "parameters": {  
        "hostingPlanNames": {  
            "type": "array",  
            "minLength": 1  
        },  
        "...": {  
            "type": "Microsoft.Resources/deployments",  
            "name": "fsdfsdf",  
            "apiVersion": "2017-05-10",  
            "subscriptionId": "[parameters('subscriptions')[copyIndex())]]",  
            "resourceGroup": "[parameters('resourceGroups')[copyIndex()]]",  
            "copy": {  
                "count": "[length(parameters('hostingPlanNames'))]"  
            }  
        }  
    }  
}
```

```

        "name": "mywebsites",
        "mode": "Parallel"
    ...
]
}

```

The code for the `parameters` file is shown next. Readers are advised to change the values of the parameters including the `resourceid` of the key vault and the secret name. The names of App Services should be unique, otherwise the template will fail to deploy. The code for the `parameters` file is available in the `CrossSubscriptionLinkedStorageAccount.parameters.json` code file:

```

{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/
deploymentParameters.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "hostingPlanNames": {
            "value": [ "firstappservice", "secondappservice" ]
        ...
        "storageKey": {
            "reference": {
                "keyVault": { "id": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourceGroups/keyvaluedemo/providers/Microsoft.KeyVault/
vaults/forsqlvault1" },
                "secretName": "storageKey"
            }
        }
    }
}

```

Here's the command to deploy the template. The deployment script is available in the `CrossSubscriptionLinkedStorageAccount.ps1` file:

```

New-AzureRmResourceGroupDeployment -TemplateFile "c:\users\rites\source\repos\CrossSubscription\CrossSubscription\CrossSubscriptionLinkedStorageAccount.json" -ResourceGroupName <>replace with the base subscription resource group name </>
-TemplateParameterFile "c:\users\rites\source\repos\CrossSubscription\CrossSubscription\CrossSubscriptionLinkedStorageAccount.parameters.json" -Verbose

```

Summary

The ability to deploy resources using a single deployment to multiple subscriptions, resource groups, and regions provides better ability to deploy, reduce bugs in deployment, and provide advanced benefits, such as creating disaster-recovery sites and high availability.

In the next chapter, we will focus on creating modular ARM templates, an essential skill for architects who really want to take their ARM templates to the next level. The next chapter will show various ways to design ARM templates and create reusable and modular ARM templates.

5

ARM Templates Modular Design and Implementation

We know that there are multiple ways to author an **Azure Resource Manager (ARM)** template. It is quite easy to author one that provisions all of the necessary resources in Azure using Visual Studio and Visual Studio Code. A single ARM template can consist of all the required resources for a solution on Azure. This single ARM template could be as small as a few resources, or it could be a larger one consisting of many resources.

While authoring a single template consisting of all resources is quite tempting, it is advisable to plan an ARM template implementation divided into multiple smaller ARM templates beforehand, so that future troubles related to them can be avoided.

In this chapter, we will look at how to write ARM templates in a modular way so that they can evolve over a period of time with minimal involvement in terms of changes and effort in testing and deployment.

However, before writing modular templates, it is best to understand the problems solved by writing them in a modular fashion.

The following topics will be covered in this chapter:

- Problems with a single template
- Understanding nested and linked deployment
- Linked templates
- Nested templates
- Free-flow configurations
- Known configurations

Problems with the single template

On the surface, it might not sound like a single large template consisting of all resources will have problems, but there are issues that could arise in the future. Let's understand the issues that might come up with single large templates.

Reduced flexibility in changing templates

Using a single large template with all resources makes it difficult to change it in future. With all dependencies, parameters, and variables in a single template, changing the template can take a considerable amount of time compared to smaller templates. The change could have an impact on other sections of the template, which might go unnoticed, as well as introducing bugs.

Troubleshooting large templates

Large templates are difficult to troubleshoot. This is a known fact. The larger the number of resources in a template, the more difficult it is to troubleshoot the template. A template deploys all resources in it, and finding a bug involves deploying the template quite often. Developers would have reduced productivity while waiting for the completion of template deployment.

Also, deploying a single template is more time-consuming than smaller templates. Developers have to wait for resources containing errors to be deployed before taking any action.

Dependency abuse

The dependencies between resources also tend to become more complex in larger templates. It is quite easy to abuse the usage of the dependson feature in ARM templates because of the way they work. Every resource in a template can refer to all its prior resources rather than building a tree of dependencies. ARM templates do not complain if a single resource is dependent on all other resources in the ARM template, even though those other resources might have inter-dependencies within themselves. This makes changing ARM templates bug prone and, at times, it is not even possible to change them.

Reduced agility

Generally, there are multiple features teams in a project, with each owning their own resources in Azure. These teams will find it difficult to work with a single ARM template because a single developer should be updating them.

Updating a single template with multiple teams might induce conflict and difficult-to-solve merges. Having multiple smaller templates can enable each team to author their own piece of an ARM template.

No reusability

If you have a single template, then that's all that you have, and using this template means deploying all resources. There is no possibility, out of the box, to select individual resources without some maneuvering, such as adding conditional resources. A single large template loses reusability since you take all the resources or none.

Knowing that single large templates have so many issues, it is good practice to author modular templates so that we get the benefits of the following:

- Multiple teams can work on their templates in isolation
- Templates can be reused across projects and solutions
- Templates are easy to debug and troubleshoot

Understanding the Single Responsibility Principle

The **Single Responsibility Principle** is one of the core principles of the SOLID design principles. It states that a class or code segment should be responsible for a single function and that it should own that functionality completely. The code should change or evolve only if there is a functional change or bug in the current functionality and not otherwise. This code should not change because of changes in some other component or code that is not part of the current component.

Applying the same principle to ARM templates helps to create templates that have the sole responsibility of deploying a single resource or functionality instead of deploying all resources and a complete solution.

Using this principle will help you create multiple templates, each responsible for a single resource or a smaller group of resources rather than all resources.

Faster troubleshooting and debugging

Each template deployment is a distinct activity within Azure and is a separate instance consisting of inputs, outputs, and logs. When multiple templates are deployed for deploying a solution, each template deployment have separate log entries along with its input and output descriptions. It is much easier to isolate bugs and troubleshoot issues using these independent logs from multiple deployments compared to a single large template.

Modular templates

When a single large template is decomposed into multiple templates in which each smaller template takes care of the resources in it, and those resources are solely owned, maintained, and the responsibility of the template containing it, we can say we have modular templates. Each template within these templates follows the Single Responsibility Principle.

Before understanding how to divide a large template into multiple smaller reusable templates, it is important to understand the technology behind creating smaller templates and how to compose them to deploy complete solutions.

Deployments resources

ARM provides a facility to link templates. Although we have already gone through linked templates in detail, I will mention it here to help you understand how linking templates helps in achieving modularity, composition, and reusability.

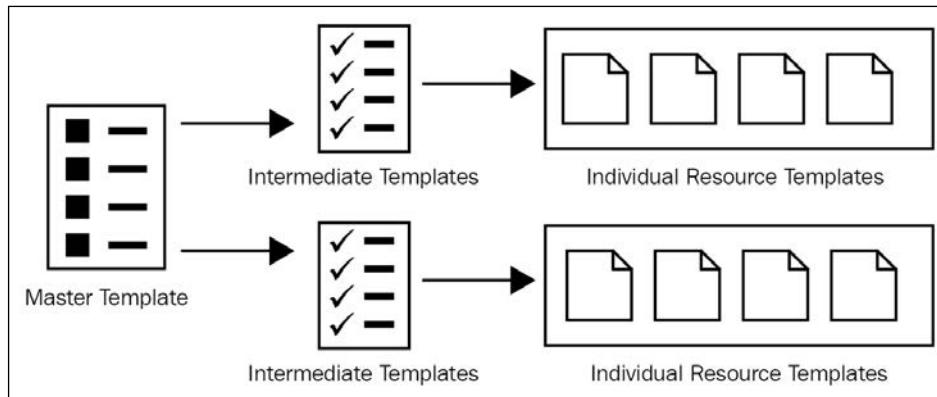
ARM templates provide specialized resources known as **deployments**, which are available from the `Microsoft.Resources` namespace. A deployments resource in an ARM template looks very similar to the code segment as follows:

```
"resources": [
  {
    "apiVersion": "2017-05-10",
    "name": "linkedTemplate",
    "type": "Microsoft.Resources/deployments",
    "properties": {
      "mode": "Incremental",
      <nested-template-or-external-template>
    }
  }
]
```

This template is self-explanatory, and the two most important configurations in template resource are the type and properties. The type here refers to the deployments resource rather than any specific Azure resource (storage, VM, and so on) and the properties specify the deployment configuration, including a linked template deployment or a nested template deployment.

However, what does the deployments resource do? The job of a deployments resource is to deploy another template. Another template could be an external template in a separate ARM template file, or it could be a nested template. It means that it is possible to invoke other templates from a template, just like a function call.

There can be nested levels of deployments in ARM templates. What this means is that a single template can call another template, and the called template can call another template, and this can go on for five levels of nested callings. In the next diagram, there is a master template that internally calls two other templates using two distinct deployments resources, and those two deployments resources call further templates that deploy other resources. This parent-child relationship can go up to five levels deep:



Linked templates

Linked templates are templates that invoke external templates. External templates are stored in different ARM template files. An example of linked templates is as follows:

```

"resources": [
  {
    "apiVersion": "2017-05-10",
    "name": "linkedTemplate",
    "type": "Microsoft.Resources/deployments",
    "properties": {
      "mode": "Incremental",
      "templateLink": {
        "uri": "https://mystorageaccount.blob.core.windows.net/
AzureTemplates/newStorageAccount.json",
        "contentVersion": "1.0.0.0"
      },
      "parametersLink": {
        "uri": "https://mystorageaccount.blob.core.windows.net/
AzureTemplates/newStorageAccount.parameters.json",
        "contentVersion": "1.0.0.0"
      }
    }
  }
]
  
```

```
        }
    }
}
]
```

Important additional properties in this template compared to the previous template are `templateLink` and `parametersLink`. Now, `templateLink` refers to the actual URL of the location of the external template file, and `parametersLink` is the URL location for the corresponding parameters file. It is important to note that the caller template should have access rights to the location of the called template. For example, if the external templates are stored in Azure Blob Storage, which is protected by keys, then the appropriate SAS keys must be available to the caller template to be able to access the linked templates.

It is also possible to provide explicit inline parameters instead of the `parametersLink` value, as shown here:

```
"resources": [
{
    "apiVersion": "2017-05-10",
    "name": "linkedTemplate",
    "type": "Microsoft.Resources/deployments",
    "properties": {
        "mode": "Incremental",
        "templateLink": {
            "uri": "https://mystorageaccount.blob.core.windows.net/AzureTemplates/newStorageAccount.json",
            "contentVersion": "1.0.0.0"
        },
        "parameters": {
            "StorageAccountName": { "value": "parameters('StorageAccountName')" }
        }
    }
}]
```

Nested templates

Nested templates are a relatively new feature in ARM templates compared to external linked templates.

Nested templates do not define resources in external files. The resources are defined within the caller template itself and within the deployments resource, as shown here:

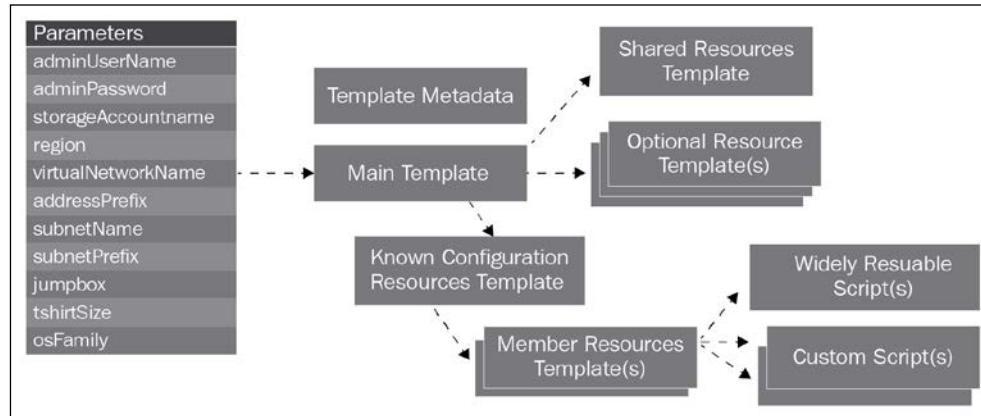
```

"resources": [
  {
    "apiVersion": "2017-05-10",
    "name": "nestedTemplate",
    "type": "Microsoft.Resources/deployments",
    "properties": {
      "mode": "Incremental",
      "template": {
        "$schema": "https://schema.management.azure.com/schemas/2015-
          01-01/deploymentTemplate.json#",
        "contentVersion": "1.0.0.0",
        "resources": [
          {
            "type": "Microsoft.Storage/storageAccounts",
            "name": "[variables('storageName')]",
            "apiVersion": "2015-06-15",
            "location": "West US",
            "properties": {
              "accountType": "Standard_LRS"
            }
          }
        ]
      }
    }
  ]
]
  
```

In this code segment, we can see that the storage account resource is nested within the original template as part of the deployments resource. Instead of using the `templateLink` and `parametersLink` attributes, a `resources` array is used to create multiple resources as part of a single deployment. The advantage of using a nested deployment is that resources within a parent can be used to reconfigure them by using their names. Usually, a resource with a name can exist only once within a template. Nested templates allow us to use them within the same template and ensure that all templates are self-sufficient rather than being stored separately, and they may or may not be accessible to those external files.

Now that we understand the technology behind modular ARM templates, how should we divide a large template into smaller templates?

There are multiple ways a large template can be decomposed into smaller templates. Microsoft recommends the following pattern for the decomposition of ARM templates:



When we decompose a large template into smaller templates, there is always the main template, which is used for deploying the solution. This main or master template internally invokes other nested or linked templates and they, in turn, invoke other templates, and finally, the templates containing Azure resources are deployed.

The main template can invoke a known configuration resource template, which, in turn, will invoke templates comprising Azure resources. The known configuration resource template is specific to a project or solution and it does not have many reusable factors associated with it. The member resource templates are reusable templates invoked by the known configuration resource template.

Optionally, the master template can invoke shared resource templates and other resource templates if they exist.

It is important to understand known configurations. Templates can be authored as known configurations or as free-flow configurations.

Free-flow configurations

ARM templates can be authored as generic templates where most, if not all, of the values assigned to variables are obtained as parameters. This allows the person using the template to pass any value they deem necessary to deploy resources in Azure. For example, the person deploying the template could choose a virtual machine of any size, any number of virtual machines, and any configuration for its storage and networks. This is known as free-flow configuration, where most of the configuration is allowed and the templates come from the user instead of being declared within the template.

There are challenges with this kind of configuration. The biggest one is that not all configurations are supported on every Azure region and data center in Azure. The templates will fail to create resources if those resources are not allowed to be created in specific locations or regions. Another issue with free-flow configuration is that users can provide any value they deem necessary and a template will honor them, thereby increasing both the cost and deployment footprint even though they are not completely required.

Known configurations

Known configurations, on the other hand, are specific pre-determined configurations for deploying an environment using ARM templates. These pre-determined configurations are known as **T-shirt sizing configurations**. Similar to the way a T-shirt is available in a pre-determined configuration such as small, medium, and large, ARM templates can be pre-configured to deploy a small, medium, or large environment depending on the requirements. This means that users cannot determine any random custom size for the environment, but they can choose from various provided options, and ARM templates executed during runtime will ensure that an appropriate configuration of the environment is provided.

So, the first step in creating a modular ARM template is deciding on the known configurations for an environment.

As an example, here is the configuration of a data center deployment on Azure:

T-shirt size	ARM template configuration
Small	Four virtual machines with 7 GB of memory along with four CPU cores
Medium	Eight virtual machines with 14 GB of memory along with eight CPU cores
Large	16 virtual machines with 28 GB of memory along with eight CPU cores

Now that we know the configurations, we can create modular ARM templates.

There are two ways to write modular ARM templates:

- **Composed templates:** Composed templates link to other templates. Examples of composed templates are master and intermediate templates.
- **Leaf-level templates:** Leaf-level templates are templates that contain a single Azure resource.

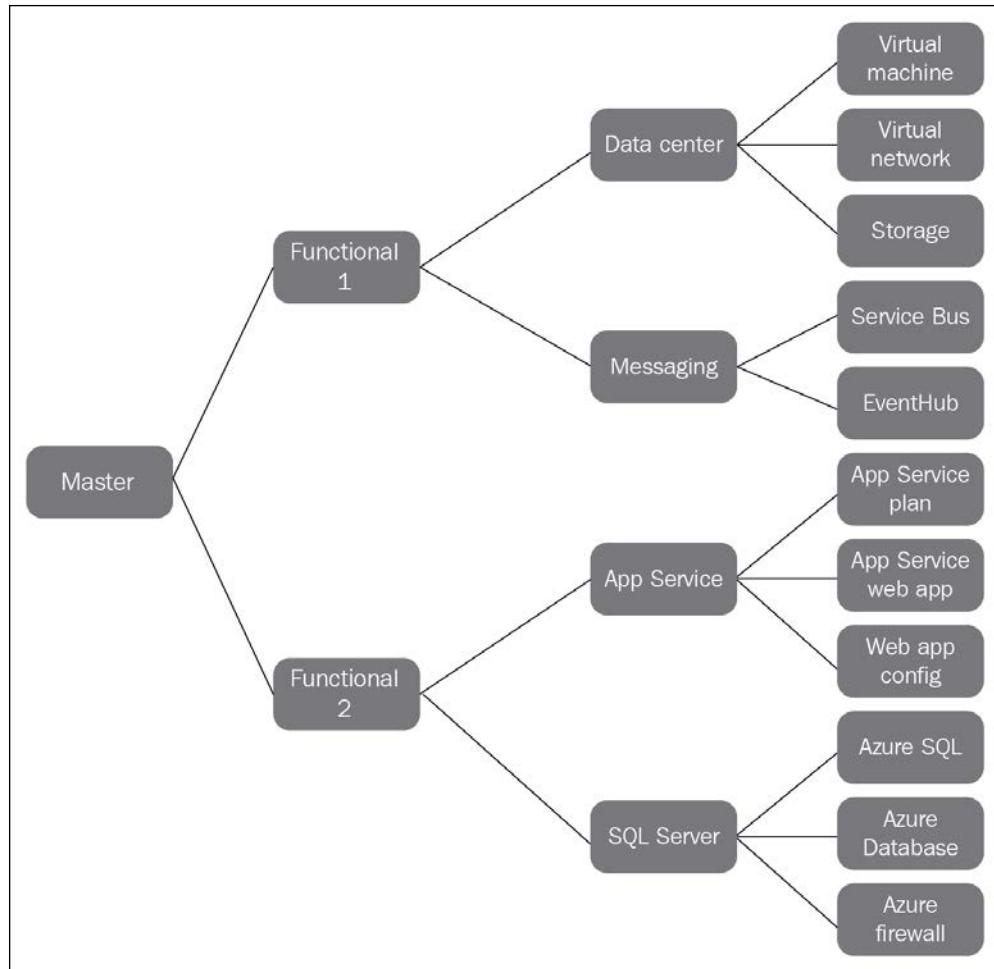
ARM templates can be divided into modular templates based on the following:

- Technology
- Functionality

An ideal way to decide on the modular method to author an ARM template is as follows:

- Define resource-or leaf-level templates consisting of single resources. In the upcoming diagram, the extreme right templates are leaf-level templates. Within the diagram, virtual machines, virtual network, storage, and others in the same column represent leaf-level templates.
- Compose environment specific templates using the leaf-level templates. These environment-specific templates provide an Azure environment, such as a SQL Server environment, an App Service environment, or a data center environment. Let's drill down a bit more into this topic. Let's take the example of an Azure SQL environment. To create an Azure SQL environment, multiple resources are needed. At a bare minimum, a logical SQL Server, a SQL database, and a few SQL firewall resources should be provisioned. All these resources are defined in individual templates at leaf level. These resources can be composed together in a single template that has the capability to create an Azure SQL environment. Anybody wanting to create an SQL environment can use this composed template. The diagram shown next has Data center, Messaging, and App Service as environment-specific templates.
- Create templates with higher abstraction composing multiple environment-specific templates into solutions. These templates are composed of environment-specific templates that were created in the previous step. For example, to create an e-commerce inventory solution that needs an App Service environment and a SQL environment, two environment templates, App Service and SQL Server, can be composed together. The diagram shown next has **Functional 1** and **Functional 2** templates, which are composed of child templates.
- Finally, a master template should be created, which should be composed of multiple templates where each template is capable of deploying a solution.

The preceding steps for creating a modular designed template can be easily be understood by means of a diagram, as follows:



Now, let's implement a part of the functionality shown in the previous diagram. In this implementation, we will provide a virtual machine with a script extension using a modular approach. The custom script extension deploys Docker binaries and prepares a container environment on a Windows Server 2016 virtual machine.

Now, we are going to create a solution using ARM templates using a modular approach. As mentioned before, the first step is to create individual resource templates. These individual resource templates will be used to compose additional templates capable of creating an environment. These templates will be needed to create a virtual machine. All ARM templates shown here are available in the accompanying chapter code. The names and code of these templates are as follows:

- Storage.json
- virtualNetwork.json
- PublicIPAddress.json
- NIC.json
- VirtualMachine.json
- CustomScriptExtension.json

First, let's look at the code for the `Storage.json` template. This template provides a storage account, which every virtual machine needs for storing its OS and data disk files:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "storageAccountName": {
            "type": "string",
            "minLength": 1
        },
        "storageType": {
            "type": "string",
            "minLength": 1
        },
        ...
    },
    "outputs": {
        "resourceDetails": {
            "type": "object",
            "value": "[reference(parameters('storageAccountName'))]"
        }
    }
}
```

Next, let's look at the code for the public IP address template. A virtual machine that should be accessible over the internet needs a public IP address resource assigned to its network interface card. Although exposing a virtual machine to the internet is optional, this resource might get used for creating a virtual machine. The following code is available in the `PublicIPAddress.json` file:

```
{  
    "$schema": "https://schema.management.azure.com/schemas/2015-01-  
01/deploymentTemplate.json#",  
    "contentVersion": "1.0.0.0",  
    "parameters": {  
        "publicIPAddressName": {  
            "type": "string",  
            "minLength": 1  
        },  
        "publicIPAddressType": {  
            "type": "string",  
            "minLength": 1  
        }  
    },  
    "outputs": {  
        "resourceDetails": {  
            "type": "object",  
            "value": "[reference(parameters('publicIPAddressName'))]"  
        }  
    }  
}
```

Next, let's look at the code for the virtual network. Virtual machines on Azure need a virtual network for communication. This template will be used to create a virtual network on Azure with a pre-defined address range and subnets. The following code is available in the `virtualNetwork.json` file:

```
{  
    "$schema": "https://schema.management.azure.com/schemas/2015-01-  
01/deploymentTemplate.json#",  
    "contentVersion": "1.0.0.0",  
    "parameters": {  
        "virtualNetworkName": {  
            "type": "string",  
            "minLength": 1  
        }  
    },  
    "subnets": {  
        "type": "array",  
        "value": [  
            {  
                "name": "Subnet1",  
                "addressPrefix": "10.0.0.0/24",  
                "cidr": "10.0.0.0/24",  
                "size": 256  
            }  
        ]  
    },  
    "resourceLocation": {  
        "type": "string",  
        "value": "[parameters('virtualNetworkName')]"  
    }  
}
```

```
        "minLength": 1
    }
    ...
    "subnets": [
        {
            "name": "[parameters('subnetName')]",
            "properties": {
                "addressPrefix": "[parameters('subnetPrefix')]"
            }
        }
    ]
},
"outputs": {
    "resourceDetails": {
        "type": "object",
        "value": "[reference(parameters('virtualNetworkName'))]"
    }
}
}
```

Next, let's look at the code for the network interface card. A virtual network card is needed by a virtual machine to connect to a virtual network and to accept and send requests to and from the internet. The following code is available in the `NIC.json` file:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-
01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "nicName": {
            "type": "string",
            "minLength": 1
        },
        "publicIpReference": {
            "type": "string",
            "minLength": 1
        },
        ...
    },
    "resources": [
        {
            "type": "Microsoft.Network/networkInterfaces",
            "name": "[resourceId(subscription().subscriptionId,resourceGroup().name,
'Microsoft.Network/publicIPAddresses', parameters('publicIpReferen-
ce'))]",
            "properties": {
                "vnetRef": "[resourceId(subscription().
subscriptionId,resourceGroup().name, 'Microsoft.Network/
virtualNetworks', parameters('virtualNetworkReference'))]" ,
                ...
            }
        }
    ]
}
```

```

        "subnet1Ref": "[concat(variables('vnetRef'), '/subnets/', parameters('subnetReference'))]"
    },
    ...
    "id": "[variables('subnet1Ref')]"
}
}
]
}
],
"outputs": {
    "resourceDetails": {
        "type": "object",
        "value": "[reference(parameters('nicName'))]"
    }
}
}

```

Next, let's look at the code for creating a virtual machine. Each virtual machine is a resource in Azure, and note that this template has no reference to storage, network, public IP addresses, or other resources created earlier. This reference and composition will happen later in this section using another template. The following code is available in the `VirtualMachine.json` file:

```

{
    "$schema": "https://schema.management.azure.com/schemas/2015-01
01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "vmName": {
            "type": "string",
            "minLength": 1
        },
        ...
        "imageOffer": {
            "type": "string",
            "minLength": 1
        },
        "windowsOSVersion": {
            "type": "string",
            "minLength": 1
        },
        ...
    }
}

```

```
    "outputs": {
        "resourceDetails": {
            "type": "object",
            "value": "[reference(parameters('vmName'))]"
        }
    }
}
```

Next, let's look at the code for creating a custom script extension. This resource executes a PowerShell script on a virtual machine after it is provisioned. This resource provides an opportunity to execute post-provisioning tasks in Azure virtual machines. The following code is available in the `CustomScriptExtension.json` file:

```
{
    "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "VMName": {
            "type": "string",
            "defaultValue": "sqldock",
            "metadata": {
                ...
                "commandToExecute": "[concat('powershell -ExecutionPolicy Unrestricted -file docker.ps1')]"
            },
            "protectedSettings": {
            }
        }
    },
    "outputs": {
    }
}
```

Next, we'll look at the custom script extension PowerShell code that prepares the Docker environment. Please note that a virtual machine reboot might happen while executing the PowerShell script, depending on whether the Windows containers feature is already installed or not. The following script installs the NuGet package, the `DockerMsftProvider` provider, and the Docker executable. The `docker.ps1` file is available with the accompanying chapter code:

```
#  
# docker.ps1  
#
```

```
Install-PackageProvider -Name Nuget -Force -ForceBootstrap  
-Confirm:$false  
Install-Module -Name DockerMsftProvider -Repository PSGallery -Force  
-Confirm:$false -verbose  
Install-Package -Name docker -ProviderName DockerMsftProvider -Force  
-ForceBootstrap -Confirm:$false
```

All the previously seen linked templates should be uploaded to a container within an Azure Blob Storage account. This container can have private access policy applied as we saw in the previous chapter; however, for this example, we will set the access policy as container. This means these linked templates can be accessed without the need of an SAS token.

Finally, let's focus on writing the master template. Within the master template, all the linked templates are composed together to create a solution – to deploy a virtual machine and execute a script within it. The same approach can be used for creating other solutions such as providing a data center consisting of multiple inter-connected virtual machines. The following code is available in the `Master.json` file:

```
{  
    "$schema": "https://schema.management.azure.com/  
schemas/2015-01-01/deploymentTemplate.json#",  
    "contentVersion": "1.0.0.0",  
    "parameters": {  
        "storageAccountName": {  
            "type": "string",  
            "minLength": 1  
        },  
        "...  
    },  
    "subnetName": {  
        "type": "string",  
        "minLength": 1  
    },  
    "subnetPrefix": {  
        "type": "string",  
        "minLength": 1  
    },  
    "...  
    "windowsOSVersion": {  
        "type": "string",  
        "minLength": 1  
    },  
    "vhdStorageName": {  
        "type": "string",  
        "minLength": 1  
    },  
}
```

```
"vhdStorageContainerName": {
    "type": "string",
    "minLength": 1
    ...[concat('https://',parameters('storageAccountName'),'armfiles.
blob.core.windows.net/',variables('containerName'), '/Storage.json')]",
        "contentVersion": "1.0.0.0"
    },
    "parameters": {
        "storageAccountName": {
            "value": "[parameters('storageAccountName')]"
        },
        "storageType": {
            "value": "[parameters('storageType')]"
        },
        "resourceLocation": {
            "value": "[resourceGroup().location]"
        ...
    }
    "outputs": {
        "resourceDetails": {
            "type": "object",
            "value": "[reference('GetVM').outputs.resourceDetails.value]"
        }
    }
}
```

The master templates invoke the external templates and also co-ordinate inter-dependencies among them.

The external templates should be available at a well-known location so that the master template can access and invoke them. In this example, the external templates are stored in the Azure Blob Storage container and this information was passed to the ARM template by means of parameters.

The external templates in Azure Blob Storage could be access protected by setting up access policies. The command used to deploy the master template is shown next. It might look like a complex command, but a majority of values are used as parameters. Readers are advised to change the value of these parameters before running it. The linked templates have been uploaded to a storage account named st02gvwlfdcsm5suwe within the armtemplates container. The resource group should be created if it does not currently exist. The first command is used to create a new resource group in the **West Europe** region:

```
New-AzureRmResourceGroup -Name "testvmrg" -Location "West Europe"
-Verbose
```

The rest of the parameter values are needed for configuring each resource. The storage account name and the dnsNameForPublicIP value should be unique within Azure:

```
New-AzureRmResourceGroupDeployment -Name "testdeploy1"
-ResourceGroupName testvmrg -Mode Incremental -TemplateFile "C:\chapter 05\Master.json" -storageAccountName "st02gvwldcxm5suwe"
-storageType "Standard_LRS" -publicIPAddressName "uniipaddname"
-publicIPAddressType "Dynamic" -dnsNameForPublicIP
"azureforarchitectsbook" -virtualNetworkName vnetwork01
-addressPrefix "10.0.1.0/16" -subnetName "subnet01" -subnetPrefix
"10.0.1.0/24" -nicName nic02 -vmSize "Standard_DS1" -adminUsername
"sysadmin" -adminPassword $(ConvertTo-SecureString -String
sysadmin@123 -AsPlainText -Force) -vhdStorageName oddnewuniqueacc
-vhdStorageContainerName vhds -OSDiskName mynewvm -vmName
vm10 -windowsOSVersion 2012-R2-Datacenter -imagePublisher
MicrosoftWindowsServer -imageOffer WindowsServer -containerName
armtemplates -Verbose
```

Summary

ARM templates are the preferred means of provisioning resources in Azure. They are idempotent in nature, bringing consistency, predictability, and reusability to environment creation. In this chapter, we looked at how to create a modular ARM template. It is important for teams to spend quality time designing ARM templates in an appropriate way, so that multiple teams can work on them together. They are highly reusable and require minimal changes to evolve.

The next chapter will move on to a different and very popular strand of technology known as serverless within Azure. Azure Functions is one of the major serverless resources on Azure, and this will be covered in complete depth, including Durable Functions.

6

Designing and Implementing Serverless Solutions

Serverless is one of the hottest buzzwords in technology these days, and everyone wants to ride this bandwagon. Serverless brings a lot of advantages in overall computing, software development processes, infrastructure, and technical implementation. There is a lot going on in the industry: at one end of the spectrum is **Infrastructure as a Service (IaaS)**, and at the other is serverless. In between are **Platform as a Service (PaaS)** and containers. I have met many developers and it seems to me that there exists some level of confusion among them about IaaS, PaaS, containers, and serverless computing. Also, there is much confusion about use cases, applicability, architecture, and implementation for the serverless paradigm. Serverless is a new paradigm that is changing not only technology, but also culture and processes within organizations.

In this chapter, we will explore serverless by covering the following topics:

- Serverless
- Azure Functions
- Serverless events
- Azure Event Grid
- Serverless workflows
- Logic Apps
- Creating a complete solution using serverless integration

Serverless

Serverless refers to a deployment model in which users are responsible for only their application code and configuration. Customers of serverless do not have to bother about the underlying platform and infrastructure and can concentrate on solving their business problems by writing code.

Serverless does not mean that there are no servers. Code and configuration will always need some server to run on. However, from the customer's perspective, an application can be practically serverless. Serverless means that the customer does not see the server at all. They do not care about the underlying platform and infrastructure. They do not need to manage or monitor anything. Serverless provides an environment that can scale up and down, in and out, automatically, without the customers even knowing about it. All operations related to platforms and infrastructures happen behind the scenes. Customers are provided with performance-related **service-level agreements (SLAs)** and Azure ensures that it meets those SLAs irrespective of the server demands.

Customers are required to only bring in their code; it is the responsibility of the cloud provider to provide the infrastructure and platform needed to run the code.

The evolution of serverless

Before we understand serverless architecture and implementation, it is important to understand its history and how it has evolved. In the beginning, there were physical servers. Although users had complete control over physical servers, there were a lot of disadvantages, as follows:

- Long gestation periods between the ordering and the actual deployment of the server
- Capital-intensive in nature
- Waste of resources
- Lower return on investment
- Difficult to scale out and up

A natural evolution from physical servers was virtualization. Virtualization refers to the creation of virtual machines on top of physical servers and deploying applications within them. Virtual machines provide several advantages:

- No need to procure physical hardware
- Comparatively easier to create newer virtual machines

- Complete isolation of environments
- Lower costs compared to physical servers

However, virtualization had its own set of disadvantages:

- Still dependent on the physical procurement of a server for scaling out after a number of virtual machine instances.
- Still costly because of human and hardware dependence.
- Wastage of compute resources – each virtual machine runs a complete operating system within it.
- High maintenance costs.

The next evolution was IaaS from cloud providers. Instead of procuring and managing data centers and infrastructures, the strategy was to create virtual machines, storage, and networks in the cloud. The cloud provides software-defined infrastructure services and hides all the details related to physical servers, networks, and storage. This had some advantages:

- No capital expenditure, only operational expenses. Functions are charged based on consumption model, instead of fixed cost (although there is an App Service model based on fixed cost).
- No gestation time to create new virtual machines – new virtual machines can be provisioned within minutes rather than hours.
- Flexible size of virtual machines.
- Easier scaling up and out of virtual machines.
- Completely secure.

Virtual machines in the cloud do have some disadvantages:

- Requires active monitoring and auditing.
- Requires active maintenance of virtual machines.
- Scalability, high availability, and performance of virtual machines should be managed by users – any degradation and subsequent improvement is the user's responsibility.
- A costly option because users pay for the entire machine, whether it is used or not.

The cloud also provides another pattern for deploying applications, popularly known as PaaS. PaaS provides abstraction from the underlying infrastructure in terms of virtual machines, virtual networks, and storage on the cloud. It provides a platform and an ecosystem where users do not need to know a thing about the infrastructure; they can simply deploy their application on these platforms. There are definite advantages of using PaaS compared to IaaS, but there are still better options. The main disadvantages of PaaS are the following:

- PaaS applications are deployed on virtual machines behind the scenes and the payment model is not granular; it is still at the deployment level.
- PaaS still demands monitoring for scaling out and in.
- Users still need to identify the requirements for their platform. There are limited options available for different types of platform. Azure exclusively provided a Windows-based environment until recently, when it began to offer Linux as well. Moreover, the installation of packages, utilities, and software is the responsibility of its users.

A new paradigm then emerged, known as containers, which is primarily made popular by Docker. Containers provide a lightweight, isolated, and secure environment that has all the benefits of virtual machines, minus their disadvantages. They do not have a dedicated operating system and instead rely on a base server operating system. Containers come in both IaaS and PaaS patterns. Containers provide many advantages:

- Faster provisioning of environments
- Consistent and predictable creation of environments
- Eases the creation of microservices architectures
- A rich ecosystem with advanced services from Kubernetes, Swarm, and DC/OS

Containers do have a few disadvantages; they are as follows:

- They require active monitoring and auditing.
- They require active maintenance.
- The scalability, high availability, and performance of containers should be managed by orchestration tools like Kubernetes. These orchestration tools need extension deployments and skills to be managed.

Serverless, by definition, is a deployment paradigm. It can be deployed on virtual machines, as well as on containers. To get the best out of serverless, they should be deployed onto containers to take advantage of their faster creation and tear-down features. This will have a direct impact on scalability and high availability of serverless platform, and will also be much quicker compared to a virtual machine.

Principles of serverless technology

Serverless technology is based on the following principles:

- **Lower cost:** Cost is based on the actual consumption of computing resources and power. There is no cost if there is no consumption.
- **Unlimited scalability:** With serverless, there is no need to perform either manual or automatic scaling operations. All scaling up and down is handled directly by the Azure platform. The platform ensures that enough servers are provided to support the serverless deployment, whether there are a few hundred users or millions of users; this scaling happens transiently, without any interference or knowledge of the organization deploying the serverless components.
- **Event-driven:** Serverless functions should be able to execute based on certain events happening. The event should trigger the execution of the function. In other words, serverless should allow functions to decouple themselves from other functions and instead rely on the firing of certain events in which they are interested.
- **Single responsibility:** Serverless functions should implement a single functionality and responsibility and should do that well. Multiple responsibilities should not be coded or implemented within a single function.
- **Execute quickly:** Serverless functions should not take a long time to complete a job. They should be able to execute quickly and return back.

The advantages of Azure Functions

Serverless computing is a relatively new paradigm that helps organizations convert large functionalities into smaller, discrete, on-demand functions that can be invoked and executed through automated triggers and scheduled jobs. They are also known as **Functions as a Service (FaaS)**, in which organizations can focus on their domain challenges instead of the underlying infrastructure and platform. FaaS also helps in devolving solution architectures into smaller, reusable functions, thereby increasing return on investments.

There is a plethora of serverless compute platforms available. Some of the important ones are listed here:

- Azure Functions
- AWS Lambda
- IBM OpenWhisk
- Iron.io
- Google Cloud Functions

In fact, every few days it feels like there is a new framework being introduced, and it is becoming increasingly difficult for enterprises to decide on the framework that works best for them. Azure provides a rich serverless environment known as Azure Functions, and I would like to point out a few features that it supports:

- Numerous ways to invoke a function – manual, on schedule, or based on an event
- Numerous types of binding support
- Ability to run functions synchronously as well as asynchronously
- Execute functions based on multiple types of triggers
- Ability to run both long and short-duration functions
- Ability to use proxy features to use different function architectures
- Multiple usage models including consumption, as well as the App Service model
- Able to author functions using multiple languages such as JavaScript, Python, and C#
- Authorization based on OAuth
- Multiple authentication options including Azure AD, Facebook, Twitter, and other identity providers
- Easily configure inbound and outbound parameters
- Visual Studio integration for authoring of Azure Functions
- Massive parallelism

FaaS

Azure provides FaaS. These are serverless implementations from Azure. With Azure Functions, code can be written in any language the user is comfortable with and Azure Functions will provide a runtime to execute it. Based on the language chosen, an appropriate platform is provided for users to bring their own code. Functions are a unit of deployment and can automatically be scaled out and in. When dealing with functions, users cannot view the underlying virtual machines and platform, but Azure Functions provides a small window to see them via the **Kudu** console.

There are two main components of Azure Functions:

- Azure Functions runtime
- Azure Functions binding and triggers

Azure Functions runtime

The core of Azure Functions is its Azure runtime. The precursor to Azure Functions was Azure WebJobs. The code for Azure WebJobs also forms the core for Azure Functions. There are additional features and extensions added to Azure WebJobs to create Azure Functions. The functions runtime is the magic behind making functions work. Azure Functions are hosted within Azure App Service. Azure App Service loads the Azure runtime and either waits for an external event to occur or for any HTTP requests. On arrival of a request or the occurrence of a trigger, App Service loads the incoming payload, reads the function's `function.json` file to find the function's bindings and trigger, maps the incoming data to incoming parameters, and invokes the function with parameter values. Once the function completes its execution, the value is again passed back to the Azure Functions runtime by way of an outgoing parameter defined as a binding in the `function.json` file. The function runtime returns the values to the caller. The Azure Functions runtime acts as the glue that enables the entire performance of functions.

Azure Functions bindings and triggers

If the Azure Functions runtime is the brain of Azure Functions, then Azure Functions bindings and triggers are the heart. Azure Functions promote loose-coupling and high-cohesion between services using triggers and bindings. The application implements code using imperative syntax for incoming and outgoing parameters and return values. This generally results in hardcoding the incoming parameters. Since Azure Functions should be capable of invoking any function defined, they implement a generic mechanism to invoke functions by means of triggers and bindings.

Binding refers to the process of creating a connection between the incoming data and the Azure Function, mapping the data types. The connection could be a single direction from the runtime to the Azure Functions, from the Azure Functions to runtime for return values, or could be multi-directional – the same binding can transmit data between the Azure runtime and Azure Functions. Azure Functions use a declarative way to define bindings.

Triggers are a special type of binding through which functions can be invoked based on external events. Apart from invoking the function, triggers also pass the incoming data, payload, and metadata to the function.

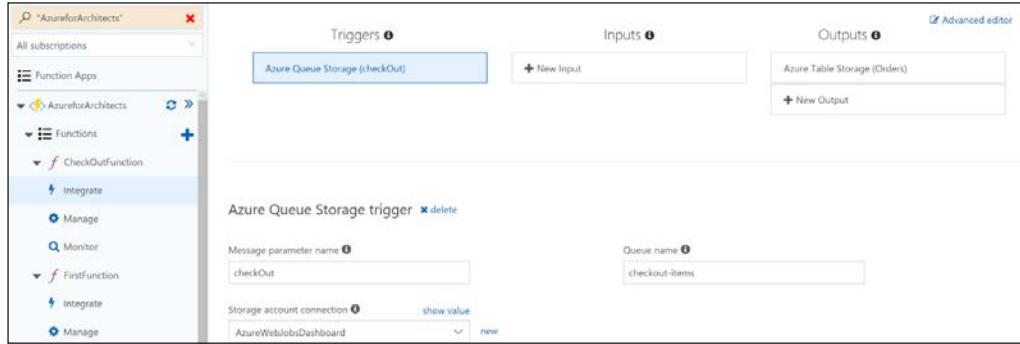
Bindings are defined in the `function.json` file as follows:

```
{  
  "bindings": [  
    {  
      "name": "checkOut",  
      "type": "queueTrigger",  
      "direction": "in",  
      "queueName": "checkout-items",  
      "connection": "AzureWebJobsDashboard"  
    },  
    {  
      "name": "Orders",  
      "type": "table",  
      "direction": "out",  
      "tableName": "OrderDetails",  
      "connection": "<>Connection to table storage account>>"  
    }  
,  
    "disabled": false  
  ]  
}
```

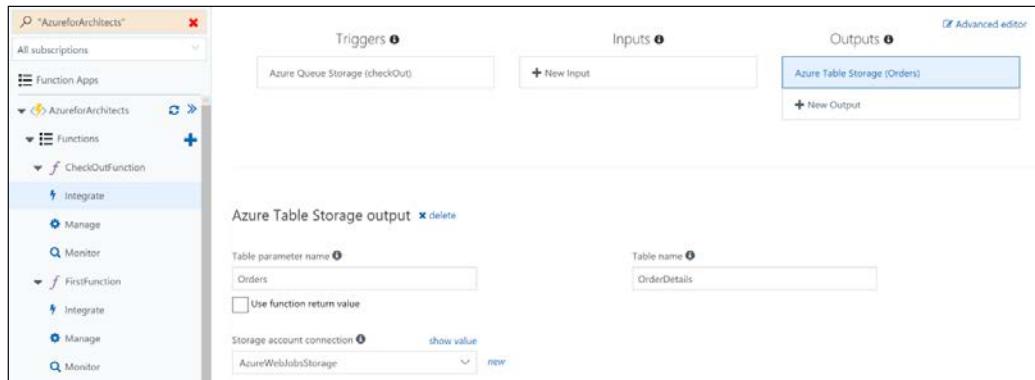
In this example, a trigger is declared that invokes the function whenever there is a new item in the storage queue. The type is `queueTrigger`, the direction is inbound, `queueName` is `checkout-items`, and details about the target storage account connection and table name are also shown. All these values are important for the functioning of this binding. The `checkOut` name can be used within the function's code as a variable.

Similarly, a binding for the return values is declared. Here, the return value is named `Orders` and the data is the output from Azure Functions. The binding writes the return data into Azure table storage using the connection string provided.

Both bindings and triggers can be modified and authored using the Integrate tab in Azure Functions. Behind the scenes, the function.json file is updated. The checkOut trigger is declared, as shown here:



The Orders output is shown next:



The authors of Azure Functions do not need to write any plumbing code to get data from multiple sources. They just decide the type of data expected from the Azure runtime. This is shown in the next code segment. Notice that the checkout is available as a string to the function. Functions provide multiple different types to be able to send to a function. For example, a queue binding can provide the following:

- Plain old simple object (POCO)
- String
- Byte[]
- CloudQueueMessage

The author of the function can use any one of these datatypes, and the Azure Functions runtime will ensure that a proper object is sent to the function as a parameter:

```
using System;
public static void Run(string checkOut, TraceWriter log)
{
    log.Info($"C# Queue trigger function processed: { checkOut }");
}
```

It is also important to know that, in the previous screenshots, the storage account names are `AzureWebJobsStorage` and `AzureWebJobsDashboard`. These are keys that are defined within the Azure Functions `appSettings` setting.



For more information on Azure bindings and triggers, refer to <https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-storage-queue>.



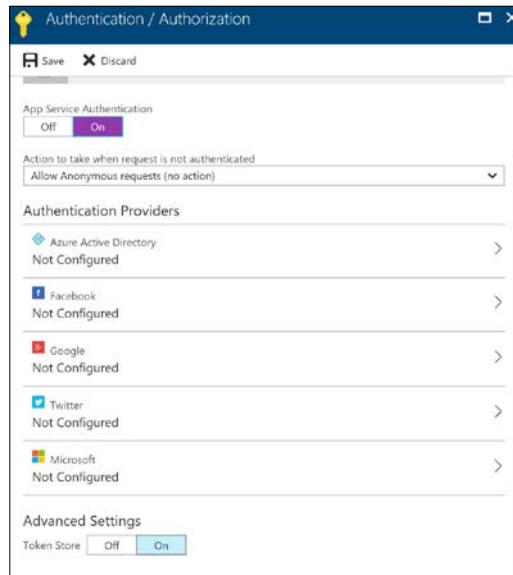
Monitoring

Complete log information is provided for each request or trigger in the **Monitor** tab of Azure Functions. This helps in identifying issues and auditing any risks, bugs, or exceptions in Azure Functions:

The screenshot shows the Azure Functions monitor interface for the "FirstFunction" under the "AzureforArchitects" subscription. The left sidebar lists "Function Apps" and "Functions" (with "Firstfunction" selected). The main area displays monitoring metrics: "Success count since Sep 1st" (1) and "Error count since Sep 1st" (0). Below these are sections for "Invocation log" (showing a single successful invocation of "Firstfunction" 2 hours ago) and "Invocation details" (listing parameters like req, log, binder, context, logger, and \$return). A "Logs" section is also present at the bottom.

Authentication and authorization

Azure Functions rely on Azure App Service for their authentication needs. App Service has rich authentication features where clients can use OpenConnectID for authentication and OAuth for authentication. Users can be authenticated using Azure AD, Facebook, Google, Twitter, or Microsoft accounts:



Azure Functions that are based on HTTP can also incorporate the use of keys, which should be sent along with HTTP requests:

NAME	VALUE	ACTIONS
default	Click to show	
_master	Click to show	
default	Click to show	

The following keys are available from the **Manage** tab in Azure Functions:

- **Function keys:** Allows authorization to individual functions. These keys should be sent as part of the HTTP request header.
- **Host keys:** Allows authorization to all functions within a function app. These keys should be sent as part of the HTTP request header.
- **Default keys:** Used when using function and host keys. There is an additional host key named **_master** that helps in administrative access to the Azure Functions runtime API.

Azure Functions configuration

Azure Functions provide configuration options at multiple levels. They provide configuration for the following:

- The platform itself
- The Function App Services

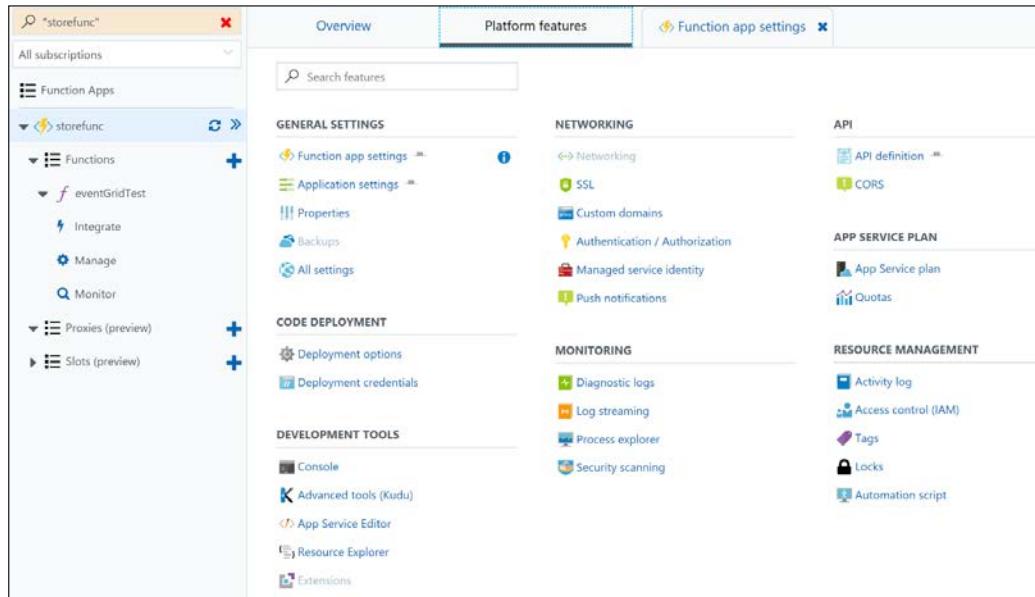
These settings affect every function contained by them. More information about these settings are available at <https://docs.microsoft.com/en-us/azure/azure-functions/functions-how-to-use-azure-function-app-settings>.

Platform configuration

Azure Functions are hosted within Azure App Service, so they get all of its features. Diagnostic and monitoring logs can be configured easily using platform features. Furthermore, App Service provides options for assigning SSL certificates, using a custom domain, authentication, and authorization as part of its networking features.

Although customers are not concerned about the infrastructure, operating system, filesystem, and platform on which the functions actually execute, Azure Functions provides the necessary tooling to peek within the underlying system and make changes. The console and the Kudu console are the tools used for this purpose. They provide a rich editor to author Azure Functions and edit their configuration.

Azure Functions, just like App Service, allows for the storage of configuration information within the `web.config` application setting section, which can be read on demand:



App Service function settings

These settings affect all functions. Application settings can be managed here. Proxies in Azure Functions can be enabled and disabled. We will discuss proxies later in this chapter. They also help in changing the edit mode of a function application and the deployment to slots:

The screenshot shows the Azure portal's 'Function App' settings for a 'storefunc' application. The 'Function app settings' tab is active. Key configuration items shown include:

- Daily Usage Quota (GB-Sec)**: An input field for entering usage quota values.
- Application settings**: A link to manage application settings.
- Runtime version**: Set to 1.0.11184.0 (~1).
- Proxies (preview)**: A switch to enable Azure Functions Proxies, currently set to Off.
- Function app edit mode**: A switch to change the edit mode, currently set to Read/Write.
- Slots (preview)**: A switch to enable deployment slots, currently set to Off.
- Host Keys (All functions)**: A table listing host keys for '_master' and 'default'. Each entry has 'Click to show' under VALUE, and 'Actions' columns for Copy, Renew, and Revoke.

Azure Functions cost plans

Azure Functions are based on the Azure App Service and provide a better costing model for users. There are two cost models:

- **Consumption plan**: This is based on the actual consumption and execution of functions. This plan calculates the cost based on the compute usage during the actual consumption and execution of the function. If a function is not executed, there is no cost associated with it. However, it does not mean that performance is compromised in this plan. Azure Functions will automatically scale out and is based on demand, to ensure basic minimum performance levels are maintained. A function execution is allowed 10 minutes for its completion.

- **App service plan:** This plan provides functions with complete dedicated virtual machines behind the scenes, and so the cost is directly proportional to the cost of the virtual machine and its size. There is a cost associated with this plan, even if functions are not executed at all. Function code can run for as long as necessary. There is no time limit. Within the App Service plan, the function runtime goes idle if not used within a few minutes and can be awoken only using an HTTP trigger. There is an **Always On** setting that can be used to prevent the function runtime from going idle. Scaling is either manual or based on auto-scale settings.

Azure Functions use cases

There are many valid use cases for using and implementing Azure Functions:

- **Implementing microservices:** Azure Functions help in breaking down large applications into smaller, discreet functional code units. Each unit is treated independently of the other and evolves in its own life cycle. Each such code unit has its own compute, hardware, and monitoring requirements. Each function can be connected to all other functions. These units are woven together by orchestrators to build complete functionality. For example, in an e-commerce application, there can be individual functions (code units), each responsible for listing catalogs, recommendations, categories, subcategories, shopping carts, checkouts, payment types, payment gateways, shipping addresses, billing addresses, taxes, shipping charges, cancellations, returns, emails, SMS, and so on. Some of these functions are brought together to create use cases for e-commerce applications, such as product browsing and checkout flow.
- **Integration between multiple endpoints:** Azure Functions can build overall application functionality by integrating multiple functions. The integration can be based on the triggering of events or it could be on a push basis. This helps in decomposing large monolithic applications into small components.
- **Data processing:** Azure Functions can be used for processing incoming data in batches. They can help in processing data in multiple formats, such as XML, CSV, JSON, and TXT. They can also run conversion, enrichment, cleaning, and filtering algorithms. In fact, multiple functions can be used, each doing either conversion or enrichment, cleaning or filtering. Azure Functions can also be used to incorporate advanced cognitive services, such as **optical character recognition** (OCR), computer vision, and image manipulation and conversion.

- **Integrating legacy applications:** Azure Functions can help in integrating legacy applications with newer protocols and modern applications. Legacy applications might not be using industry-standard protocols and formats. Azure Functions can act as a proxy for these legacy applications, accept requests from users or other applications, convert the data into a format understood by a legacy application, and talk to it on protocols it understands. This opens a world of opportunity for integrating and bringing old and legacy applications into the mainstream portfolio.
- **Scheduled jobs:** Azure Functions can be used to execute continuously or periodically for certain application functions. These application functions can perform tasks such as periodically taking backups, restoring, running batch jobs, exporting and importing data, and bulk emailing.
- **Communication gateways:** Azure Functions can be used in communication gateways when using notification hubs, SMS, and email, for instance.

Types of Azure Functions

Azure Functions can be categorized into three different types:

- **On-demand functions:** These are functions that are executed when they are explicitly called or invoked. Examples of such functions include HTTP-based functions and webhooks.
- **Scheduled functions:** These functions are like timer jobs and execute functions on fixed intervals.
- **Event-based functions:** These functions are executed based on external events. For example, uploading a new file to Azure blob storage generates an event that could start the execution of Azure Functions.

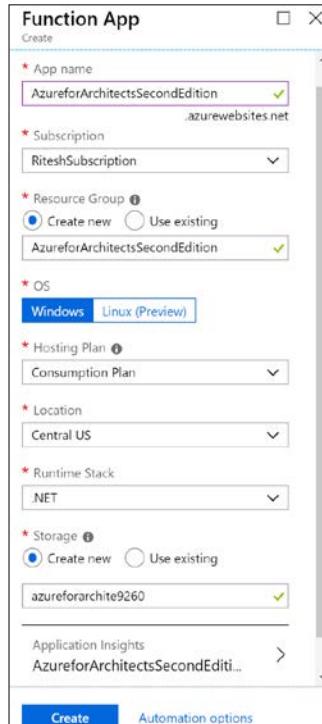
Creating your first Azure Functions

Azure Functions can be created using the Azure portal, PowerShell, Azure CLI, and REST APIs. The steps for creating a function using the ARM template are already detailed at <https://docs.microsoft.com/en-us/azure/azure-functions/functions-infrastructure-as-code>. In this section, Azure Functions will be provisioned using the portal.

Azure Functions are hosted within the Azure App Service. Users create a new function app, which in turn creates an App Service plan and App Service. The App Service plan is configured based on the following:

1. After you have configured the App Service plan, fill in the details as follows:
 - **Name:** The name of the app service. The name should be unique within `.azurewebsites.net`.
 - **Location:** The location for hosting the Azure Functions App Service.
 - **Hosting plan:** This is also known as the pricing plan. Here, two options, as discussed previously, are available – the consumption plan and the App Service plan.
 - **Resource group name:** The name of the resource group containing both the App Service plan and App Service.
 - **Storage account:** Azure Functions need an Azure Storage account to store their internal data and logs.
 - **Enable application insights:** Enable this to capture telemetry information from Azure Functions.

Once the details are filled, click on **Create**:



Creating an Azure Functions App will lead you to the following dashboard after provisioning:

The screenshot shows the Azure Functions App Overview page for 'AzureforArchitectsSecondEdition'. The left sidebar lists 'Function Apps' and 'AzureforArchitectsSecondEdition'. The main area has tabs for 'Overview' (selected) and 'Platform features'. Under 'Overview', it shows the status as 'Running', subscription 'RiteshSubscription', resource group 'AzureforArchitectsSecondEdition', URL 'https://azureforarchitectssecondedition.azurewebsites.net', and location 'Central US'. It also shows 'Subscription ID' and 'App Service plan / pricing tier'. Below this is a section titled 'Configured features' with icons for 'Function app settings', 'Application settings', and 'Application Insights'. A message says 'You have created a function app!' and 'Now it is time to add your code...'. There are 'Stop', 'Swap', 'Restart', 'Get publish profile', 'Reset publish profile', 'Download app content', and 'Delete' buttons at the top.

2. Clicking on the + button next to **Functions** will show a wizard for creating a new function. This wizard shows the tools that can be used to create Azure Functions. Select **In-Portal** as an option and click on the **Continue** button to navigate to the next screen, which displays multiple types of template, including **WebHook + API**, **Timer** and **More Templates....** Select **WebHook + API** and click on the **Create** button. This will create the function with scaffolding code and structure for getting started. This scaffolding code is also generated for default bindings and triggers:

The screenshot shows the code editor for 'run.csx'. The code is as follows:

```
run.csx Save Run </> Get function URL

1 #r "Newtonsoft.Json"
2
3 using System.Net;
4 using Microsoft.AspNetCore.Mvc;
5 using Microsoft.Extensions.Primitives;
6 using Newtonsoft.Json;
7
8 public static async Task<IActionResult> Run(HttpContext req, ILogger log)
9 {
10     log.LogInformation("C# HTTP trigger function processed a request.");
11
12     string name = req.Query["name"];
13
14     string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
15     dynamic data = JsonConvert.DeserializeObject(requestBody);
16     name = name ?? data?.name;
17
18     return name != null
19         ? (ActionResult)new OkObjectResult($"Hello, {name}")
20         : new BadRequestObjectResult("Please pass a name on the query string or in the request body");
21 }
22
```

3. Creating this function provides a complete function-authoring integrated environment, along with some code. This code gets the raw content from the incoming `req` parameter, which is filled up by the Azure runtime with incoming data (query string, form values, and so on). There could be multiple types of data within this incoming parameter, and, in this function, a single value is extracted out of it. The value is converted from JSON to .NET object and based on whether the name parameter is present or absent, and the appropriate response is returned back to user.
4. This function can be invoked using an HTTP request from the browser. The URL for this function is available from the environment and is composed of the function app name, along with the function name. The format is `https://<<function app name>>.azurewebsites.net/api/<<function name>>`. In this case, the URL will be `https://azureforarchitects.azurewebsites.net/api/FirstFunction`.
5. To send parameters to this function, additional query string parameters can be appended at the end of the URL. For example, to send name parameters to this function, the `https://azureforarchitects.azurewebsites.net/api/FirstFunction?name=ritesh` URL can be used. The output of the function is shown in the following screenshot:



6. For HTTP-based functions, the Azure Functions already provide triggers and binding within the `function.json` file, as shown here. This file is used for defining all function-level triggers and bindings, and there is one associated with every function:

```

function.json Save Run </> Get function URL
1 {
2   "bindings": [
3     {
4       "name": "req",
5       "type": "httpTrigger",
6       "direction": "in",
7       "authLevel": "anonymous"
8     },
9     {
10      "name": "res",
11      "type": "http",
12      "direction": "out"
13    }
14  ],
15  "disabled": false
16 }

```

The HTTP template creates a trigger for all incoming requests. The trigger invokes the Azure Functions and passes in the entire incoming data and payload as a parameter named as `req`. This parameter is available within the Azure Functions. The response from the function is a binding that takes output from the `res` variable from the Azure Functions and sends it back to the HTTP channel as a response.

Creating an event-driven function

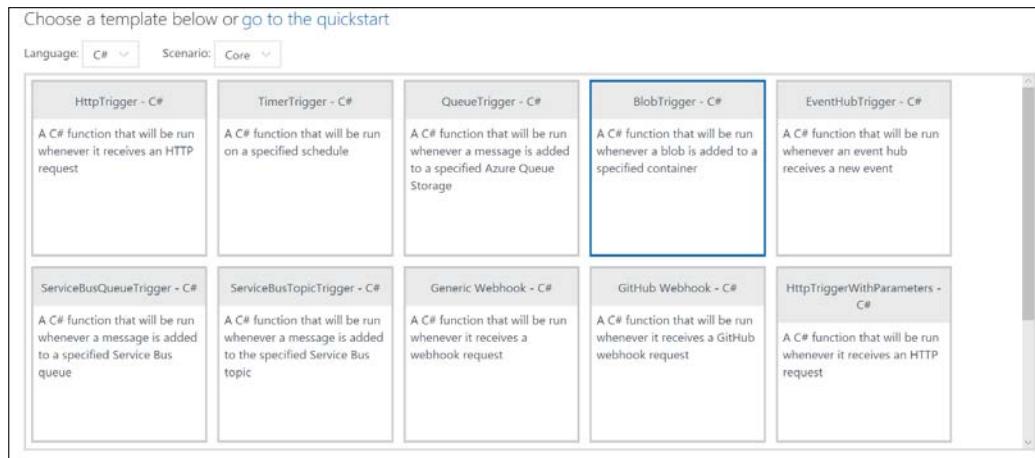
In this example, an Azure Functions will be authored and connected to the Azure Storage account. The Storage account has a container for holding all blob files. The name of the Storage account is `incomingfiles` and the container is `orders`, as shown in the following screenshot:

The screenshot shows the Azure Blob service interface. At the top, it displays the storage account name "incomingfiles". Below this, there is a "Container" button with a plus sign and a "Refresh" button with a circular arrow icon. The main area contains the following details for the storage account:

- Storage account: `incomingfiles`
- Status: Primary: Available
- Location: West Central US
- Subscription (change): `RiteshSubscription`
- Subscription ID: `9755ffce-e94b-4332-9be8-1ade15e78909`

Below these details is a search bar with the placeholder text "Search containers by prefix". A table follows, with the first column labeled "NAME" and the second column showing the value "orders".

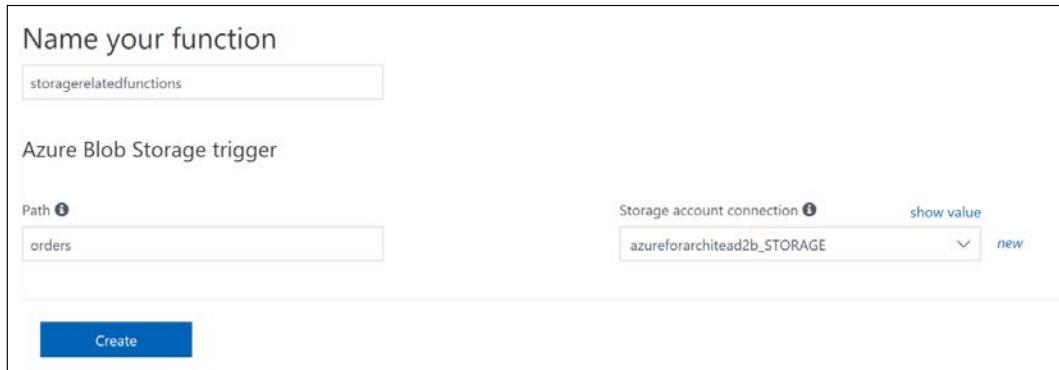
Create a new Azure Function from the Azure portal.



Right now, this Azure Function does not have connectivity to the Storage account. Azure Functions need connection information for the Storage account, and that is available from the **Access keys** tab in the Storage account. The same information can be obtained using the Azure Functions editor environment. In fact, that environment allows for the creation of a new Storage account from the same editor environment.

This can be added using the new button beside the **Storage account connection** input type. It allows for the selection of an existing Storage account or the creation of a new Storage account. Since I already have a couple of Storage accounts, I am reusing them. You should create a separate Azure Storage account. Selecting a Storage account will update the settings in the **appsettings** section with the connection string added to it.

Ensure that a container already exists within the blob service of the target Azure Storage account. The path input refers to the path to the container. In this case, the **orders** container already exists within the Storage account. The **Create** button shown here will provision the new function monitoring the storage account container:



The code for the Azure Functions is as follows:

```
public static void Run(Stream myBlob, TraceWriter log)
{
    log.Info($"C# Blob trigger function Processed blob\n \n Size
{myBlob.Length} Bytes");
}
```

The bindings are shown here:

```
{
  "bindings": [
    {
      "name": "myBlob",
      "type": "blobTrigger",
      "direction": "in",
      "path": "orders",
      "connection": "azureforarchitead2b_STORAGE"
    }
  ],
  "disabled": false
}
```

Now, uploading any blob file to the orders container should trigger the function:

The screenshot shows the Azure Functions developer portal interface. At the top, there's a code editor window titled "run.csx" with a "Save" button and a "Run" button. The code in the editor is:1 public static void Run(Stream myBlob, TraceWriter log)
2 {
3 log.Info(\$"C# Blob trigger function Processed blob\n \n Size: {myBlob.Length}");
4 }
5In the bottom half of the interface, there's a "Logs" section. It displays the following log entries:2017-09-20T10:24:12.504 Function started (Id=ac68a8f8-712f-4df8-975e-238b0ecf5b05)
2017-09-20T10:24:12.536 C# Blob trigger function Processed blob
Size: 2480471 Bytes
2017-09-20T10:24:12.536 Function completed (Success, Id=ac68a8f8-712f-4df8-975e-238b0ecf5b05, DurThe log entry "C# Blob trigger function Processed blob" is highlighted with a yellow background.

Function proxies

Azure Functions proxies are one of the latest additions to Azure Functions. After starting to use Azure Functions, there will be a time when there might be lots of function implementation and it will be difficult to integrate the functions together in a workflow. Instead of letting clients weave these functions together, Azure proxies can be used. Proxies help by providing clients with a single function URL and then invoking multiple Azure Functions behind the scenes to complete workflows.

It is important to understand that proxies are applicable in those cases where functions accept requests on demand, instead of being driven by events. These internal functions connected to proxies can be within a single function app or on multiple separate apps. Proxies get requests from clients, convert, override, and augment the payload, and send them to backend internal functions. Once they get a response from these functions, they can again convert, override, and augment the response and send it back to the client.



More information about Azure Functions proxies can be found at <https://docs.microsoft.com/en-us/azure/azure-functions/functions-proxies>.



Understanding workflows

A workflow is a series of steps or activities that are executed either in parallel or in sequence, or in a combination of the two. Since activities can be executed in parallel, they can perform jobs across multiple services at the same time without being blocked.

The following are the features of workflows:

- **Ability to recover from failure:** A workflow can be hydrated, meaning that its state can be saved at well-defined points within the workflow. If the workflow fails, it starts again from the last saved state rather than from beginning. This feature is also known as checkpoints.
- **Long-running:** Workflows are generally long-running in nature. They can run from minutes to hours or days. They again save the state when they are waiting for an external action to complete and can start again from the last saved state once the external activity is complete.
- **Execute steps in parallel:** We've already discussed this point, but this is a major benefit of a workflow compared to sequence programming.
- **Maintaining state:** Workflows are typically stateful in nature. They maintain the state so that in the event of failure or the restart of a workflow, the workflow does not start from beginning, but can continue from these checkpoints.

Azure Functions need to have finished executing within five minutes. This means they are short-lived, and composing workflows with them can really be difficult. There is the possibility of implementing multiple functions and ensuring they are coordinated to execute one after another. Even this endeavor has its limitations.

This is more of a hack and needs considerable effort, design, and operations. All of the functions will need to have inputs tied to the outputs of their previous function, and each function will still run for few minutes. Another drawback of implementing workflows this way is that it is extremely difficult to understand their connectivity, dependency, and sequence of execution at a glance. In short, Azure Functions are good for implementing a single responsibility that is short-lived. They are not well suited for implementing workflows.

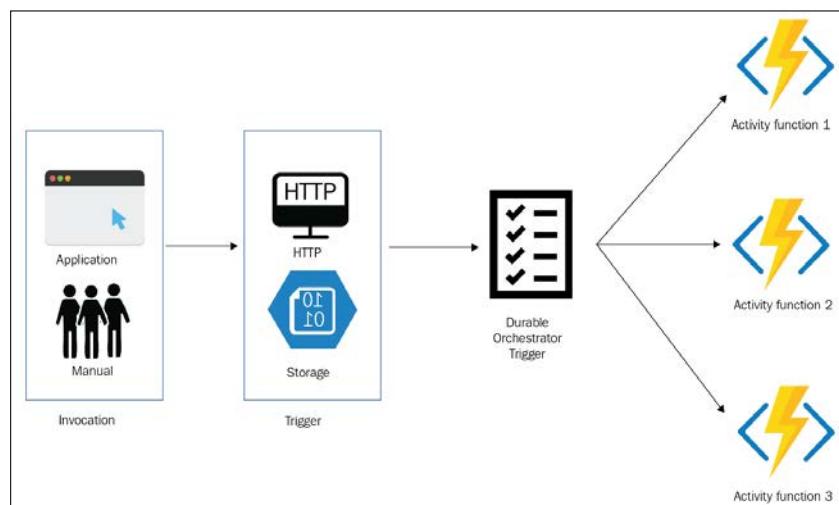
Another important feature of Azure Functions is that they are stateless. This means that the functions are fleeting, and are only available at the time they are being executed. One function instance execution has no relation to the previous or consequent running instance. This means there is no possibility of storing state with Azure Functions.

Azure Functions can run on any server behind the scenes, and that server might not be the same as in subsequent executions. This means that functions cannot store their state in the server they are executed on.

The only way to save state in Azure Functions is to store state externally in data stores such as Cosmos DB, SQL databases, or Azure Storage. But, this design will have significant performance issues, since every time the function executes, it would need to connect to a data store, retrieve, and eventually write to it. Now that we have looked into understanding workflows, we will proceed with Durable Functions.

Durable Functions

Durable Functions are one of the latest additions to Azure Functions and they really fill a gap that existed for a while:



Azure Durable Functions can be invoked by any trigger provided by Azure Functions. These triggers include HTTP, blob storage, table storage, Service Bus queues, and more. They can be triggered manually by someone with access to them, or by an application. The preceding diagram shows a couple of triggers as an example. These are also known as starter Durable Functions. The starter Durable Functions invokes the **durable orchestrator trigger**, which contains the main logic for orchestration, and orchestrates the invocation of activity functions. These activity functions can be called with or without a retry mechanism. Durable Functions can help to solve many challenges and provide features to write functions that can do the following:

- Execute long-running functions
- Maintain state
- Execute child functions in parallel or sequence
- Recover from failure easily
- Orchestrate execution of functions in a workflow

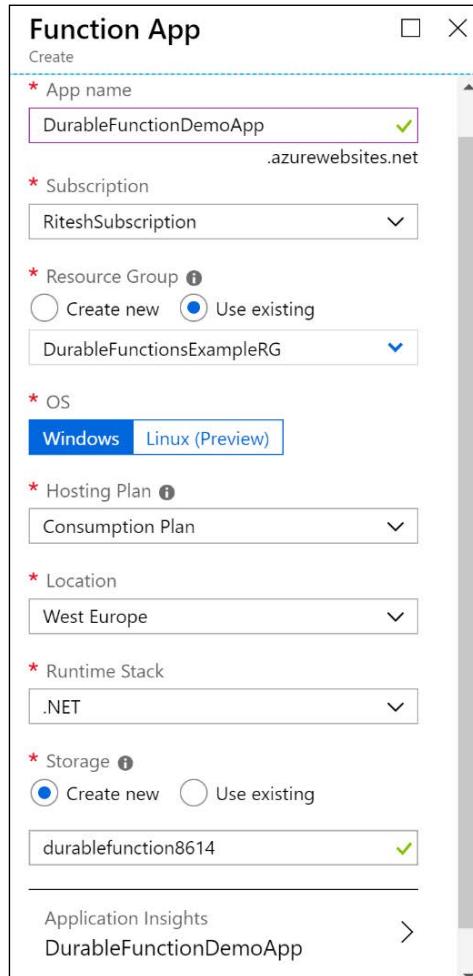
Steps for creating a Durable Functions

Here are the following steps to create a Durable Function:

1. Navigate to the Azure portal and click on **Resource groups** in the left menu.
2. Click on the **+Add** button in the top menu to create a new resource group.
3. Provide the resource group information on the resultant form and click on the **Create** button, as shown here:

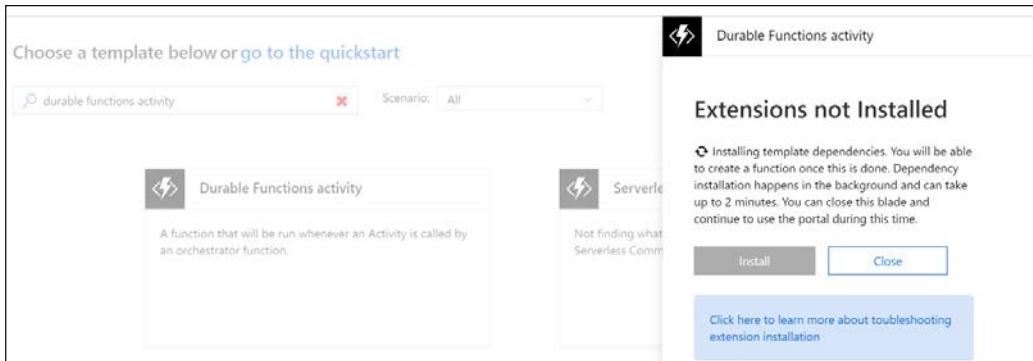
The screenshot shows the 'Create a resource group' dialog. At the top, there are three tabs: Basics (which is selected), Tags, and Review + Create. Below the tabs, a descriptive text explains what a Resource group is: 'A container that holds related resources for an Azure solution. The resource group can include all the resources for the solution, or only those resources that you want to manage as a group. You decide how you want to allocate resources to resource groups based on what makes the most sense for your organization.' A 'Learn more' link is provided. The 'PROJECT DETAILS' section contains fields for 'Subscription' (set to 'RiteshSubscription') and 'Resource group' (set to 'DurableFunctionsExampleRG'). The 'RESOURCE DETAILS' section contains a field for 'Region' (set to 'West Europe').

4. Navigate to the newly created resource group and add a new function app by clicking on the **+Add** button in the top menu and search for **function app** in the resultant search box.
5. Select **Function App** and click on the **Create** button. Fill the resultant function app form and click on the **Create** button, as shown here:

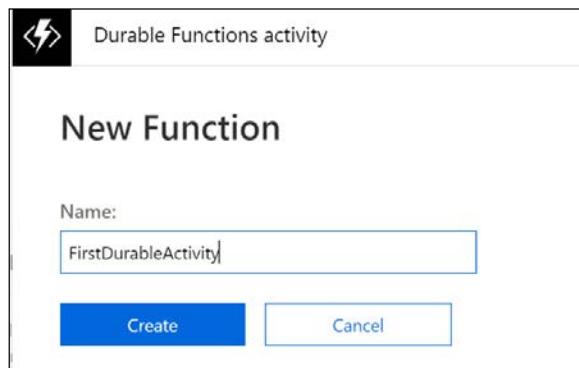


6. Once the function app is created, click on the **Functions +** button in the left menu to create a new function. Select **In-Portal | Continue | More templates... | Finish and view templates.**

7. In the resultant search box, search for Durable Functions activity and select the **Durable Functions activity** template. If the Microsoft.Azure.WebJobs.Extensions.DurableTask extension is not already installed within the function app, it will ask you to install it.
8. Click on the **Install** button to install the extension:



9. Durable Functions activities are functions that are invoked by the main orchestrator function. There is generally one main orchestrator function and multiple Durable Functions activities. Once the extension is installed, provide a name for the function and write code that does something useful, such as sending an email or an SMS, connecting to external systems and executing logic, or executing services using their endpoints, such as Cognitive Services. There should be one Durable Functions activity for each distinct job within the workflow:



10. The generated code should be modified to reflect the code, as shown in the following screenshot:

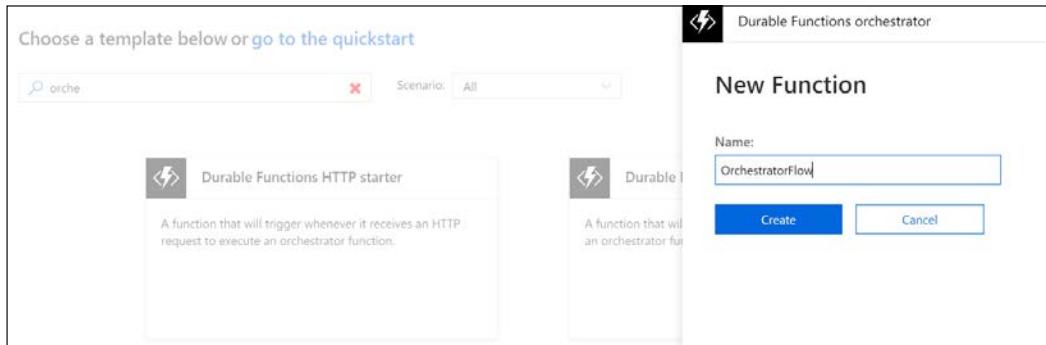
```
#r "Microsoft.Azure.WebJobs.Extensions.DurableTask"

public static async Task< string> Run(string name)
{
    await Task.Delay(10000);
    return $"Hello {name}!";
}
```

The only change is the addition of a single line of code that makes the function wait for 10 seconds.

Similarly create another activity function with same code within it and name it differently: for example, `SecondDurableActivity`.

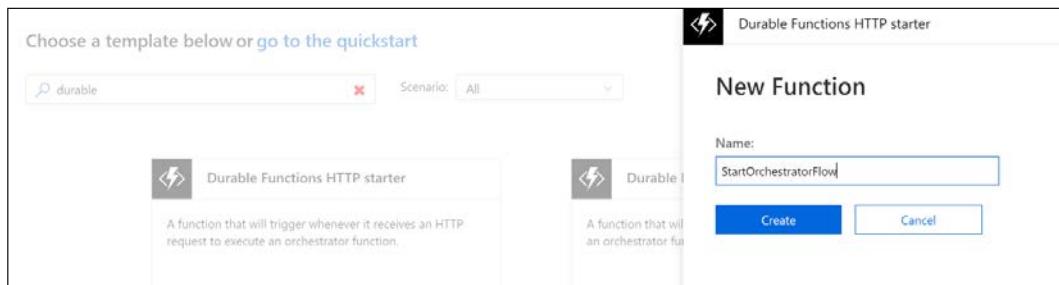
Create a new orchestrator function called `OrchestratorFlow`, as shown in the following screenshot:



Change the generated code to the code shown here:

```
#r "Microsoft.Azure.WebJobs.Extensions.DurableTask"
public static async Task<List<string>> Run(DurableOrchestrationContext context)
{
    var outputs = new List<string>();
    // Replace "Hello" with the name of your Durable Activity Function.
    outputs.Add(await context.CallActivityAsync<string>("FirstDurableActivity", "Tokyo"));
    outputs.Add(await context.CallActivityAsync<string>("SecondDurableActivity", "Seattle"));
    // returns ["Hello Tokyo!", "Hello Seattle!", "Hello London!"]
    return outputs;
}
```

Next, create an orchestrator trigger function. There is a HTTP starter Durable Function provided out-of-the-box, however, it is possible to create a start a Durable Function that is based on external triggers, such as a file getting added to Azure blob storage container. Here, we are creating a simple function that simply starts the orchestrator workflow by calling the `OrchestratorFlow` orchestrator Durable Function. We can start the workflow by invoking it using a browser, or tools such as Postman:



The code for this is listed here:

```
#r "Microsoft.Azure.WebJobs.Extensions.DurableTask"
#r "Newtonsoft.Json"
using System.Net;
public static async Task<HttpResponseMessage> Run(
    HttpRequestMessage req,
    DurableOrchestrationClient starter,
    ILogger log)
{
    // Function input comes from the request content.
    dynamic eventData = await req.Content.ReadAsAsync<object>();
    // Pass the function name as part of the route
    string instanceId = await starter.StartNewAsync("OrchestratorFlow", eventData);
    log.LogInformation($"Started orchestration with ID = '{instanceId}'.");
    return starter.CreateCheckStatusResponse(req, instanceId);
}
```

The `function.json` file containing the declaration of triggers and bindings is modified and the resultant code is shown here:

```
{
  "bindings": [
    {
      "authLevel": "function",
```

```

    "name": "req",
    "type": "httpTrigger",
    "direction": "in",
    "route": "orchestrators",
    "methods": [
        "post",
        "get"
    ],
},
{
    "name": "$return",
    "direction": "out"
},
{
    "name": "starter",
    "type": "orchestrationClient",
    "direction": "in"
},
],
"disabled": false
}

```

Click on the **Get Function URL** link from the UI and copy the resultant URL.

We are going to invoke this URL using a tool known as **Postman** (this can be downloaded from <https://www.getpostman.com/>). This activity is shown in the following screenshot:

KEY	VALUE	DESCRIPTION
code	q0IW1/KoWhx27ykldB7BA2cxF7EcPzOFkhCx1XElOWb9KKCqcH0Lvw==	
key	Value	Description

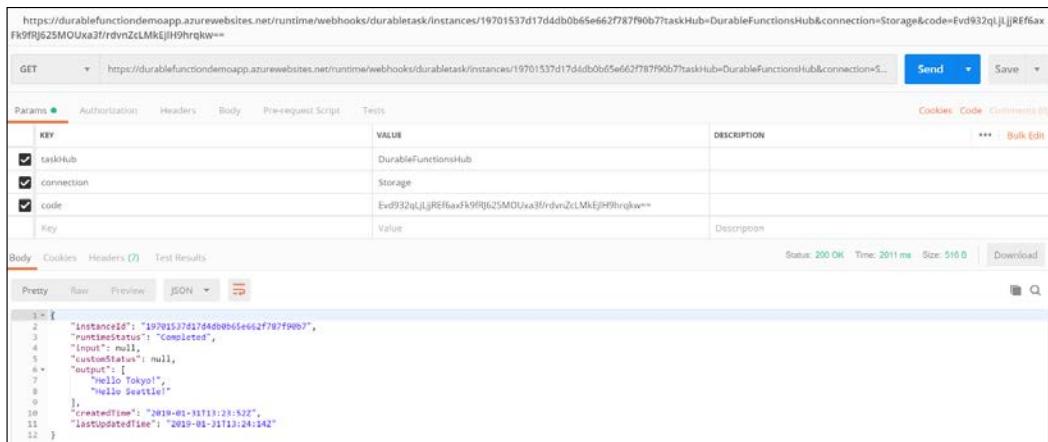
```

[{"id": "19701537d17d4d0b65e62f787f9007", "statusQueryGetUrl": "https://durablefunctiondemoapp.azurewebsites.net/runtime/webhooks/durabletask/instances/19701537d17d4d0b65e62f787f9007?taskHub=durablefunctionsHub&connection=Storage&code=fv932qJlJjR#Eaxfk9f832900xa3f/rdv2zLhMfJ1Hhrqkwa", "sendEventUrl": "https://durablefunctiondemoapp.azurewebsites.net/runtime/webhooks/durabletask/instances/19701537d17d4d0b65e62f787f9007/raiseEvent/{eventName}?taskHub=durablefunctionsHub&connection=Storage&code=fv932qJlJjR#Eaxfk9f832900xa3f/rdv2zLhMfJ1Hhrqkwa", "terminateOrchestratorUrl": "https://durablefunctiondemoapp.azurewebsites.net/runtime/webhooks/durabletask/instances/19701537d17d4d0b65e62f787f9007/terminate?reason={text}&taskHub=durablefunctionsHub&connection=Storage&code=fv932qJlJjR#Eaxfk9f832900xa3f/rdv2zLhMfJ1Hhrqkwa", "rewindOrchestratorUrl": "https://durablefunctiondemoapp.azurewebsites.net/runtime/webhooks/durabletask/instances/19701537d17d4d0b65e62f787f9007/rewind?reason={text}&taskHub=durablefunctionsHub&connection=Storage&code=fv932qJlJjR#Eaxfk9f832900xa3f/rdv2zLhMfJ1Hhrqkwa"}]

```

Notice that four URLs are generated when you start an orchestrator:

- The `statusQueryGetUri` URL is used to find the current status of the orchestrator. Clicking this URL on Postman opens a new tab and shows the status of the workflow:



The screenshot shows a Postman request to the URL `https://durablefunctiondemoapp.azurewebsites.net/runtime/webhooks/durabletask/instances/19701537d17d4db0b65e662f787f90b7?taskHub=DurableFunctionsHub&connection=Storage&code=Evd932qlJLjjREf6axFk9fRj62SMOUxa3frdvnZcLMkEjH9hrqkw==`. The request method is GET. The Params tab shows three checked parameters: `taskHub` (value: `DurableFunctionsHub`), `connection` (value: `Storage`), and `code` (value: `Evd932qlJLjjREf6axFk9fRj62SMOUxa3frdvnZcLMkEjH9hrqkw==`). The Body tab shows a JSON response with the following content:

```
1 × [
  {
    "instanceId": "19701537d17d4db0b65e662f787f90b7",
    "runStatus": "Completed",
    "input": null,
    "customStatus": null,
    "outputs": [
      {
        "name": "Hello Tokyo",
        "value": "Hello Seattle"
      }
    ],
    "createdTime": "2020-01-11T13:23:52Z",
    "lastUpdatedTime": "2020-01-11T13:24:19Z"
  }
]
```

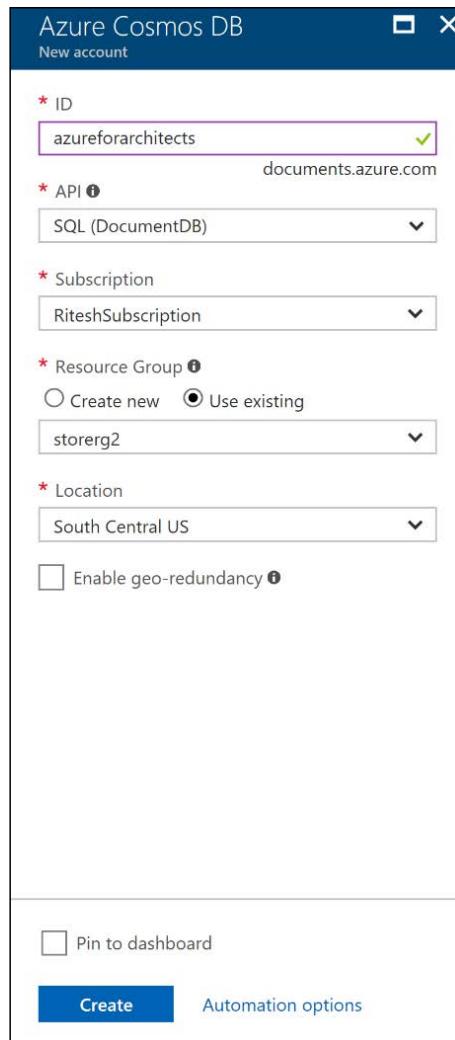
- The `terminatePostUri` URL is used for stopping an already running orchestrator function.
- The `sendEventPostUri` URL is used to post an event to a suspended durable function. Durable Functions can be suspended if they are waiting for an external event. This URL is used in those cases.
- The `rewindPostUri` URL is used to post a message to rewind an orchestrator function.

Creating a connected architecture with functions

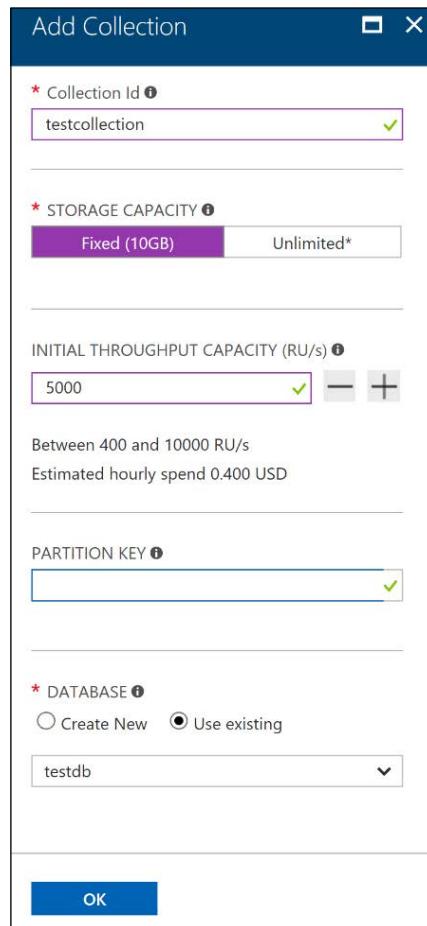
A connected architecture with functions refers to creating multiple functions, whereby the output of one function triggers another function and provides data for the next function to execute its logic. In this section, we will continue with the previous scenario of the Storage account. In this case, the output of the function being triggered using Azure Storage blob files will write the size of the file to Azure Cosmos DB.

The configuration of Cosmos DB is shown next. By default, there are no collections created in Cosmos DB.

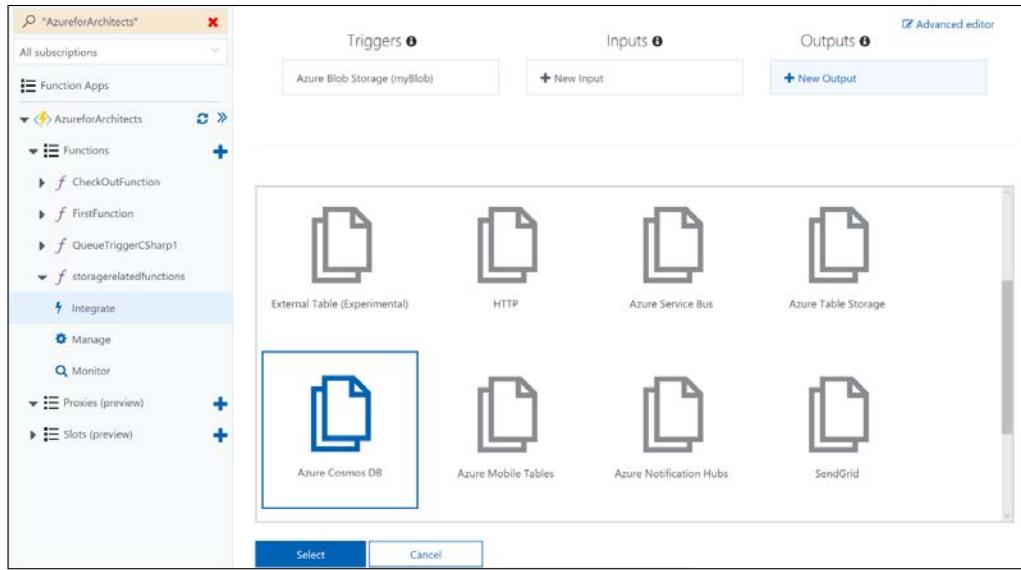
A collection will automatically be created when creating a function that will be triggered when Cosmos DB gets any data:



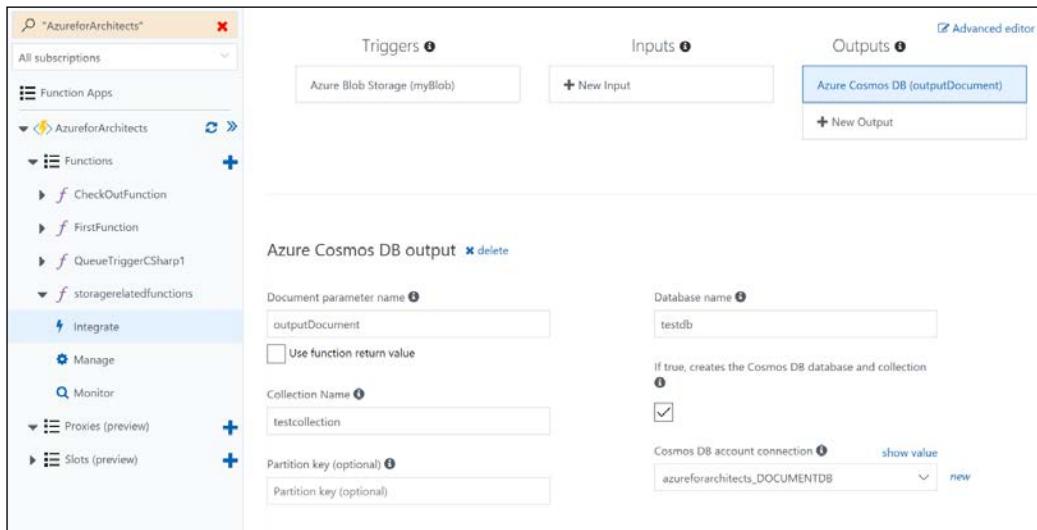
Create a new database, `testdb`, within Cosmos DB, and create a new collection named `testcollection` within it. You need both the database and collection name when configuring Azure Functions:



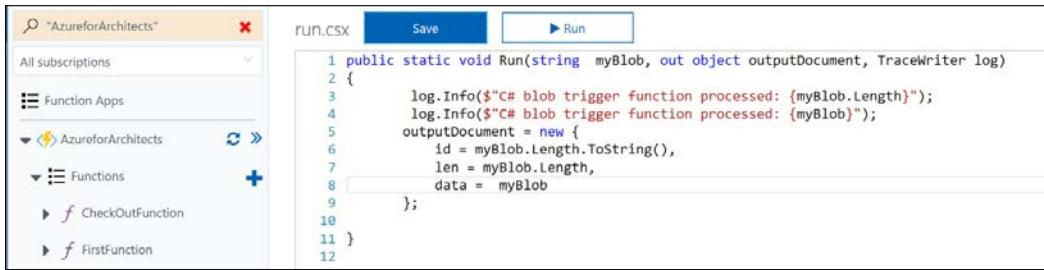
It's time to revisit the `storagerelatedfunctions` function and change its binding to return the size of the data for the uploaded file. This returned value will be written to Cosmos DB. This will require a change to the bindings as well, with an additional one responsible for capturing output values. This binding will eventually write to the Cosmos DB collection. Navigate to the **Integrate** tab and click on the **New Output** button below the **Outputs** label and select **Azure Cosmos DB**:



Provide the appropriate names for the database and collection (check the checkbox to create the collection if it does not exist), click on the **New** button to select our newly created Azure Cosmos DB, and leave the parameter name as `outputDocument`:



Modify the function as shown in the following screenshot:



The screenshot shows the Azure Functions developer portal. On the left, there's a sidebar with a search bar for "AzureforArchitects", a dropdown for "All subscriptions", and a tree view showing "Function Apps" under "AzureforArchitects", which has "Functions" expanded, listing "CheckOutFunction" and "FirstFunction". The main area is titled "RUN.CS" and contains the following C# code:

```
1 public static void Run(string myBlob, out object outputDocument, TraceWriter log)
2 {
3     log.Info($"C# blob trigger function processed: {myBlob.Length}");
4     log.Info($"C# blob trigger function processed: {myBlob}");
5     outputDocument = new {
6         id = myBlob.Length.ToString(),
7         len = myBlob.Length,
8         data = myBlob
9     };
10 }
11 }
```

Now, uploading a new file to the orders collection in the Azure Storage account will execute a function that will write to the Azure Cosmos DB collection. Another function can be written with the newly created Azure Cosmos DB account as a trigger binding. It will provide the size of files and the function can act on it. This is shown here:



The screenshot shows the Azure Cosmos DB portal. On the left, there's a sidebar with "Create", "Upload", and "More" buttons, and a dropdown set to "testcollection". Below that is a "Documents" section with a search bar. On the right, there's a "Save", "Discard", "Delete", "Refresh", and "Properties" toolbar. The main area shows a list of documents with IDs 137, 1580, and 603. The document with ID 137 is selected and its content is displayed in a JSON-like format:

```
1 {
2     "id": "137",
3     "len": 137,
4     "data": "<?xml version='1.0\' encoding='utf-8'?>\r\n<configuration>\r\n    <Ritesh id='1'>aaaa</Ritesh>\r\n    <Ritesh id='2'>bbbb</Ritesh>\r\n</configuration>"
5 }
```

Summary

The evolution of functions from traditional methods has led to the design of the loosely coupled, independently evolving, self-reliant serverless architecture that was only a concept in earlier days. Functions are a unit of deployment and provide an environment that does not need to be managed by the user at all. All they have to care about is the code written for the functionality. Azure provides a mature platform for hosting functions and integrating them seamlessly, based on events or on demand. Nearly every resource in Azure can participate in an architecture composed of Azure Functions. The future is functions, as more and more organizations want to stay away from managing infrastructures and platforms. They want to offload this to cloud providers. Azure Functions is an essential feature to master for every architect dealing with Azure.

The next chapter will build on what we learned in this chapter, as it will explore other serverless technologies such as Azure Logic Apps and Event Grids. It will also show a complete end-to-end solution using all of these technologies.

7

Azure Integration Solutions

This chapter is a continuation of the previous chapter. In Chapter 6, *Designing and Implementing Serverless Solutions*, we discussed serverless compute Azure Functions, and in this chapter, we will continue our discussion of serverless technologies and cover Azure Event Grids as part of serverless events, and also Azure Logic Apps as part of serverless workflows. A complete end-to-end solution will also be created using multiple Azure services, such as Automation, Logic Apps, Functions, Event Grid, SendGrid, Twilio, PowerShell, Azure AD, and Key Vaults.

The following topics will be covered in this chapter:

- Azure Event Grid
- Azure Logic Apps
- Creating an end-to-end solution using serverless technologies

Azure Event Grid

Azure Event Grid is a relatively new service. It has also been referred to as a serverless eventing platform. It helps with the creation of applications based on events (also known as **event-driven design**). It is important to understand what events are and how we dealt with them prior to Event Grid. An event is *something that happened* – that is, an activity that changed the state of a subject. When a subject undergoes a change in its state, it generally raises an event.

Events typically follow the publish/subscribe pattern (also popularly known as the **pub/sub pattern**), in which a subject raises an event due to its state change, and that event can then be subscribed to by multiple interested parties, also known as **subscribers**. The job of the event is to notify the subscribers of such changes and also provide them with data as part of its context. The subscribers can take whatever action they deem necessary, which varies from subscriber to subscriber.

Prior to Event Grid, there was no service that could be described as a real-time event platform. There were separate services, and each provided its own mechanism for handling events.

For example, Log Analytics, also known as **Operations Management Suite (OMS)**, provides an infrastructure for capturing environment logs and telemetry on which alerts can be generated. These alerts can be used to execute a runbook, a webhook, or a function. This is near to real time, but they are not completely real time. Moreover, it was quite cumbersome to trap individual logs and act on them. Similarly, there is Application Insights, which provides similar features to Log Analytics but for applications instead.

There are other logs, such as activity logs and diagnostic logs, but again, they rely on similar principles as other log-related features. Solutions are deployed on multiple resource groups in multiple regions, and events raised from any of these should be available to the resources that are deployed elsewhere.

Event Grid removes all barriers, and as a result, events can be generated by most resources (they are increasingly becoming available), and even custom events can be generated. These events can then be subscribed to by any resource, in any region, and in any resource group within the subscription.

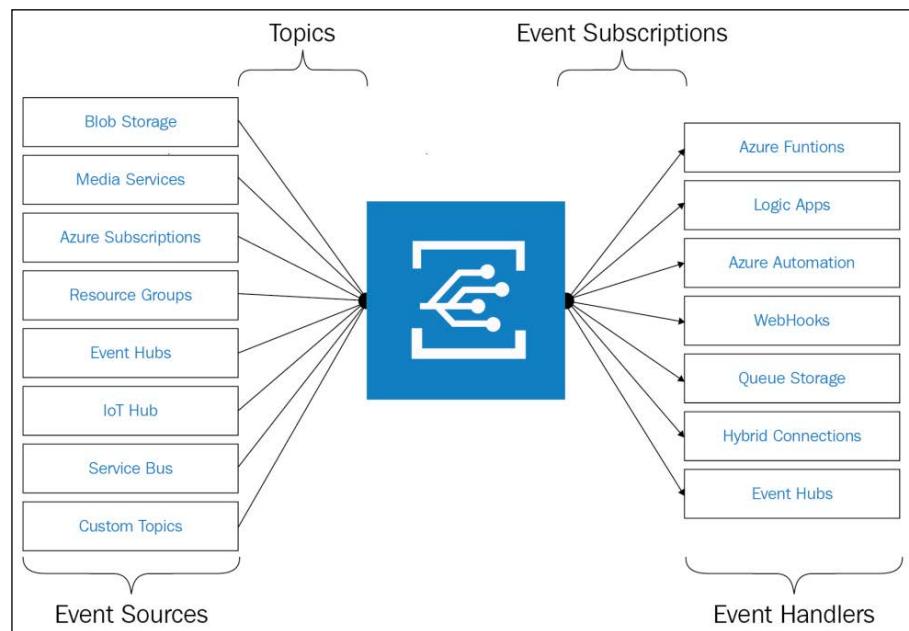
Event Grid is already laid down as part of the Azure infrastructure, along with data centers and networks. Events raised in one region can easily be subscribed to by resources in other regions, and since these networks are connected, it is extremely efficient for the delivery of events to subscribers.

The Event Grid architecture

The Event Grid architecture is based on service bus topics. Service bus topics, as we already know, are based on the publish/subscribe mechanism. There are publishers of events and there are consumers of events; however, there can be multiple subscribers for the same event.

The publisher of an event can be an Azure resource, such as Blob storage, **Internet of Things (IoT)** hubs, and many others. These publishers are also known as event sources. These publishers use out-of-the-box Azure topics to send their events to Event Grid. There is no need to configure either the resource or the topic. The events raised by Azure resources are already internally using topics to send their events to Event Grid. Once the event reaches the grid, it can be consumed by the subscribers.

The subscribers, or consumers, are resources who are interested in events and want to execute an action based on these events. These subscribers provide an event handler when they subscribe to the topic. The event handlers can be Azure functions, custom web hooks, logic apps, or other resources. Both the event sources and subscribers that execute event handlers are shown in the following diagram:



When an event reaches a topic, multiple event handlers can be executed simultaneously, each taking its own action.

It is also possible to raise a custom event and send a custom topic to Event Grid. Event Grid provides features for creating custom topics, and these topics are automatically attached to Event Grid. These topics know the storage for Event Grid and automatically send their messages to it. Custom topics have two important properties, as follows:

- **An endpoint:** This is the endpoint of the topic. Publishers and event sources use this endpoint to send and publish their events to Event Grid. In other words, topics are recognized using their endpoints.
- **Keys:** Custom topics provide a couple of keys. These keys enable security for the consumption of the endpoint. Only publishers with these keys can send and publish their messages to Event Grid.

Each event has an event type and it is recognized by it. For example, blob storage provides event types, such as `blobAdded` and `blobDeleted`. Custom topics can be used to send a custom-defined event, such as a custom event of the `KeyVaultSecretExpired` type.

On the other hand, subscribers have the ability to accept all messages or only get events based on filters. These filters can be based on the event type or other properties within the event payload.

Each event has at least the following five properties:

- `id`: This is the unique identifier for the event.
- `eventType`: This is the event type.
- `eventTime`: This is the date and time when the event was raised.
- `subject`: This is a short description of the event.
- `data`: This is a dictionary object and contains either resource-specific data or any custom data (for custom topics).

Currently, Event Grid's functionalities are not available with all resources; however, Azure is continually adding more and more resources with Event Grid functionality.

To find out more about the resources that can raise events related to Event Grid and handlers that can handle these events, please go to <https://docs.microsoft.com/en-us/azure/event-grid/overview>.

Resource events

In this section, the following steps are provided to create a solution in which events that are raised by Blob storage are published to Event Grid and ultimately routed to an Azure function:

1. Log in to the Azure portal using the appropriate credentials and create a new storage account in an existing or a new resource group. The storage account should be either StorageV2 or Blob storage. As demonstrated in the following screenshot, Event Grid will not work with StorageV1:

Create storage account

[Basics](#) [Advanced](#) [Tags](#) [Review + create](#)

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

PROJECT DETAILS

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

* Subscription: RiteshSubscription

* Resource group: azureclitest [Create new](#)

INSTANCE DETAILS

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

* Storage account name: storageforeventgrid

* Location: West Europe

Performance: Standard Premium

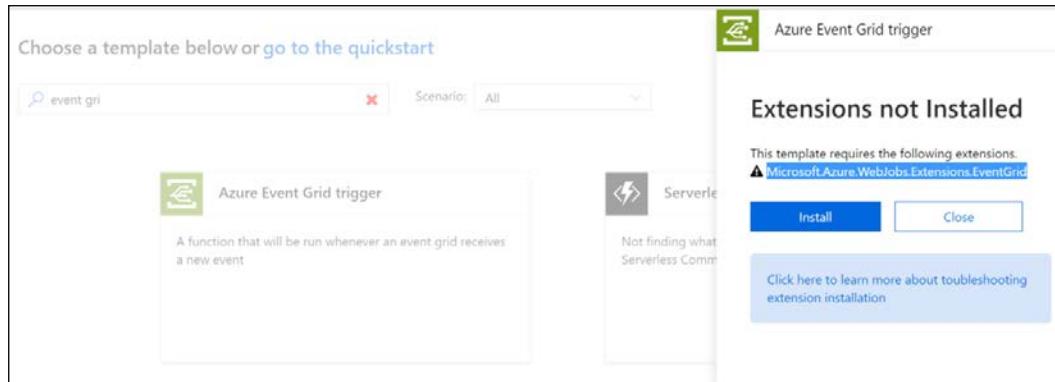
Account kind: StorageV2 (general purpose v2)

Replication: Read-access geo-redundant storage (RA-GRS)

Access tier (default): Cool Hot

[Review + create](#) [Previous](#) [Next : Advanced >](#)

2. Create a new function app or reuse an existing function app to create an Azure function. The Azure function will be hosted within the function app.
3. Create a new function using the Azure Event Grid trigger template. Install the `Microsoft.Azure.WebJobs.Extensions.EventGrid` extension if it's not already installed, as shown in the following screenshot:



4. Name the `StorageEventHandler` function and create it. The following default generated code will be used as the event handler:

The screenshot shows the Azure Functions code editor with a single file named 'run.csx'. The file contains the following C# code:

```
1 #r "Microsoft.Azure.EventGrid"
2 using Microsoft.Azure.EventGrid.Models;
3
4 public static void Run(EventGridEvent eventGridEvent, ILogger log)
5 {
6     log.LogInformation(eventGridEvent.Data.ToString());
7 }
8
```

At the top of the editor, there are three buttons: 'Save', 'Run' (with a play icon), and 'Add Event Grid subscription'. The code editor has a light gray background with syntax highlighting for C# and JSON.

The subscription to storage events can be configured either from the Azure Functions **user interface (UI)** by clicking on Add Event Grid subscription, or from the storage account itself.

5. Click on the **Add Event Grid subscription** link in the Azure Functions UI to add a subscription to the events raised by the storage account created in the previous step. Provide a name for the subscription, and then choose **Event Schema** followed by **Event Grid Schema**. Set **Topic Types** as **Storage Accounts**, set an appropriate **Subscription**, and the resource group containing the storage account:

Home > DurableFunctionDemoApp - StorageEventHandler > Create Event Subscription

Create Event Subscription

Event Grid

Basic **Filters** **Additional Features**

Event Subscriptions listen for events emitted by the topic resource and send them to the endpoint resource. [Learn more](#)

EVENT SUBSCRIPTION DETAILS

Name	StorageEventSubscription
Event Schema	Event Grid Schema

TOPIC DETAILS

Pick a topic resource for which events should be generated and pushed. [Learn more](#)

Topic Types	Storage Accounts
Subscription	RiteshSubscription
Resource Group	azureclitest
Resource	someoddstoreacc1 storageforeventgrid someoddstoreacc

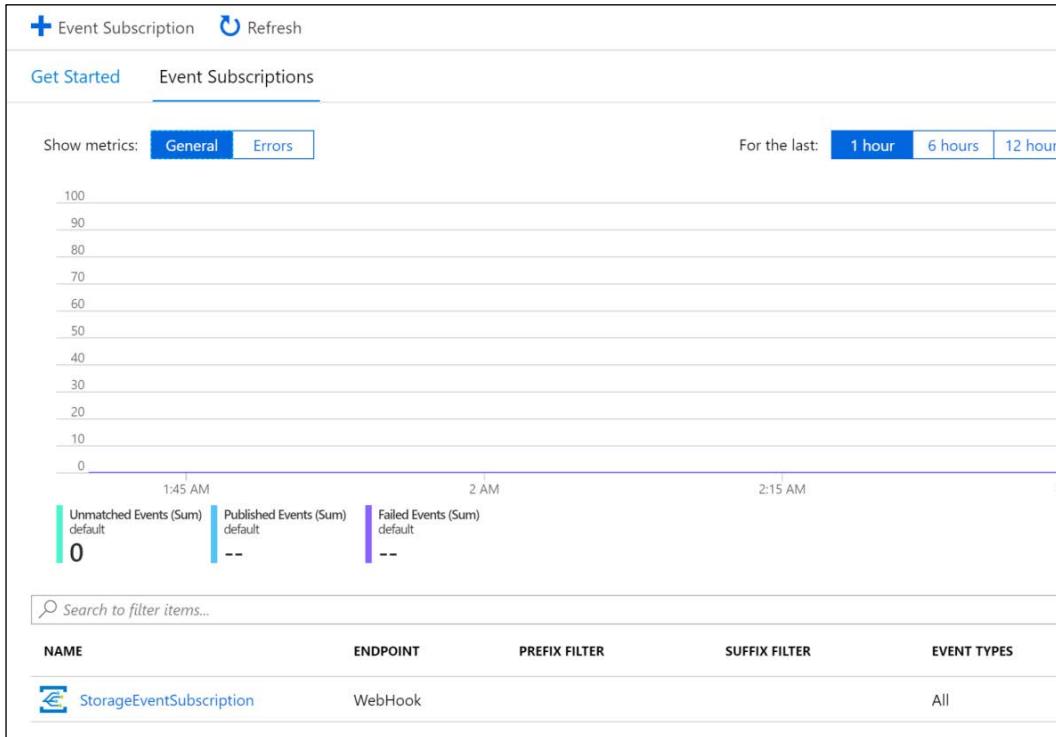
EVENT TYPES

Pick which event types get pushed to your d

Subscribe to all event types

Ensure that the Subscribe to all event types checkbox is checked and click on the Create button (it should be enabled as soon as a storage account is selected).

6. If we now navigate to the storage account in the Azure portal and click on the Events link in the left-hand menu, the subscription for the storage account should be visible:



7. Upload a file to the Blob storage after creating a container, and the Azure function should be executed. The upload action will trigger a new event of the `blobAdded` type and send it to the Event Grid topic for storage accounts. As shown in the following screenshot, the subscription is already set to get all the events from this topic, and the function gets executed as part of the event handler:

```
1 #r "Microsoft.Azure.EventGrid"
2 using Microsoft.Azure.EventGrid.Models;
3
4 public static void Run(EventGridEvent eventGridEvent, ILogger log)
5 {
6     log.LogInformation(eventGridEvent.Data.ToString());
7 }
8
```

Logs Console

Reconnect Copy logs Pause Clear Expand

```
2019-01-10T07:48:40 Welcome, you are now connected to log-streaming service.
2019-01-10T07:48:51.428 [Information] Executing 'Functions.StorageEventHandler' (Reason='EventGrid trigger fired at 2019-01-10T07:48:51.428Z 5000:00:00', Id=5c4e2121-f903-40bb-a45a-e49e674348ec)
2019-01-10T07:48:51.429 [Information] {
  "api": "PutBlockList",
  "clientRequestId": "010e48eb-a88e-4a48-bc1-44da8e8b503c",
  "requestId": "a0217c1f-f0fe-0032-53b8-a83345000000",
  "etag": "0xb0d7600fA07895",
  "contentType": "application/pdf",
  "contentLength": 126822,
  "blobType": "BlockBlob",
  "url": "https://storageforeventgrid.blob.core.windows.net/filecontainer/superdry.pdf",
  "sequencer": "000000000000000000000000000000042d8000000000002d532",
  "correlationId": f
```

Custom events

In this example, instead of using out-of-the-box resources to generate events, custom events will be used. We will use PowerShell to create this solution and reuse the same Azure function that was created in last exercise as the handler:

1. Log in and connect to your Azure subscription of choice using `Login-AzureRMAccount` and `Set-AzureRmContext` cmdlet.
 2. The next step is to create a new Event Grid topic in Azure in a resource group. The `New-AzureRmEventGridTopic` cmdlet is used to create a new topic:

```
New-AzureRmEventGridTopic -ResourceGroupName CustomEventGridDemo  
-Name "KeyVaultAssetsExpiry" -Location "West Europe"
```

3. Once the topic is created, its endpoint URL and key should be retrieved as they are needed to send and publish the event to it. The `Get-AzureRmEventGridTopic` and `Get-AzureRmEventGridTopicKey` cmdlets are used to retrieve these values. Note that `Key1` is retrieved to connect to the endpoint:

```
$topicEndpoint = (Get-AzureRmEventGridTopic -ResourceGroupName containers -Name KeyVaultAssetsExpiry).Endpoint
```

```
$keys = (Get-AzureRmEventGridTopicKey -ResourceGroupName containers -Name KeyVaultAssetsExpiry).Key1
```

4. A new hash table is created with all five important Event Grid event properties. A new guid property is generated for the ID, the subject property is set to `Key vault Asset Expiry`, `eventType` is set to `Certificate Expiry`, `eventTime` is set to current time, and data contains information regarding the certificate:

```
$eventgridDataMessage = @{
    id = [System.guid]::NewGuid()
    subject = "Key Vault Asset Expiry"
    eventType = "Certificate Expiry"
    eventTime = [System.DateTime]::UtcNow
    data = @{
        CertificateThumbprint = "sdfervdserwetsgfhgdf"
        ExpiryDate = "1/1/2019"
        Createdon = "1/1/2018"
    }
}
```

5. Since Event Grid data should be published in the form of a JSON array, the payload is converted in the JSON array. The `"[,]"` square brackets represent a JSON array:

```
$finalBody = "[" + $(ConvertTo-Json $eventgridDataMessage) + "]"
```

6. The event will be published using the HTTP protocol, and the appropriate header information has to be added to the request. The request is sent using the application/JSON content type and the key belonging to the topic is assigned to the `aeg-sas-key` header. It is mandatory to name the header and key set to `aeg-sas-key`:

```
$header = @{
    "contentType" = "application/json"
    "aeg-sas-key" = $keys}
```

7. A new subscription is created to the custom topic with a name, the resource group containing the topic, the topic name, the webhook endpoint, and the actual endpoint that acts as the event handler. The event handler in this case is the Azure function:

```
New-AzureRmEventGridSubscription -TopicName KeyVaultAssetsExpiry
-EventSubscriptionName "customtopicsubscriptionautocar"
-ResourceGroupName CustomEventGridDemo -EndpointType webhook
-Endpoint "https://durablefunctiondemoapp.azurewebsites.net/runtime/webhooks/
EventGrid?functionName=StorageEventHandler&code=0aSw6sxvtFmafXHvt7iOw/
Dsb8o1M9RKKagzVchTUkwe9Elkzl4mCg=="
-Verbose
```

The URL of the Azure function is available from the Integrate tab, as shown in the following screenshot:



8. By now, both the subscriber (event handler) and the publisher have been configured. The next step is to send and publish an event to the custom topic. The event data was already created in the previous step and, by using the Invoke-WebRequest cmdlet, the request is sent to the endpoint along with the body and the header:

```
Invoke-WebRequest -Uri $topicEndpoint -Body $finalBody -Headers
$header -Method Post
```

Azure Logic Apps

Logic Apps is the serverless workflow offering from Azure. It has all the features of serverless technologies, such as consumption-based costing and unlimited scalability. Logic Apps helps us to build a workflow solution with ease using the Azure portal. It provides a drag and drop UI to create and configure the workflow.

Using Logic Apps is the preferred way to integrate services and data, create business projects, and create a complete flow of logic. There are a number of important concepts that should be understood before building a logic app.

Activity

Activity refers to performing a single unit of work. Examples of activities include converting XML to JSON, reading blobs from Azure storage, and writing to a Cosmos DB document collection. Logic Apps is a workflow consisting of multiple co-related activities in a sequence. There are two types of activity in Logic Apps, as follows:

- **Trigger:** Triggers refer to the *initiation of an activity*. All logic apps have a single trigger that forms the first activity. It is the trigger that creates an instance of the logic app and starts the execution. Examples of triggers are the arrival of Event Grid messages, the arrival of an email, an HTTP request, and a schedule.
- **Actions:** Any activity that is not a trigger is a step activity, and they each perform one responsibility. Steps are connected to each other in a workflow.

Connectors

Connectors are Azure resources that help connect a logic app to external services. These services can be in the cloud or on premises. For example, there is a connector for connecting Logic Apps to Event Grid. Similarly, there is another connector to connect to Office 365 Exchange. Almost all types of connectors are available in Logic Apps, and they can be used to connect to services. Connectors contain connection information and also logic to connect to external services using this connection information.



The entire list of connectors is available at <https://docs.microsoft.com/en-us/connectors/>.



Working on a logic app

Let's create a Logic Apps workflow that gets triggered when one of the email accounts receives an email. It replies to the sender with a default email and performs sentiment analysis on the content of the email. For sentiment analysis, the Text Analytics resource from Cognitive Services should be provisioned before creating the logic app:

1. Navigate to the Azure portal, log in, and create a **Text Analytics** resource in a resource group, as shown in the following screenshot:

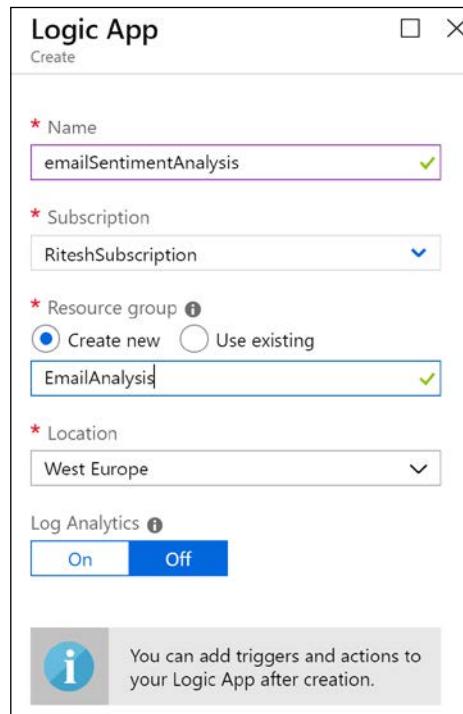
2. Provide a **Name**, **Location**, and **Subscription** name, a **Resource group** name, and **Pricing tier**, as follows:

* Name	emailanalysis
* Subscription	RiteshSubscription
* Location	West Europe
* Pricing tier (View full pricing details)	F0 (5K Transactions per 30 days)
* Resource group	EmailAnalysis
Create new	

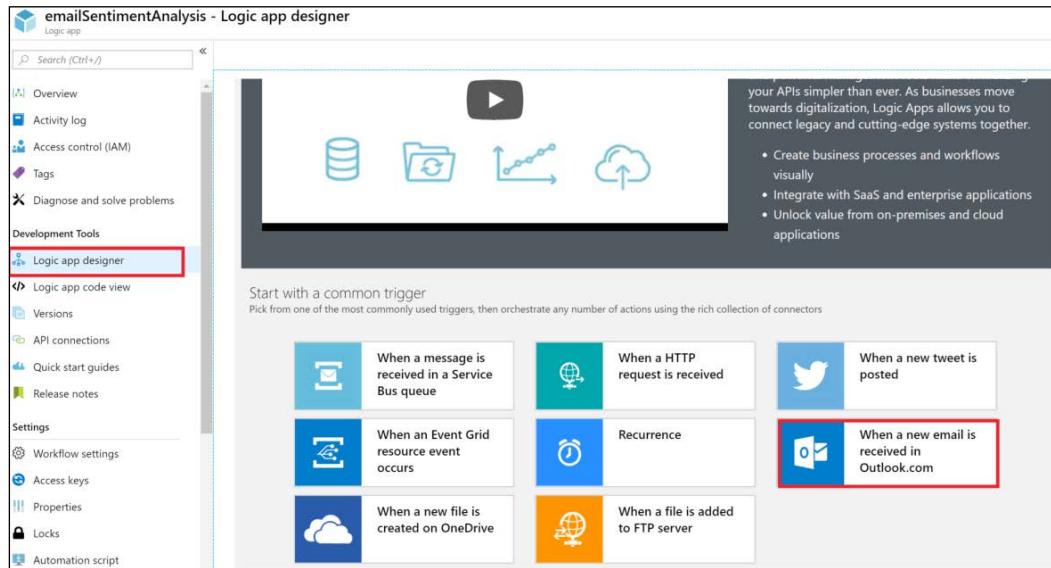
- Once the resource is provisioned, navigate to the **Overview** page and copy the endpoint URL. Store it in a temporary location. This value will be required when configuring the logic app:



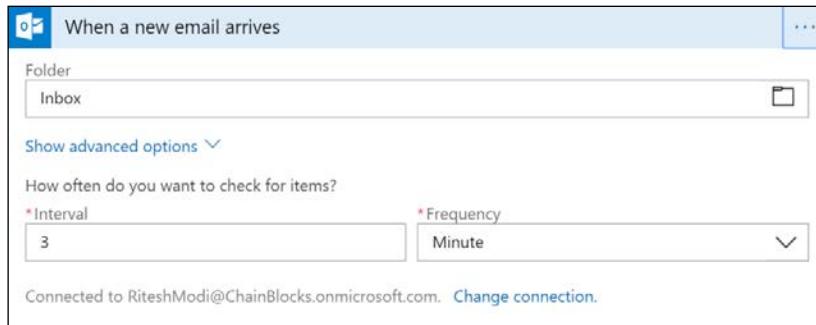
- Navigate to the **Keys** page and copy the value from **Key 1** and store it in a temporary location. This value will be needed when configuring the logic app.
- The next step is to create a logic app; to create a logic app, navigate to the resource group in the Azure portal in which the logic app should be created. Search for **Logic App** and create it by providing a **Name**, **Location**, **Resource group** name, and **Subscription** name:



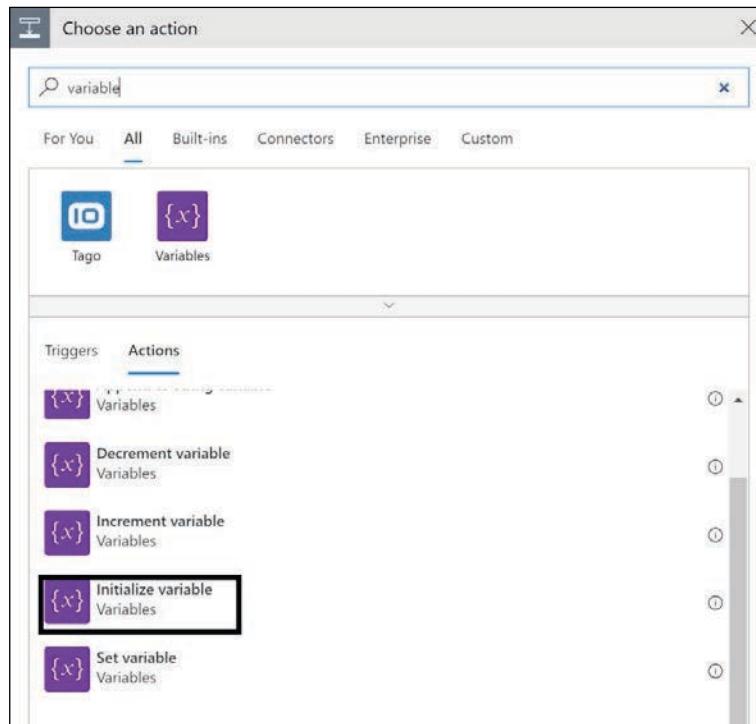
- After the logic app has been created, navigate to the resource, click on Log app designer in the left-hand menu, and then select the **When a new email is received in Outlook.com** template to create a new workflow. The template provides a jump-start by adding boilerplate triggers and activities. This will add an Office 365 Outlook trigger automatically to the workflow, as shown in the following screenshot:



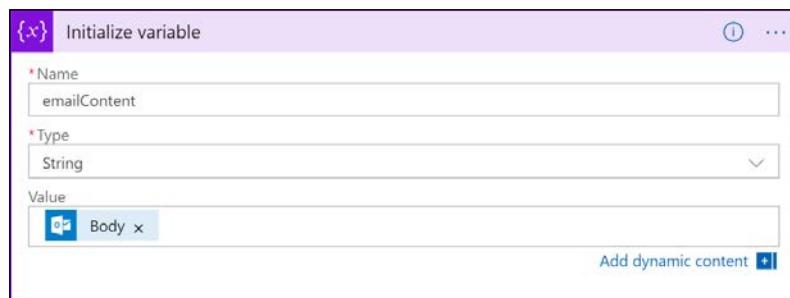
- Click on the Sign in button on the trigger; it will open a new Internet Explorer window. Then, sign in to your account. After successfully connecting, a new Office 365 mail connector will be created, containing the connection information to the account.
- Click on the Continue button and configure the trigger with a 3-minute poll frequency, as shown in the following screenshot:



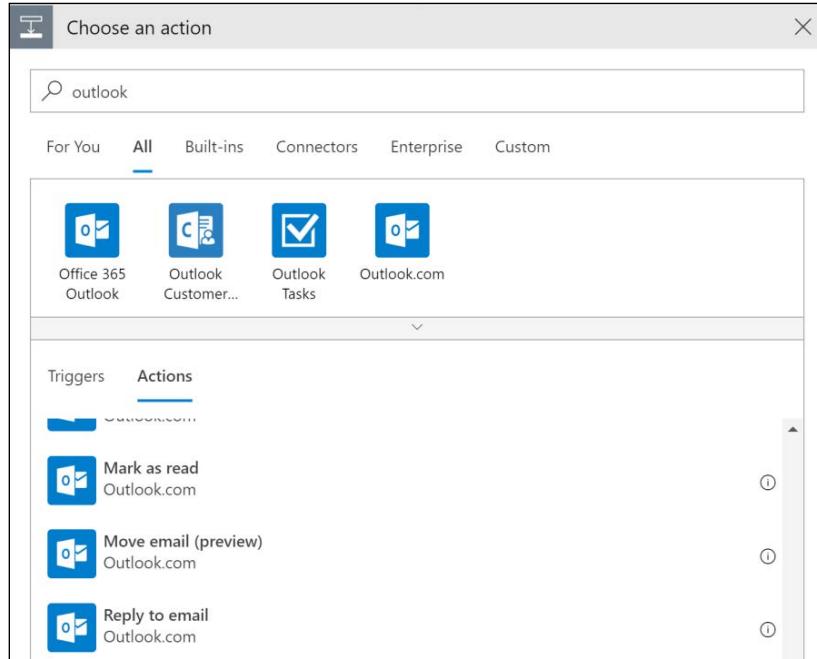
9. Click on the next step to add another action and type variables. Then, select the Initialize variable action, as demonstrated in the following screenshot:



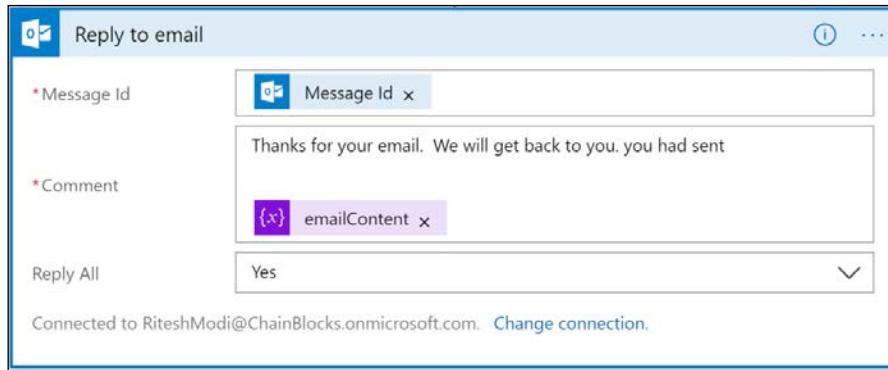
10. Next, configure the variable action. When the **Value** textbox is clicked on, a pop-up window appears that shows Dynamic content and Expression. Dynamic content refers to properties that are available to the current action, which are filled with runtime values from previous actions and triggers. Variables help in keeping the workflows generic. From this window, select **Body** from Dynamic content:



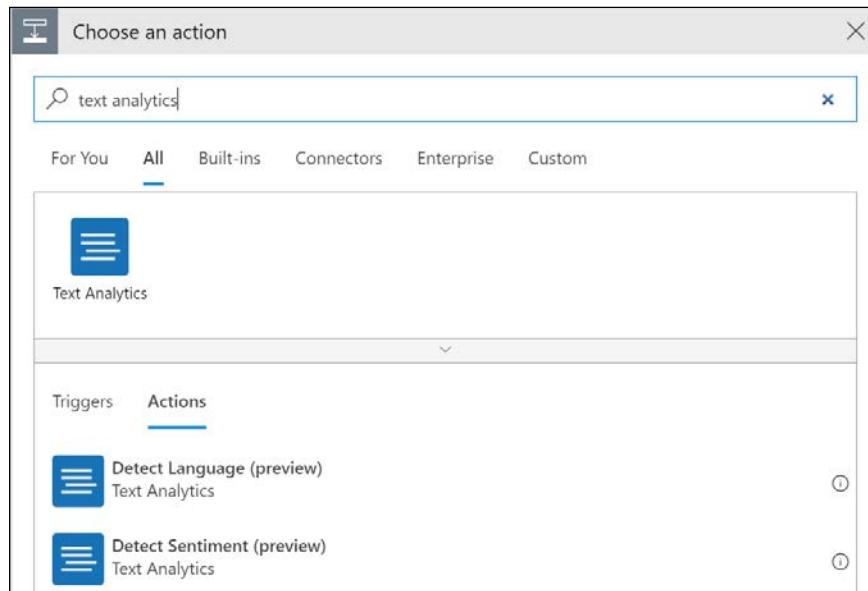
11. Add another action by clicking on **Add step**, typing **outlook**, and then selecting the **Reply to email** action:



12. Configure the new action; ensure that **Message Id** is set with the dynamic content, **Message Id**, and then type the reply that you'd like to send in the **Comment** box:



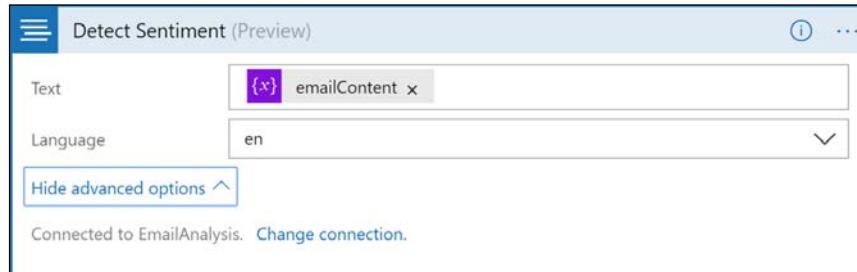
13. Add another action, type `text analytics`, and then select **Detect Sentiment (preview)**:



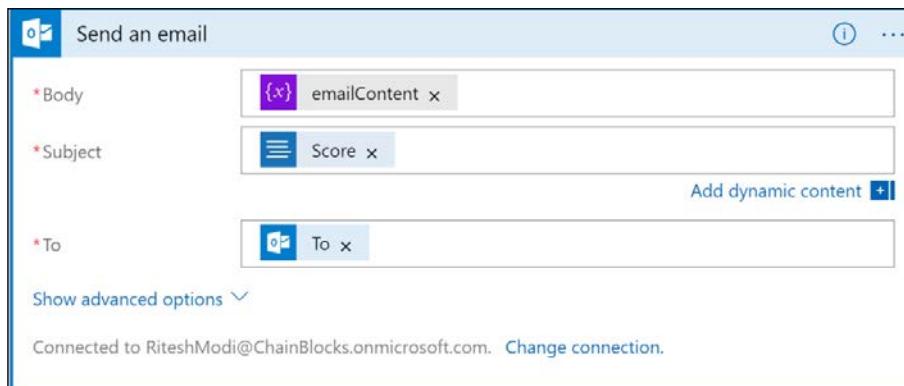
14. Configure the sentiment action, as shown in the following screenshot – both the endpoint and key values should be used here. Now click on the **Create** button, as demonstrated in the following screenshot:

*Connection Name	EmailAnalysis
*Account Key
Site URL	https://westeurope.api.cognitive.microsoft.com/text/analytics/v2.0

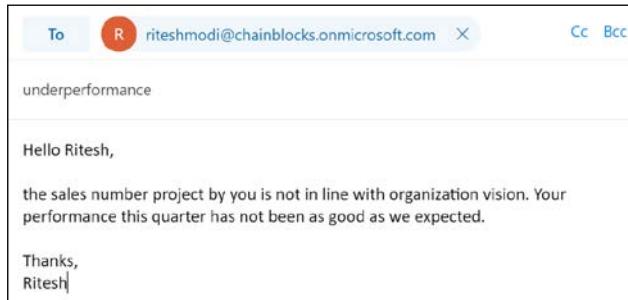
15. Provide the text to the action by adding dynamic content and selecting the previously created variable, `emailContent`. Then, click on **Show advanced options** and select `en` for **Language**:



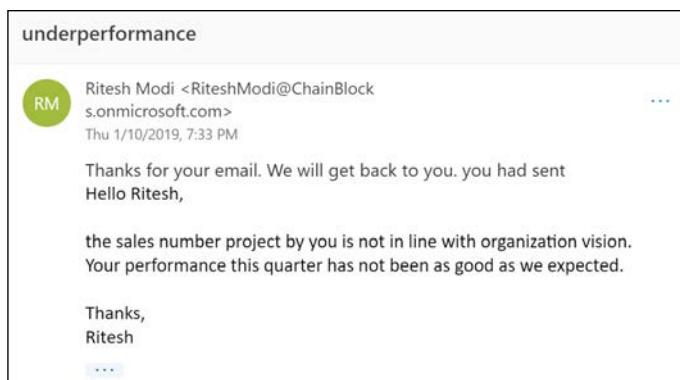
16. Next, add a new action by selecting **Outlook**, and then select **Send an email**. This action sends the original recipient the email content with the sentiment score in its subject. It should be configured as shown in the following screenshot. If the score is not visible in the dynamic contents window, click on the **See more** link beside it:



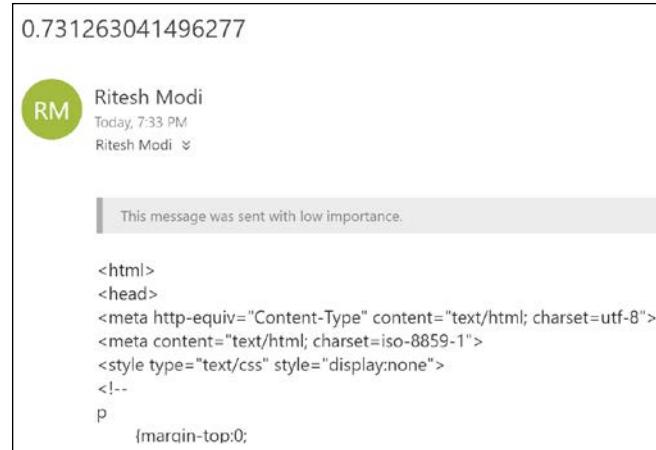
17. Save the logic app, navigate back to the overview page, and click on **Run** trigger. The trigger will check for new emails every three minutes, reply to the senders, perform sentiment analysis, and send an email to the original recipient. A sample email with negative connotations is sent to the given email ID:



18. After a few seconds, the logic app executes, and the sender gets the following reply:



19. The original recipient gets an email with the sentiment score and the original email text, as shown in the following screenshot:



Creating an end-to-end solution using serverless technologies

In this section, we will create an end-to-end solution comprising serverless technologies that we discussed in the previous sections.

The problem statement

The problem that we are going to solve here is that users and organizations are not notified regarding the expiration of any secret in their key vault, and applications stop working when they expire. Users are complaining that Azure does not provide the infrastructure to monitor key vault secrets, keys, and certificates.

Vision

Azure Key Vault is an Azure service that provides secure storage and access to credentials, keys, secrets, and certificates. It provides a vault that is backed up by hardware devices known as **Hardware Security Modules (HSM)**, and it forms the highest form of secure storage for secrets and keys.

Azure Key Vault allows the storage of these keys, secrets, and credentials, and they have an expiry date. For example, a certificate uploaded to the Azure Key Vault will expire in one year, or a secret will expire in two years. While it is one of the Azure security best practices to store sensitive information in Key Vault, Key Vault does not provide any infrastructure to monitor these secrets and notify users that these secrets are expiring in advance. In such situations, if the user is not monitoring their own Key Vault secrets actively, applications that rely on these secrets (such as connection strings and usernames) will stop working, and reactive measures will need to be undertaken to fix the application by renewing the secret in Key Vault.

The vision is to create a complete end-to-end solution using Azure services so that users are notified well in advance that Key Vault secrets are going to expire soon and that they need to take action in order to renew them.

Solution

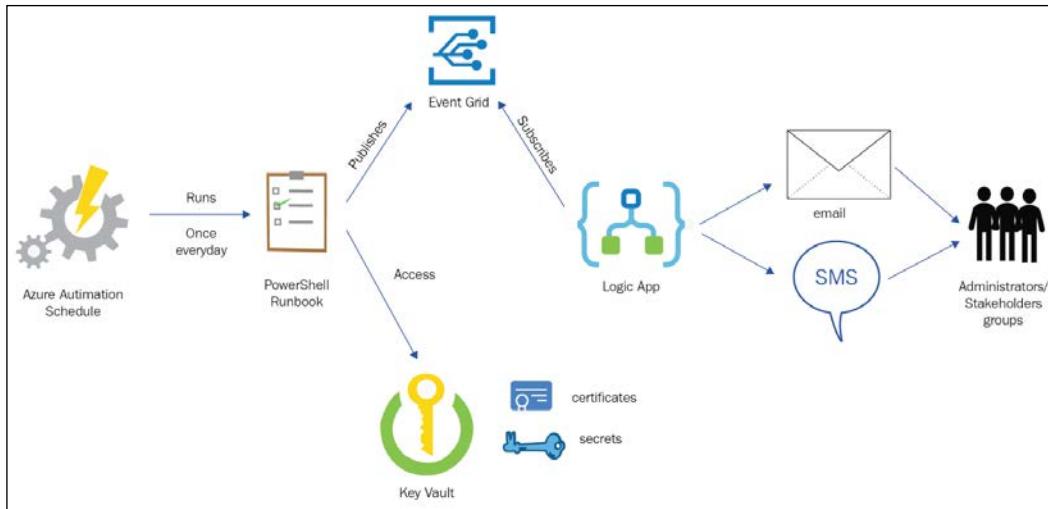
The solution to this problem is to combine multiple Azure services and integrate them so that users can be proactively notified of the expiration of secrets. The solution will send notifications using two channels - email and SMS.

The Azure services used to create this solution include the following:

- Azure Key Vault
- Azure AD
- Azure Event Grid
- Azure Automation
- Logic Apps
- Azure Functions
- SendGrid
- Twilio SMS

Architecture

The architecture of the solution comprises multiple services, as shown in the following diagram:



Let's go through each of these services and understand their roles and the functionality that they provide in the overall solution.

Azure Automation

Azure Automation provides runbooks, and these runbooks can be executed to run logic using PowerShell, Python, and other scripting languages. The runbook can be executed either on-premises or in the cloud, and provides rich infrastructure and facilities to create scripts. These scripts are known as **runbooks**. Typically, runbooks implement a scenario such as stopping or starting a virtual machine, or creating and configuring storage accounts. It is quite easy to connect to the Azure environment from runbooks with the help of assets such as variables, certificates, and connections.

In the current solution, we want to connect to Azure Key Vault, read all the secrets and keys stored within it, and fetch their expiry dates. These expiry dates should be compared with today's date and, if the expiry date is within a month, the runbook should raise a custom event on Event Grid using an Event Grid custom topic.

An Azure Automation runbook using a PowerShell script will be implemented to achieve this. Along with the runbook, a scheduler will also be created that will execute the runbook once a day at 12.00 am.

A custom Azure Event Grid topic

Once the runbook identifies that a secret or key is going to expire within a month, it will raise a new custom event and publish it to the custom topic created specifically for this purpose. Again, we will go into the details of the implementation in the next section.

Azure Logic Apps

A logic app is a serverless service that provides workflow capabilities. Our logic app will be configured to be triggered as and when an event is published on the custom Event Grid topic. After it is triggered, it will invoke the workflow and execute all the activities in it one after the other. Generally, there are multiple activities, but for the purpose of this example, we will invoke one Azure function that will send both email and SMS messages. In a full-blown implementation, these notification functions should be implemented separately in separate Azure functions.

Azure Functions

Azure Functions is used to notify users and stakeholders about the expiration of secrets and keys using email and SMS. SendGrid is used to send emails, while Twilio is used to send SMS messages from Azure Functions.

Prerequisites

You will need an Azure subscription with contributor rights at the very least.

Implementation

An Azure Key Vault should already exist. If not, it should be created.

This step should be performed if a new Azure Key Vault needs to be provisioned. Azure provides multiple ways in which to provision resources in Azure; prominent among them is Azure PowerShell and Azure CLI. Azure CLI is a command-line interface that works across platforms. The first task will be to provision a key vault in Azure. Azure PowerShell is used to provision the Azure Key Vault.

Before Azure PowerShell can be used to create a key vault, it is important to log in to Azure so that subsequent commands can be executed successfully and create a key vault.

Step 1

The first step is to prepare the environment for the sample. This involves logging in to Azure portal, selecting an appropriate subscription, and then creating a new Azure Resource Group and a new Azure Key Vault resource:

1. Execute the `Login-AzureRmAccount` command to log in to Azure. It will prompt for credentials in a new window.
2. After a successful login, if there are multiple subscriptions available for the login ID provided, they will all be listed. It is important to select an appropriate subscription; this can be done by executing the `Set-AzureRmContent` cmdlet:

```
Set-AzureRmContext -Subscription xxxxxxxx-xxxx-xxxx-xxxx-  
xxxxxxxxxx
```

3. Create a new resource group in your preferred location. In this case, the name of the resource group is `IntegrationDemo` and it is created in the `West Europe` region:

```
New-AzureRmResourceGroup -Name IntegrationDemo -Location "West  
Europe" -Verbose
```

4. Create a new Azure Key Vault; the name of the vault in this case is `keyvaultbook`, and it is enabled for deployment, template deployment, disk encryption, soft delete, and purge protection:

```
New-AzureRmKeyVault -Name keyvaultbook -ResourceGroupName  
IntegrationDemo -Location "West Europe" -EnabledForDeployment  
-EnabledForTemplateDeployment -EnabledForDiskEncryption  
-EnableSoftDelete -EnablePurgeProtection -Sku Standard -Verbose
```

The preceding command, when executed successfully, will create a new Azure Key Vault. The next step is to provide access to a service principal on the Key Vault.

Step 2

Instead of using an individual account to connect to Azure, Azure provides service principals, which are, in essence, service accounts that can be used to connect to Azure Resource Manager and execute activities. Adding a user to Azure tenant makes them available everywhere, including in all resource groups and resources, due to the nature of security inheritance in Azure. Access has to be explicitly revoked from resource groups for users if they are not allowed to access it. Service principals help by assigning granular access and control to resource groups, resources, and, if required, they can be given access to the subscription scope. They can also be assigned granular permissions, such as reader, contributor, or owner permissions.

In short, service principals should be the preferred mechanism to consume Azure services. They can be configured either with a password or with a certificate key. The process of provisioning a service principal with a password using PowerShell is as follows:

1. Create a service application using `new-AzurermAApplication` by passing values for the display name, identified URIs, home page, and password. The password should be a secure string.

2. A secure string can be generated using the `ConvertTo-SecureString` cmdlet, as shown in the following code:

```
ConvertTo-SecureString -String sysadminpassword -AsPlainText  
-Force
```

3. The service application can be created next. The return value from the command is stored in a variable so that it can be used in the following command:

```
$app = New-AzureRmADApplication -DisplayName "https://keyvault.  
book.com" -IdentifierUris "https://keyvault.book.com" -HomePage  
"https://keyvault.book.com" -Password (ConvertTo-SecureString  
-String sysadminpassword -AsPlainText -Force) -Verbose
```

4. After the service application has been created, a service principal based on the service application should be created. The application acts like a blueprint, and the principal acts like an instance of the application. We can create multiple principals using the same application in different tenants. The command to create a service principal is as follows:

```
New-AzureRmADServicePrincipal -ApplicationId $app.ApplicationId  
-DisplayName "https://keyvault.book.com" -Password (ConvertTo-  
SecureString -String sysadminpassword -AsPlainText -Force) -Scope  
"/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx" -Role  
Owner -StartDate ([datetime]::Now) -EndDate $([datetime]::now.  
AddYears(1)) -Verbose
```

5. `ApplicationId` from the previous command is used in this command by means of the `$app` variable. `StartDate` and `EndDate` are added for the service principal.

6. The important configuration values are the scope and role. The scope determines the access area for the service application; it is currently shown at the subscription level. Valid values for scope are as follows:

```
/subscriptions/{subscriptionId}  
/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}  
/subscriptions/{subscriptionId}/resourcegroups/  
{resourceGroupName}/providers/{resourceProviderNamespace}/  
{resourceType}/{resourceName}  
/subscriptions/{subscriptionId}/resourcegroups/  
{resourceGroupName}/providers/{resourceProviderNamespace}/  
{parentResourcePath}/{resourceType}/{resourceName}
```

The role provides permissions at the assigned scope. The valid values are as follows:

- Owner
- Contributor
- Reader
- Resource-specific permissions

In the previous command, owner permissions have been provided to the newly-created service principal.

Step 3

To create a service principal using certificates, the following steps should be executed:

1. Create a self-signed certificate or purchase a certificate: A self-signed certificate is used to create this example end-to-end application. For real-life deployments, a valid certificate should be purchased from a certificate authority.

To create a self-signed certificate, the following command can be executed. The self-signed certificate is exportable and stored in a personal folder on the local machine; it also has an expiry date:

```
$currentDate = Get-Date  
$expiryDate = $currentDate.AddYears(1)  
$finalDate = $expiryDate.AddYears(1)  
$servicePrincipalName = "https://automation.book.com"  
$automationCertificate = New-SelfSignedCertificate -DnsName  
$servicePrincipalName -KeyExportPolicy Exportable -Provider  
"Microsoft Enhanced RSA and AES Cryptographic Provider" -NotAfter  
$finalDate -CertStoreLocation "Cert:\LocalMachine\My"
```

2. **Export the newly created certificate:** The new certificate must be exported to the filesystem so that later, it can be uploaded to other destinations, such as Azure AD, to create a service principal.

The commands used to export the certificate to the local filesystem are shown next. Please note that this certificate has both public and private keys, and so while it is exported, it must be protected using a password, and the password must be a secure string:

```
$securepfxpwd = ConvertTo-SecureString -String 'password'  
-AsPlainText -Force # Password for the private key PFX certificate  
$cert1 = Get-Item -Path Cert:\LocalMachine\  
My\$($automationCertificate.Thumbprint)  
Export-PfxCertificate -Password $securepfxpwd -FilePath "C:\book\  
azureautomation.pfx" -Cert $cert1
```

The `Get-Item` cmdlet reads the certificate from the certificate store and stores it in the `$cert1` variable. The `Export-PfxCertificate` cmdlet actually exports the certificate in the certificate store to the filesystem. In this case, it is in the `C:\book` folder.

3. **Read the content from the newly generated PFX file:** An object of `X509Certificate` is created to hold the certificate in memory, and the data is converted to a Base64 string using the `System.Convert` function:

```
$newCert = New-Object System.Security.Cryptography.  
X509Certificates.X509Certificate -ArgumentList "C:\book\  
azureautomation.pfx", $securepfxpwd  
$newcertdata = [System.Convert]::ToBase64String($newCert.  
GetRawCertData())
```

Create an instance of the `PSADKeyCredential` class and provide values to its important properties. It is used while creating a new service principal. This class is available in the `Microsoft.Azure.Graph.RBAC.Version1_6.ActiveDirectory` namespace. A new Guid value is also generated for `keyid` property of `PSADKeyCredential`. The Base64-encoded cert value is assigned to the `CertValue` property, and values for both the `Startdate` and `EndDate` are also provided. When a service principal is created using this object, it will get configured according to the values provided here:

```
$keyid = [Guid]::NewGuid()  
$keyCredential = New-Object -TypeName Microsoft.Azure.Graph.RBAC.  
Version1_6.ActiveDirectory.PSADKeyCredential  
$keyCredential.StartDate = [datetime]::Now  
$keyCredential.KeyId = $keyid  
$keyCredential.CertValue = $newcertdata  
$keyCredential.EndDate = [datetime]::Now.AddYears(1)
```

4. **Create the service application and service principal in Azure:** We have already executed these commands once when creating the service principal with a password. This time, the key difference is that instead of a password, the `keyCredential` property is used. Finally, a service principal is created with owner rights.

We will be using this same principal to connect to Azure from the Azure Automation account. It is important that the application ID, tenant ID, subscription ID, and certificate thumbprint values are stored in a temporary location so that they can be used to configure subsequent resources:

```
$adAppName = http://automationconcertcred2
$adApp =New-AzureRmADApplication -DisplayName $adAppName
-HomePage $adAppName -IdentifierUris $adAppName -KeyCredentials
$keyCredential -Verbose
New-AzureRmADServicePrincipal -ApplicationId $adApp.ApplicationId.
Guid -Role owner
```

Step 4

At this stage, we have created the service principal and the key vault. However, the service principal still does not have access to the key vault. This service principal will be used to query and list all the secrets, keys, and certificates from the key vault, and it should have the necessary permissions to do so.

To provide the newly-created service principal permission to access the key vault, we will go back to the Azure PowerShell console and execute the following command:

```
Set-AzureRmKeyVaultAccessPolicy -VaultName keyvaultbook
-ResourceGroupName IntegrationDemo -ObjectId "ea36bc00-6eff-
4236-8c43-65c0c2e7e4cb" -PermissionsToKeys get,list,create
-PermissionsToCertificates get,list,import -PermissionsToSecrets
get,list -Verbose
```

Referring to the previous command block, take a look at the following points:

- `Set-AzureRmKeyVaultAccessPolicy` provides access permissions to users, groups, and service principals. It accepts the key vault name and the service principal object ID. This object is different from the application ID. The output of `New-AzureRmAdServicePrincipal` contains an `Id` property; while the value of `ObjectId` is this value:

<code>ServicePrincipalNames</code>	: { "values": ["urn:oid:2e84c2f...9983"] }
<code>ApplicationId</code>	: { "values": ["urn:oid:2e84c2f...9983"] }
<code>DisplayName</code>	: <code>IntegrationDemo</code>
<code>Id</code>	: <code>2e84c2f...9983</code>
<code>AzureId</code>	: { "values": ["urn:oid:2e84c2f...9983"] }
<code>Type</code>	: <code>servicePrincipal</code>

- `PermissionsToKeys` provides access to keys in the key vault, and the `get`, `list`, and `create` permissions are provided to this service principal. There is no write or update permission provided to this principal.
- `PermissionsToSecrets` provides access to secrets in the key vault, and the `get` and `list` permissions are provided to this service principal. There is no write or update permission provided to this principal.
- `PermissionsToCertificates` provides access to secrets in the key vault, and `get`, `import`, and `list` permissions are provided to this service principal. There is no write or update permission provided to this principal.

Step 5

Just like before, we will be using Azure PowerShell to create a new Azure Automation account within a resource group. Before creating a resource group and an automation account, a connection to Azure should be established. However, this time, use the credentials for Service Principal to connect to Azure. The steps for connecting to Azure using Service Principal are shown next:

1. The command to connect to Azure using the service application is as follows:

```
Login-AzureRmAccount -ServicePrincipal -CertificateThumbprint  
"003B0D26705C792DB60823DA5804A0897160C306" -ApplicationId  
"xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx" -Tenant "xxxxxxxx-xxxx-  
xxxx-xxxx-xxxxxxxxxxxx"
```

2. Here, `applicationId` is available after executing the `New-AzureRmADApplication` cmdlet, and the tenant ID and the subscription ID can be retrieved using the following command. The subscription ID will be needed in subsequent commands:

```
Get-AzureRmContext
```

3. After connecting to Azure, a new resource containing the resources for the solution and a new Azure Automation account should be created. We are naming the resource group `VaultMonitoring`, and creating it in the West Europe region. We will be creating the remainder of the resources in this resource group as well:

```
$IntegrationResourceGroup = "VaultMonitoring"  
$rgLocation = "West Europe"  
$automationAccountName = "MonitoringKeyVault"
```

```
New-AzureRmResourceGroup -name $IntegrationResourceGroup -Location  
$rgLocation  
New-AzureRmAutomationAccount -Name $automationAccountName  
-ResourceGroupName $IntegrationResourceGroup -Location $rgLocation  
-Plan Free
```

4. Next, create three automation variables. The values for these, that is, the subscription ID, tenant ID, and application ID, should already be available using the previous steps:

```
New-AzureRmAutomationVariable -Name "azuresubscriptionid"  
-AutomationAccountName $automationAccountName -ResourceGroupName  
$IntegrationResourceGroup -Value "xxxxxxxx-xxxx-xxxx-xxxx-  
xxxxxxxxxxxx" -Encrypted $true  
  
New-AzureRmAutomationVariable -Name "azuretenantid"  
-AutomationAccountName $automationAccountName -ResourceGroupName  
$IntegrationResourceGroup -Value "xxxxxxxx-xxxx-xxxx-xxxx-  
xxxxxxxxxxxx" -Encrypted $true  
New-AzureRmAutomationVariable -Name "azureappid"  
-AutomationAccountName $automationAccountName -ResourceGroupName  
$IntegrationResourceGroup -Value "xxxxxxxx-xxxx-xxxx-xxxx-  
xxxxxxxxxxxx" -Encrypted $true
```

5. Now it's time to upload a certificate that will be used to connect to Azure from Azure Automation:

```
$securepfxpwd = ConvertTo-SecureString -String 'password'  
-AsPlainText -Force # Password for the private key PFX certificate  
New-AzureRmAutomationCertificate -Name  
"RitestSubscriptionCertificate" -Path "C:\book\  
azureautomation.pfx" -Password $securepfxpwd  
-AutomationAccountName $automationAccountName -ResourceGroupName  
$IntegrationResourceGroup
```

6. The next step is to install PowerShell modules related to Key Vault and Event Grid in the Azure Automation account, as these modules are not installed by default.
7. From the Azure portal, navigate to the already-created `VaultMonitoring` resource group by clicking on the **Resource Groups** icon in the left-hand menu.

8. Click on the already-provisioned Azure Automation account, **MonitoringKeyVault**, and then click on **Modules** in the left-hand menu, as shown in the following screenshot:

NAME	LAST MODIFIED	STATUS	VERSION
AuditPolicyDsc	12/13/2018, 3:10 PM	Available	1.1.0.0
Azure	12/13/2018, 3:02 PM	Available	1.0.3
Azure.Storage	12/13/2018, 3:07 PM	Available	1.0.3
AzureRM.Automation	12/13/2018, 3:05 PM	Available	1.0.3
AzureRM.Compute	12/13/2018, 3:04 PM	Available	1.2.1
AzureRM.Profile	12/13/2018, 3:05 PM	Available	1.0.3
AzureRM.Resources	12/13/2018, 3:06 PM	Available	1.0.3
AzureRM.Sql	12/13/2018, 3:06 PM	Available	1.0.3
AzureRM.Storage	12/13/2018, 3:07 PM	Available	1.0.3
ComputerManagementDsc	12/13/2018, 3:08 PM	Available	5.0.0.0
GPORegistryPolicyParser	12/13/2018, 3:10 PM	Available	0.2
Microsoft.PowerShell.Core	12/13/2018, 3:02 PM	Available	0.0
Microsoft.PowerShell.Diagnostics	12/13/2018, 3:01 PM	Available	
Microsoft.PowerShell.Management	12/13/2018, 3:01 PM	Available	

The Event Grid module is dependent on the `AzureRM.profile` module, and so we have to install it before the Event Grid module.

9. Click on **Browse Gallery** in the top menu and type `Azurerm.profile` in the search box, as shown in the following screenshot:

Home > VaultMonitoring > MonitoringKeyVault - Modules > Browse Gallery

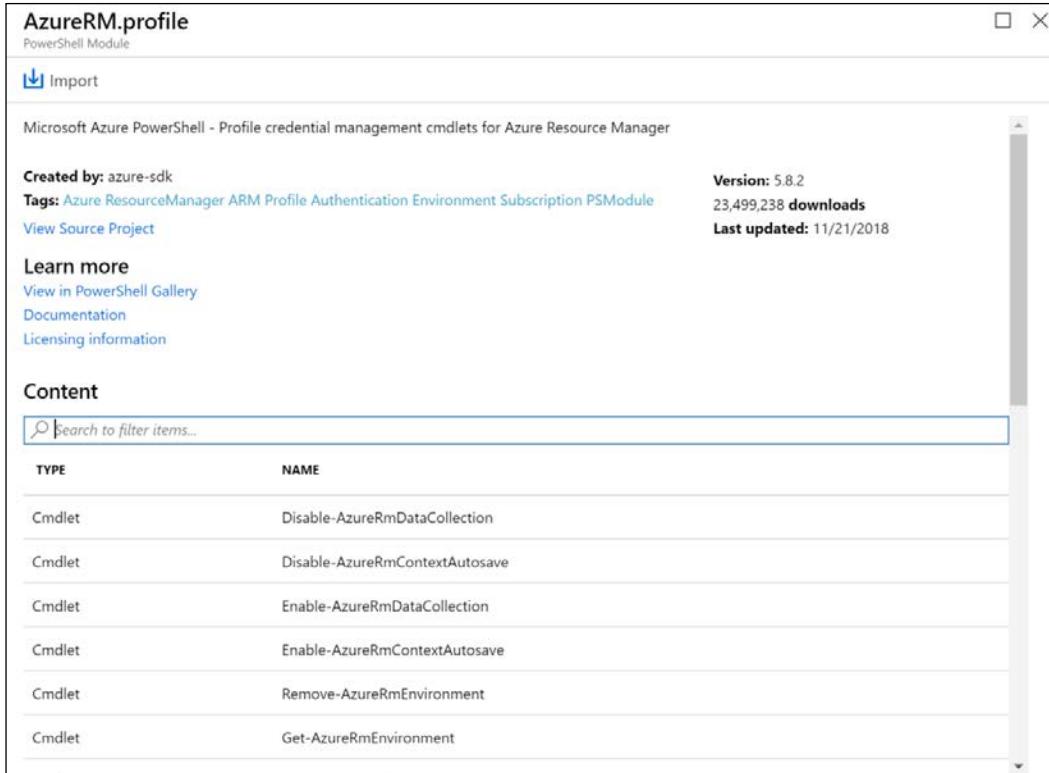
Browse Gallery

X ▼

Azurerm.profile
Microsoft Azure PowerShell - Profile credential management cmdlets for Azure Resource Manager
Tags: Azure ResourceManager ARM Profile Authentication Environment Subscription PSModule

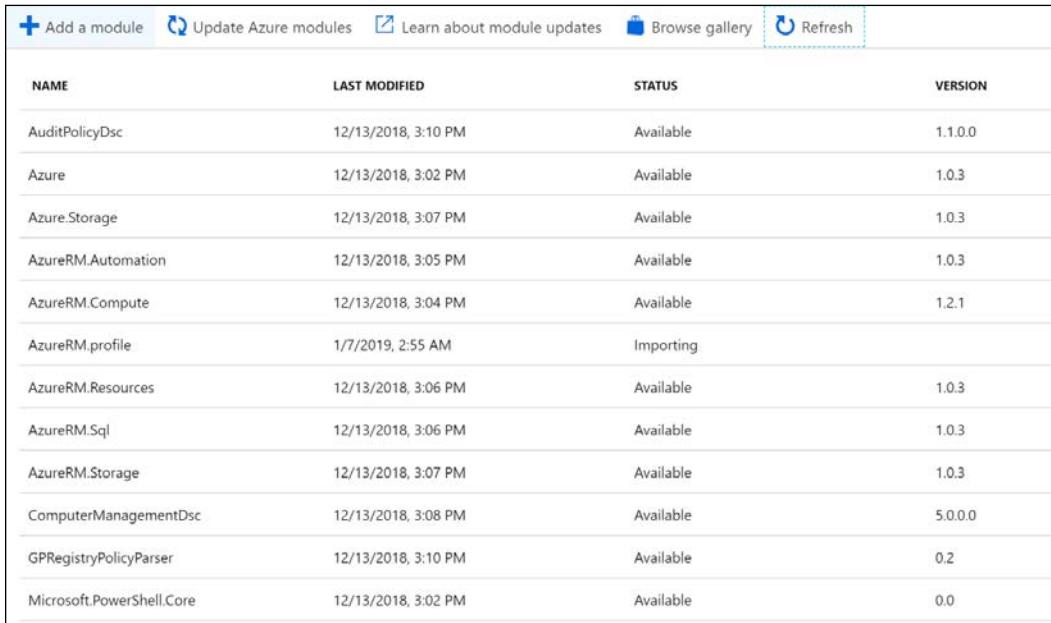
Created by: azure-sdk
23499238 downloads
Last updated: 11/21/2018

- From the search results, select `AzureRM.profile` and click on the **Import** button in the top menu. Finally, click on the **OK** button. This step takes a few seconds to complete; after a few seconds, the module should be installed, as shown in the following screenshot:



11. The status of the installation can be checked from the **Module** menu item.

The following screenshot demonstrates how we can import a module:



The screenshot shows a table of Azure PowerShell modules. The columns are NAME, LAST MODIFIED, STATUS, and VERSION. One module, 'AzureRM.profile', is shown as 'Importing'.

NAME	LAST MODIFIED	STATUS	VERSION
AuditPolicyDsc	12/13/2018, 3:10 PM	Available	1.1.0.0
Azure	12/13/2018, 3:02 PM	Available	1.0.3
Azure.Storage	12/13/2018, 3:07 PM	Available	1.0.3
AzureRM.Automation	12/13/2018, 3:05 PM	Available	1.0.3
AzureRM.Compute	12/13/2018, 3:04 PM	Available	1.2.1
AzureRM.profile	1/7/2019, 2:55 AM	Importing	
AzureRM.Resources	12/13/2018, 3:06 PM	Available	1.0.3
AzureRM.Sql	12/13/2018, 3:06 PM	Available	1.0.3
AzureRM.Storage	12/13/2018, 3:07 PM	Available	1.0.3
ComputerManagementDsc	12/13/2018, 3:08 PM	Available	5.0.0.0
GPRRegistryPolicyParser	12/13/2018, 3:10 PM	Available	0.2
Microsoft.PowerShell.Core	12/13/2018, 3:02 PM	Available	0.0

12. Perform steps 9, 10, and 11 again in order to import and install the AzureRM.EventGrid module:

The screenshot shows the PowerShell Gallery page for the `AzureRM.EventGrid` module. The top navigation bar includes 'Import' and 'View in PowerShell Gallery'. Below the title, it says 'Microsoft Azure PowerShell - EventGrid service cmdlets for Azure Resource Manager'. Key metadata includes 'Created by: azure-sdk', 'Tags: Azure ResourceManager ARM EventGrid PSModule', 'Dependencies: AzureRM.Profile (>= 5.4.0)', 'Version: 0.3.7', '2,877,032 downloads', and 'Last updated: 8/14/2018'. A 'Learn more' section links to 'View in PowerShell Gallery', 'Documentation', and 'Licensing information'. The 'Content' section lists cmdlets with their names:

TYPE	NAME
Cmdlet	New-AzureRmEventGridTopic
Cmdlet	Get-AzureRmEventGridTopic
Cmdlet	Set-AzureRmEventGridTopic
Cmdlet	New-AzureRmEventGridTopicKey
Cmdlet	Get-AzureRmEventGridTopicKey
Cmdlet	Remove-AzureRmEventGridTopic

13. Perform steps 9, 10, and 11 again in order to import and install the AzureRM.KeyVault module:

The screenshot shows the PowerShell Gallery interface. A search bar at the top contains the query 'keyvault'. Below the search bar, several modules are listed in a grid format. The first module in the list is 'AzureRM.KeyVault' by 'azure-sdk', which has been selected. To the right of the list, a detailed view of this module is displayed. This view includes the module's name ('AzureRM.KeyVault'), its description ('Microsoft Azure PowerShell - KeyVault service cmdlets for Azure Resource Manager'), the developer ('Created by: azure-sdk'), download statistics ('8578130 downloads'), and the last update date ('Last updated: 8/29/2018'). It also shows the module's tags ('Tags: Azure.ResourceManager ARM.KeyVault PSModule'), its version ('Version: 1.0.0'), and its dependencies ('Dependencies: AzureRM.Profile (≥ 5.5.1)'). Below this information is a 'Learn more' section with links to 'View in PowerShell Gallery', 'Documentation', and 'Licensing information'. At the bottom of this section is a 'Content' table with a header row 'TYPE NAME'. The table lists six cmdlets: Add-AzureKeyVaultCertificate, Update-AzureKeyVaultCertificate, Stop-AzureKeyVaultCertificateOperation, Get-AzureKeyVaultCertificateOperation, Import-AzureKeyVaultCertificate, and Add-AzureKeyVaultCertificateContent.

Step 6

The command that's used to create an Event Grid topic using PowerShell is as follows:

```
New-AzureRmEventGridTopic
```

The process of creating an Event Grid topic using the Azure portal is as follows:

1. From the Azure portal, navigate to the already-created `VaultMonitoring` resource group by clicking on the **Resource Groups** icon in the left-hand menu.
2. Next, click on **+Add** and search for `Event Grid Topic` in the search box; select it and then click on the **Create** button:

The screenshot shows the Azure Marketplace search results for 'Event Grid Topic'. The search bar at the top contains 'Event Grid Topic'. Below it, there are filters for Pricing (All), Operating System (All), and Publisher (All). The results section displays three items:

NAME	PUBLISHER	CATEGORY
Event Grid Topic	Microsoft	
Event Grid Domain	Microsoft	
Azure Blockchain Workbench	Microsoft	Compute

To the right of the results, there is a detailed description of 'Event Grid Topic' from Microsoft. It explains that Event Grid Topics provide a SAS authenticated DNS entry point for custom events. It also lists benefits such as sending custom events to Azure Event Grid, leveraging its integration with other services, and reliably delivering events to Azure or 3rd party services. A 'Save for later' button is available, and a 'Create' button is located at the bottom.

- Fill in the appropriate values in the resultant form by providing a name, selecting a subscription, selecting the newly-created resource group, the location, and the event schema:

The screenshot shows the 'Create Topic' dialog box for Event Grid. The fields are as follows:

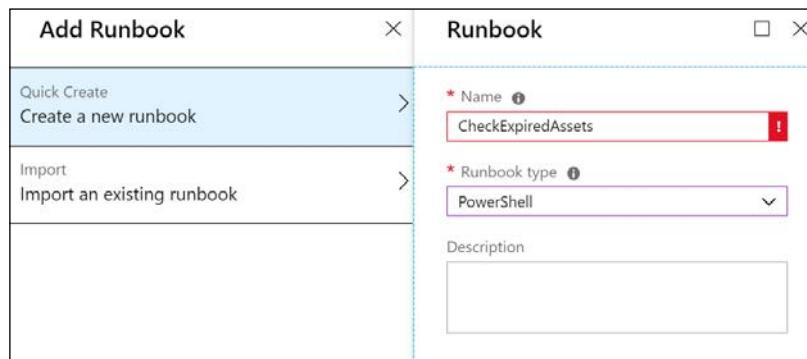
- Name:** ExpiredAssetsKeyVaultEvents
- Subscription:** RiteshSubscription
- Resource group:** VaultMonitoring (with a 'Create new' option)
- Location:** West Europe
- Event Schema:** Event Grid Schema

A 'Create' button is located at the bottom of the dialog.

Step 7

This step will focus on creating an Azure Automation Account and PowerShell runbooks that will contain the core logic of reading Azure Key Vaults and retrieving secrets stored within it. The steps required for configuring Azure Automation are mentioned next:

1. **Create the Azure Automation runbook:** From the Azure portal, navigate to the already-created **Vaultmonitoring** resource group by clicking on the **Resource Groups** icon in the left-hand menu:
 1. Click on the already-provisioned Azure Automation account, **MonitoringKeyVault**. Then, click on **Runbooks** in the left-hand menu, and click on **+Add a Runbook** from the top menu.
 2. Click on **Create a new Runbook** and provide a name. Let's call this runbook **CheckExpiredAssets**, and then set the **Runbook type** as **PowerShell**:



2. **Code the runbook:** Declare a few variables to hold the subscription ID, tenant ID, application ID, and certificate thumbprint information. These values are should be stored in Azure Automation variables, and the certificate should be uploaded to Automation certificates. The key used for the uploaded certificate is **RiteshSubscriptionCertificate**. The values are retrieved from these stores and are assigned to the variables, as shown next:

```
$subscriptionID = get-AutomationVariable "azuresubscriptionid"  
$tenantID = get-AutomationVariable "azuretenantid"  
$applicationId = get-AutomationVariable "azureappid"  
$cert = get-AutomationCertificate "RiteshSubscriptionCertificate"  
$certThumbprint = ($cert.Thumbprint).ToString()
```

The next code within the runbook helps to log in to Azure using the service principal with values from variables declared previously. Also, the code selects an appropriate subscription. The code is shown next.

```
Login-AzureRmAccount -ServicePrincipal -CertificateThumbprint  
$certThumbprint -ApplicationId $applicationId -Tenant $tenantID  
Set-AzureRmContext -SubscriptionId $subscriptionID
```

Since Azure Event Grid was provisioned in *step 6* of this section, its endpoint and keys are retrieved using the `Get-AzureRmEventGridTopic` and `Get-AzureRmEventGridTopicKey` cmdlets.

Every Azure Event Grid generates two keys – primary and secondary; the first key reference is taken as follows:

```
$eventGridName = "ExpiredAssetsKeyVaultEvents"  
$eventGridResourceGroup = "VaultMonitoring"  
$topicEndpoint = (Get-AzureRmEventGridTopic -ResourceGroupName  
$eventGridResourceGroup -Name $eventGridName).Endpoint  
$keys = (Get-AzureRmEventGridTopicKey -ResourceGroupName  
$eventGridResourceGroup -Name $eventGridName).Key1
```

Next, all key vaults that were provisioned within the subscription are retrieved using iteration. While looping, all secrets are retrieved using the `Get-AzureKeyVaultSecret` cmdlet.

The expiry date of each secret is compared to current date, and if the difference is less than a month, then it generates an Event Grid event and publishes it using the `Invoke-WebRequest` command.

The same steps are executed for certificates stored within the key vault. The cmdlet used to retrieve all the certificates is `Get-AzureKeyVaultCertificate`.

The event that is published to Event Grid should be in the JSON array. The generated message is converted to JSON using the `ConvertTo-Json` cmdlet, and then converted to an array by adding [and] as a prefix and suffix.

In order to connect to Azure Event Grid and publish the event, the sender should supply the key in its header. The request will fail if this data is missing in the request payload:

```
$keyvaults = Get-AzureRmKeyVault  
foreach($vault in $keyvaults) {  
$secrets = Get-AzureKeyVaultSecret -VaultName $vault.VaultName  
foreach($secret in $secrets) {  
if( ![string]::IsNullOrEmpty($secret.Expires) ) {  
if($secret.Expires.AddMonths(-1) -lt [datetime]::Now)  
{  
$secretDataMessage = @{
```

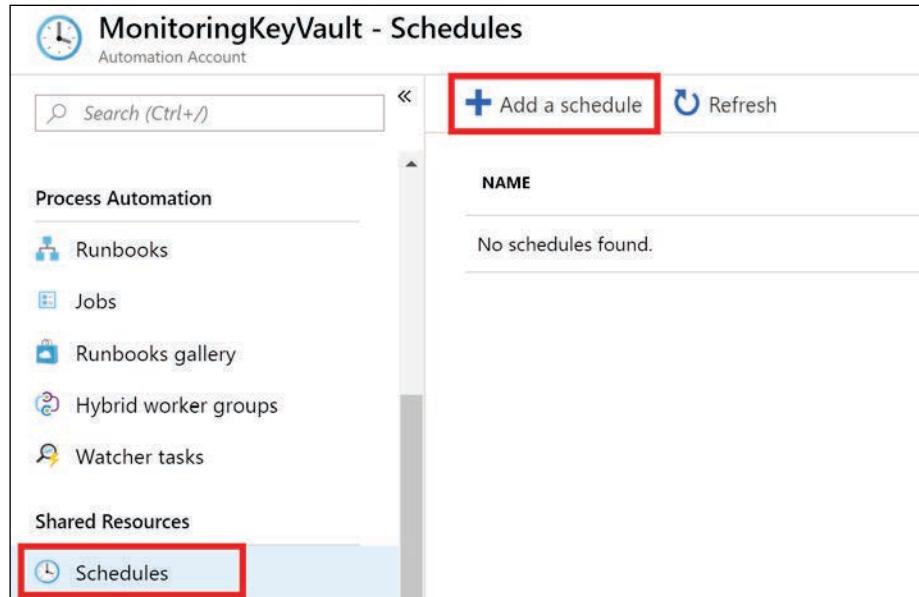
```
id = [System.guid]::NewGuid()
subject = "Secret Expiry happening soon !!"
eventType = "Secret Expiry"
eventTime = [System.DateTime]::UtcNow
data = @{
    "ExpiryDate" = $secret.Expires
    "SecretName" = $secret.Name.ToString()
    "VaultName" = $secret.VaultName.ToString()
    "SecretCreationDate" = $secret.Created.ToString()
    "IsSecretEnabled" = $secret.Enabled.ToString()
    "SecretId" = $secret.Id.ToString()
}
}
...
Invoke-WebRequest -Uri $topicEndpoint -Body $finalBody -Headers
$header -Method Post -UseBasicParsing
}
}
Start-Sleep -Seconds 5
}
}
```

Publish the runbook by clicking on the **Publish** button, as shown in the following screenshot:



3. **Scheduler:** Create an Azure Automation scheduler asset to execute this runbook once every day at 12.00 am. The steps for creating an Azure Automation Schedule are mentioned next:

1. Click on **Schedules** from the left-hand menu of Azure Automation and click on the **+Add a schedule** button in the top menu, as shown in the following screenshot:



2. Provide scheduling information in the resulting form:

New Schedule

* Name
ExecuteMonitoringRunbook ✓

Description
This schedule execute the Azure key vault monitoring runbook every day one at 12:00 AM to check for expired assets ✓

* Starts ⓘ
2019-01-11 12:00 AM

UTC

Recurrence
Once Recurring

* Recur every
1 Day

Set expiration
Yes No

Expires
Never

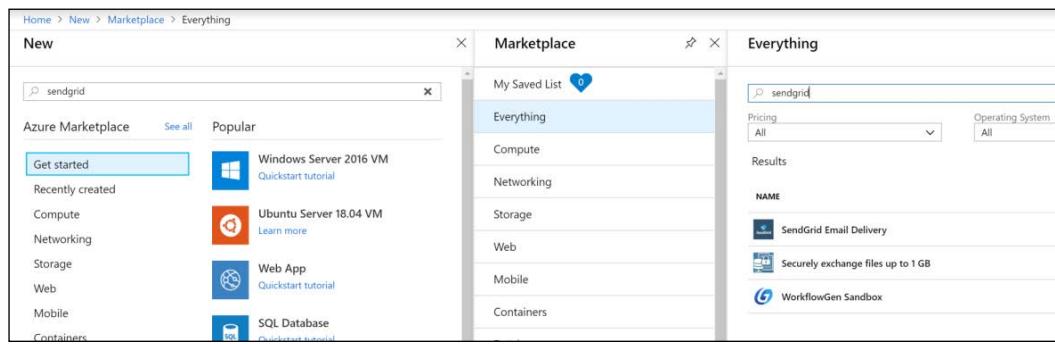
Create

This should conclude the configuration of the Azure Automation account.

Step 8

In this step, we will be creating a new SendGrid resource. The SendGrid resource is used to send emails from the application without needing to install a **Simple Mail Transfer Protocol (SMTP)** server. It provides a REST API and a C# **Software Development Kit (SDK)**, by means of which it is quite easy to send bulk emails. In the current solution, Azure Functions will be used to invoke the SendGrid APIs to send emails, and so this resource needs to be provisioned. This resource has separate costing and is not covered as part of the Azure cost; there is a free tier available and can be used for sending emails:

1. A SendGrid resource is created just like any other Azure resource. Search for **sendgrid**, and we will get **SendGrid Email Delivery** in the results:



2. Select the resource and click on the Create button to open its configuration form:

SendGrid Email Delivery

SendGrid

SendGrid is the world's largest cloud-based service for delivering email that matters. SendGrid's proven platform successfully delivers over 18B transactional and marketing related emails each month for Internet and mobile-based customers like Airbnb, Pandora, Hubspot, Spotify, Uber and FourSquare as well as more traditional enterprises like Walmart, Intuit and Costco.

Azure customers receive up to 25,000 emails per month for free with paid packages starting at only \$9.95 per month. For customers requiring ability to send larger email volumes, SendGrid also offers Silver, Gold, Platinum and Premier packages, which include a dedicated IP as well as additional IP and user management features.

[Save for later](#)

Statistics Overview

REQUESTS: 483,418 | UNIQUE OPEN: 17.61% | UNIQUE CLICKS: 3.8% | REQUESTS RECEIVED: 62.04% | SENDER: 0.22% | RECIPIENT: 0.07% | SPAM REPORTS: 0.00%

2015-02-24 2015-03-25

REQUESTS RECEIVED
RECIPIENT
SENDER
SPAM REPORTS
BLOCKED
OPEN REPORTS
UNSUBSCRIBE GROUPS
UNSUBSCRIBE URLS

Select a software plan

Free
25,000 emails per month.

Create

3. Select an appropriate pricing tier, as shown in the following screenshot:

Choose your pricing tier

For customers requesting the Premier Volume Plan (5 million or more emails/month), a promotional code is required to activate the discount rate. To get started, email your request to accountssuccess@sendgrid.com

★ Recommended | [View all](#)

S2 Silver	S1 Bronze	F1 Free
100,000 emails/month	40,000 emails/month	25,000 emails/month
 Advanced Reporting Open, Click, Bounce, Unsu...	 Advanced Reporting Open, Click, Bounce, Unsu...	 Advanced Reporting Open, Click, Bounce, Unsu...
 Custom Integration A... SMTP API, Web API, Event ...	 Custom Integration A... SMTP API, Web API, Event ...	 Custom Integration A... SMTP API, Web API, Event ...
 Advanced Delivery Fe... DKIM, SPF, Reputation Mo...	 Advanced Delivery Fe... DKIM, SPF, Reputation Mo...	 Advanced Delivery Fe... DKIM, SPF, Reputation Mo...
 Dedicated IP Address IP Whitelabeling, Automat...	 Support 24x7 Phone, Chat, Email S...	 Support 24x7 Phone, Chat, Email S...
 Sub-User Management Create and manage sub-us...		
 Support 24x7 Phone, Chat, Email S...		
79.95 USD/MONTH (ESTIMATED)	9.95 USD/MONTH	0.00 USD/MONTH

4. Provide the appropriate contact details, as listed in the following screenshot:

Contact Information □ X

Enter your contact details.

The provided information will be used as contact info for support agents / technical contacts.

* First Name
Ritesh ✓

* Last Name
Modi ✓

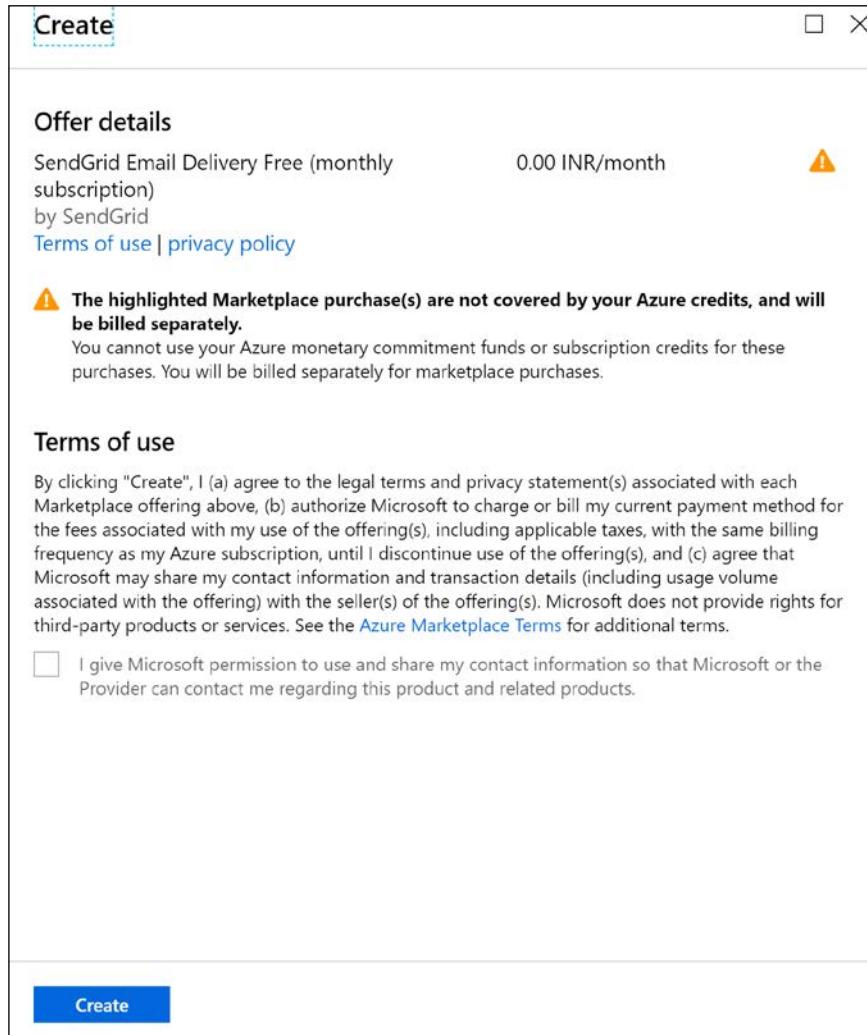
* Email
callritz@hotmail.com ✓

Company
self

Website
automationnext.wordpress.com

OK

5. Accept the **Terms of use** checkbox, as shown in the following screenshot:



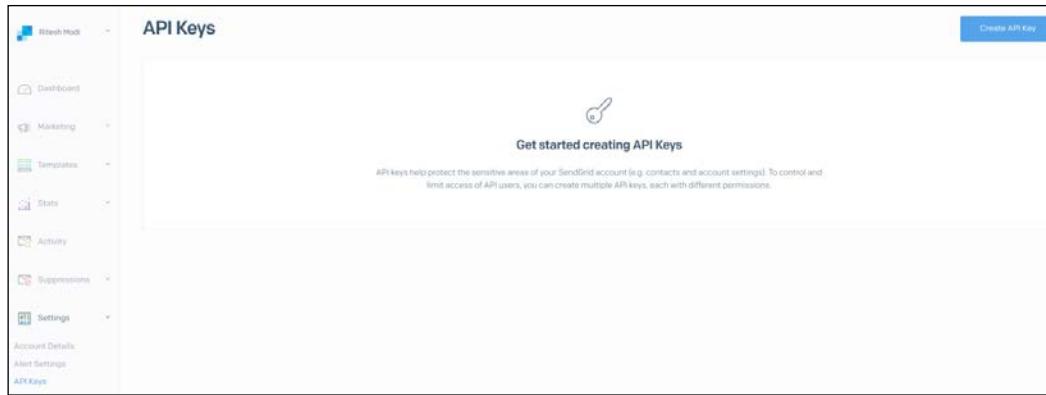
6. Complete the form and then click on the **Create** button, as shown in the following screenshot:

The screenshot shows the 'Create a New SendGrid A...' dialog box. The form includes the following fields:

- Name:** keyvaultmonitoring
- Password:** (redacted)
- Confirm Password:** (redacted)
- Subscription:** RiteshSubscription
- Resource group:** VaultMonitoring (radio button selected for 'Use existing')
- Pricing tier:** free
- Promotion Code:** (empty field)
- Contact Information:** Completed.
- Legal terms:** Legal terms accepted

At the bottom, there are two buttons: **Create** (highlighted in blue) and **Automation options**.

7. After the resource is provisioned, click on the **Manage** button in the top menu – this will open the SendGrid website. The website may request email configuration. Then, select **API Keys** from the **Settings** section and click on the **Create API Key** button:



8. From the resulting window, select **Full Access** and click on the **Create & View** button. This will create the keys for the SendGrid resource; keep a note of this key, as it will be used with the Azure Function configuration for SendGrid:

A screenshot of the 'Create API Key' dialog box. It has fields for 'API Key Name' (with a required asterisk) and 'API Key Permissions' (also with a required asterisk). Three permission levels are listed:

- Full Access**: Selected (radio button is blue). Description: Allows the API key to access GET, PATCH, PUT, DELETE, and POST endpoints for all parts of your account, excluding billing.
- Restricted Access**: Unselected (radio button is white). Description: Customize levels of access for all parts of your account, excluding billing.
- Billing Access**: Unselected (radio button is white). Description: Allows the API key to access billing endpoints for the account. (This is especially useful for Enterprise or Partner customers looking for more advanced account management.)

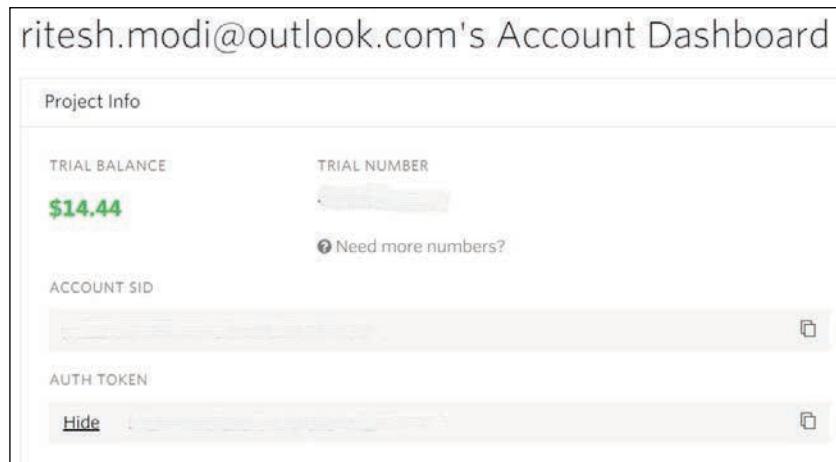
At the bottom are 'Cancel' and 'Create & View' buttons.

Step 9

In this step we will be creating a new Twilio account. Twilio is used for sending bulk SMS messages. To create an account with Twilio, navigate to [twilio.com](https://www.twilio.com) and create a new account. After successfully creating an account, a mobile number is generated that can be used to send SMS messages to receivers:



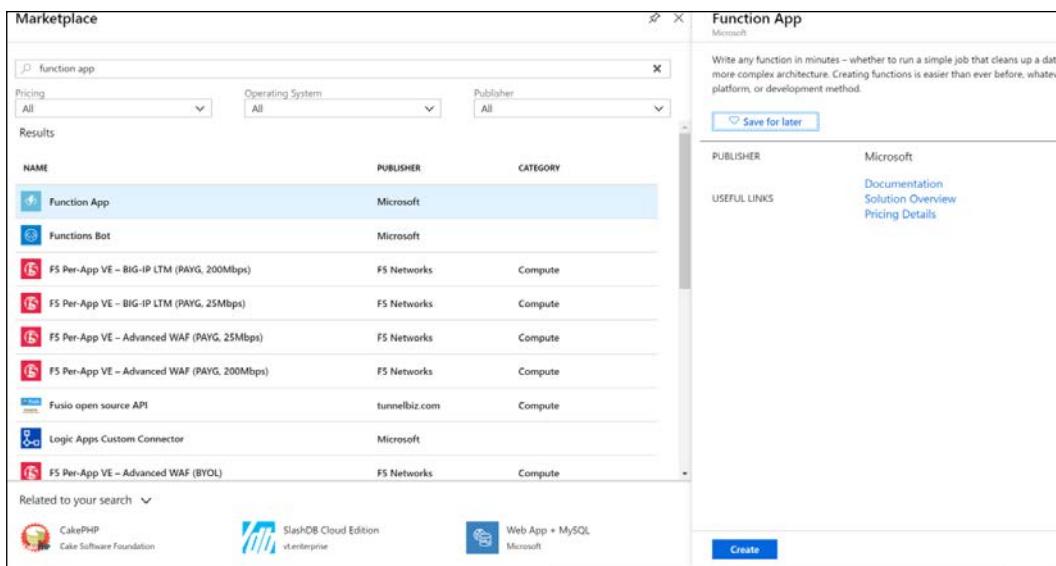
The Twilio account provides both production and test keys. Copy the test key and token to a temporary location, such as Notepad, as they will be required later within Azure Functions:



Step 10

In this step we will be creating new Azure function responsible for sending emails and SMS notifications. The purpose of the Azure function within the solution is to send notification messages to users regarding the expiry of secrets in the key vault. A single function will be responsible for sending both emails and SMS messages – note that this could have been divided into two separate functions. The first step is to create a new function app and host a function within it:

- As we have done before, navigate to your resource group, click on the **+Add** button in the top menu, and search for the **function app** resource. Then, click on the **Create** button to get the **Function App** form:



2. Fill in the **Function App** form and click on the **Create** button. The name of the function app must be unique across Azure:

The screenshot shows the 'Function App' creation dialog. The 'App name' field is set to 'NotificationFunctionAppBook'. The 'Subscription' is 'RiteshSubscription'. The 'Resource Group' is 'VaultMonitoring'. The 'OS' is 'Windows'. The 'Hosting Plan' is 'Consumption Plan'. The 'Location' is 'West Europe'. The 'Runtime Stack' is '.NET'. The 'Storage' section shows 'Create new' selected for a storage account named 'notificationfuna2a9'. At the bottom, there are 'Application Insights' settings for 'NotificationFunctionAppBook' and 'Create' and 'Automation options' buttons.

Function App

Create

* App name
NotificationFunctionAppBook .azurewebsites.net

* Subscription
RiteshSubscription

* Resource Group
VaultMonitoring

* OS
Windows Linux (Preview)

* Hosting Plan
Consumption Plan

* Location
West Europe

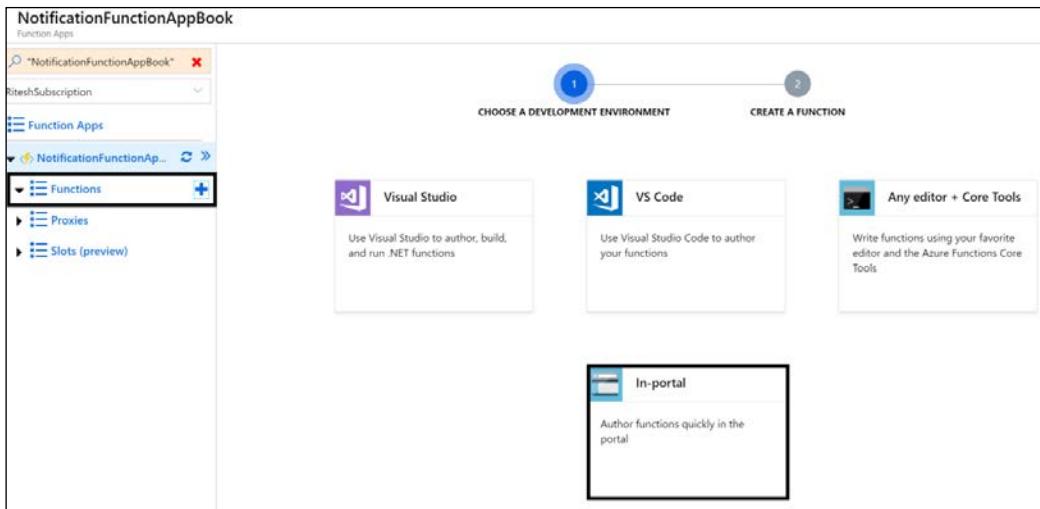
* Runtime Stack
.NET

* Storage
Create new Use existing
notificationfuna2a9

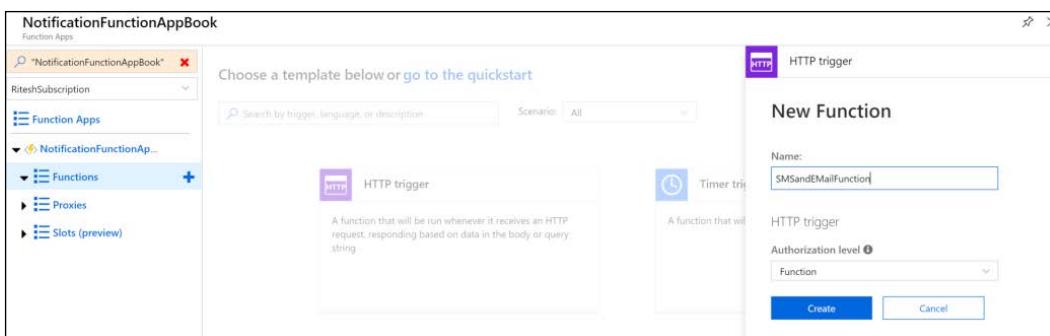
Application Insights
NotificationFunctionAppBook >

Create Automation options

- Once the function app is provisioned, create a new function called SMSandEMailFunction by clicking on the + button next to the **Functions** item in the left-hand menu. Then, select **In-portal** from the central dashboard. This is shown next:



- Select **HTTP trigger** and name it **SMSandEMailFunction**. Then, click on the **Create** button; the **Authorization level** option can be any value:



5. Remove the default code, replace it with the code shown in the following listing, and then click on the **Save** button in the top menu:

```
#r "SendGrid"
#r "Newtonsoft.Json"
#r "Twilio.Api"
using System.Net;
using System;
using SendGrid.Helpers.Mail;
using Microsoft.Azure.WebJobs.Host;
using Newtonsoft.Json;
using Twilio;
using System.Configuration;
public static HttpResponseMessage Run(HttpRequestMessage req,
TraceWriter log, out Mail message,out SMSMessage sms)
{
    log.Info("C# HTTP trigger function processed a request.");
    string alldata = req.Content.ReadAsStringAsync().GetAwaiter().
    GetResult();
    message = new Mail();
    var personalization = new Personalization();
    personalization.AddBcc(new Email(ConfigurationManager.AppSettings[
    "bccStakeholdersEmail"]));
    personalization.AddTo(new Email(ConfigurationManager.AppSettings[
    "toStakeholdersEmail"]));
    var messageContent = new Content("text/html", alldata);
    message.AddContent(messageContent);
    message.AddPersonalization(personalization);
    message.Subject = "Key Vault assets Expiring soon..";
    message.From = new Email(ConfigurationManager.
    AppSettings["serviceEmail"]);
    string msg = alldata;
    sms = new SMSMessage();
    sms.Body = msg;
    sms.To = ConfigurationManager.AppSettings["adminPhone"];
    sms.From = ConfigurationManager.AppSettings["servicePhone"];
    return req.CreateResponse(HttpStatusCode.OK, "Hello ");
}
```

6. Click on the function app name in the left-hand menu and click again on the **Application settings** link in the main window:

The screenshot shows the Azure Functions Overview page for the 'NotificationFunctionAppBook' function app. The left sidebar lists 'Function Apps' and the specific app 'NotificationFunctionApp...'. The main area has tabs for 'Overview' and 'Platform features'. Under 'Overview', there are buttons for 'Stop', 'Swap', 'Restart', and links for 'Get publish profile' and 'Reset publish profile'. The status is shown as 'Running'. Below the status are sections for 'Subscription' (RiteshSubscription), 'Resource group' (VaultMonitoring), 'Subscription ID' (redacted), and 'Location' (West Europe). A section titled 'Configured features' contains three items: 'Function app settings' (disabled), 'Application settings' (selected and highlighted with a red box), and 'Application Insights'.

7. Navigate to the **Application settings** section, as shown in the previous screenshot, and add a few entries by clicking on **+ Add new setting** for each entry.

The screenshot shows the 'Application settings' blade. At the top, a message states: 'Application Settings are encrypted at rest and transmitted over an encrypted connection.' Below are two buttons: 'Hide Values' and 'Show Values'. The main area is a table with columns 'APP SETTING NAME' and 'VALUE'. The table lists several settings with their values set to 'Hidden value. Click to edit.' A purple vertical bar highlights the first few rows. At the bottom of the table is a blue link '+ Add new setting'.

APP SETTING NAME	VALUE
APPINSIGHTS_INSTRUMENTATIONKEY	Hidden value. Click to edit.
AzureWebJobsStorage	Hidden value. Click to edit.
FUNCTIONS_EXTENSION_VERSION	Hidden value. Click to edit.
FUNCTIONS_WORKER_RUNTIME	Hidden value. Click to edit.
WEBSITE_CONTENTAZUREFILECONNECTIONSTRING	Hidden value. Click to edit.
WEBSITE_CONTENTSHARE	Hidden value. Click to edit.
WEBSITE_NODE_DEFAULT_VERSION	Hidden value. Click to edit.
adminPhone	Hidden value. Click to edit.
servicePhone	Hidden value. Click to edit.
serviceEmail	Hidden value. Click to edit.
bccStakeholdersEmail	Hidden value. Click to edit.
toStakeholdersEmail	Hidden value. Click to edit.
SendGridAPIKeyAsAppSetting	Hidden value. Click to edit.
TwilioAccountSid	Hidden value. Click to edit.
TwilioAuthToken	Hidden value. Click to edit.

Note that the entries are in the form of key-value pairs, and the values should be actual real-time values. Both `adminPhone` and `servicePhone` should already be configured on the Twilio website. `servicePhone` is the phone number generated by Twilio that is used for sending SMS messages, and `adminPhone` is the phone number of the administrator to whom the SMS should be sent.

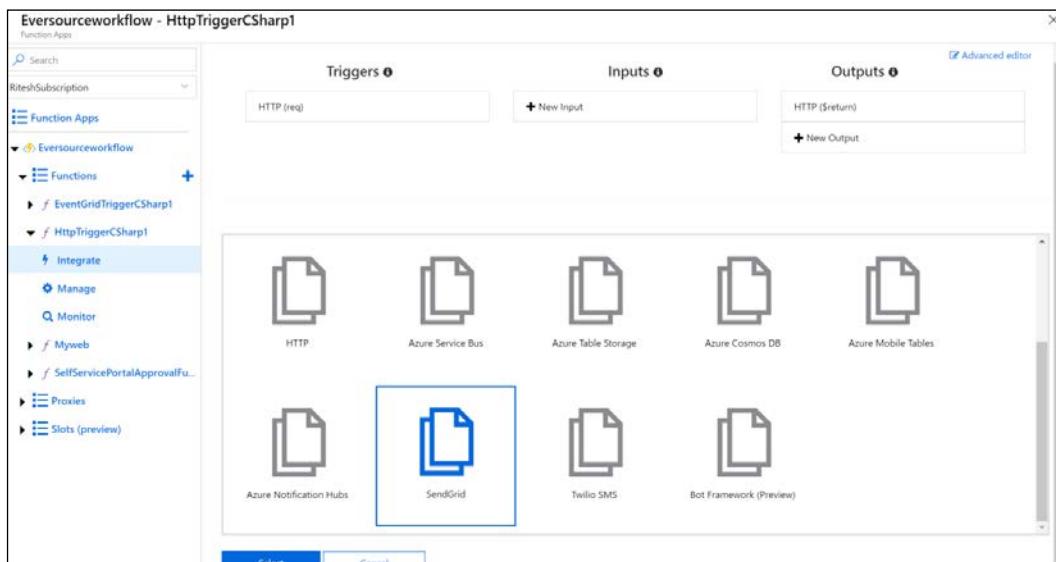
Also note that Twilio expects the destination phone number to be in a particular format depending on the country (for India, the format is +91 xxxxx xxxx). Note the spaces and country code in the number.

We also need to add the keys for both SendGrid and Twilio within the application settings. These settings are mentioned in the following list. Readers may already have these values handy because of activities performed in the earlier steps:

- The value of `SendGridAPIKeyAsAppSetting` is the key for SendGrid.
 - `TwilioAccountSid` is the system identifier for the Twilio account. This value was already copied and stored in a transient place in an earlier step.
 - `TwilioAuthToken` is the token for the Twilio account. This value was already copied and stored in a temporary place in an earlier step.
8. Save the settings by clicking on the **Save** button in the top menu:
 9. Click on the **Integrate** link in the left-hand menu just below the name of the function, and click on **+ New Output**. This is to add an output for the SendGrid service:



10. Next, select **SendGrid** – it might ask you to install the SendGrid extension.
Install the extension, which will take a couple of minutes:



11. After installing the extension, the output configuration form appears. The important configuration items in this form are **Message parameter name** and **SendGridAPIKeyAppSetting**. Leave the default value for **Message parameter name** and click on the drop-down list to select **SendGridAPIKeyAsAppSetting** as the API app setting key. This was already configured in a previous step within the app settings configuration. The form should be configured, as shown in the following screenshot, and then click on the **Save** button:

The screenshot shows the 'SendGrid output' configuration dialog. It contains the following fields:

- Message parameter name:** A text input field containing "message".
- SendGrid API Key App Setting:** A dropdown menu set to "SendGridAPIKeyAsAppSetting".
- From address:** An input field.
- To address:** An input field.
- Message subject:** An input field.
- Message Text:** An input field.

At the bottom are two buttons: **Save** (highlighted in blue) and **Cancel**.

12. Click on **+ New Output** again; this is to add an output for the Twilio service.
13. Then, select **Twilio SMS**. It might ask you to install the Twilio SMS extension. Install the extension, which will take a couple of minutes.

14. After installing the extension, the output configuration form appears. The important configuration items in this form are **Message parameter name**, **Account SID setting**, and **Auth Token setting**. Change the default value for the **Message parameter name** to `sms`. This is done because the message parameter is already used for the SendGrid service parameter. Ensure that the value of **Account SID setting** is `TwilioAccountSid` and that the value of the **Auth Token setting** is `TwilioAuthToken`. These values were already configured in a previous step within the app settings configuration. The form should be configured, as shown in the following screenshot, and then click on **Save**:

The screenshot shows the configuration interface for the Twilio SMS output extension. At the top, there is a success message: "Extension Installation Succeeded". Below this, the configuration fields are arranged in two columns:

Message parameter name ⓘ	Account SID setting ⓘ
<input type="text" value="sms"/>	<input type="text" value="TwilioAccountSid"/>
<input type="checkbox"/> Use function return value	From number ⓘ
Auth Token setting ⓘ	<input type="text" value="From number"/>
<input type="text" value="TwilioAuthToken"/>	
Message text ⓘ	
<input type="text" value="Message text"/>	
Save Cancel	

Step 11

In this step, we will be creating a new Logic App workflow. We have authored an Azure Automation runbook that queries all the secrets in all key vaults and publishes an event in case it finds any of them expiring within a month. The Logic Apps workflow acts as a subscriber to these events:

1. The first step within the **Logic App** menu is to create a Logic Apps workflow:

The screenshot shows the Azure Marketplace interface. A search bar at the top contains the text 'logic app'. Below the search bar are three filter dropdowns: 'Pricing' set to 'All', 'Operating System' set to 'All', and 'Publisher' set to 'All'. The main area is titled 'Results' and displays a list of logic app-related items:

NAME	PUBLISHER	CATEGORY
Logic App	Microsoft	
Logic Apps B2B	Microsoft	Management Tools
Logic Apps Custom Connector	Microsoft	
Logic Apps Management (Preview)	Microsoft	Management Tools
Sumo Logic for Azure Web Apps	Sumo Logic	Compute
Sumo Logic for Azure Audit	Sumo Logic	Compute
RadiantOne on Linux	Radiant Logic, Inc.	Compute
RadiantOne on Windows Server	Radiant Logic, Inc.	Compute
ASP.NET Starter Web App	Microsoft	Web

Below the list, there is a section titled 'Related to your search' with two items: 'App Service Plan' (Microsoft) and 'Scheduler' (Microsoft). To the right of the search results, there is a sidebar titled 'Logic App' with descriptive text and links for 'Compose SaaS easily', 'Easy to use design tools', 'Extensibility baked in', and 'Real integration horsepower'. A 'Save for later' button is also present in this sidebar. At the bottom right of the sidebar, there is a preview window showing a logic app workflow with several steps and a 'Create' button.

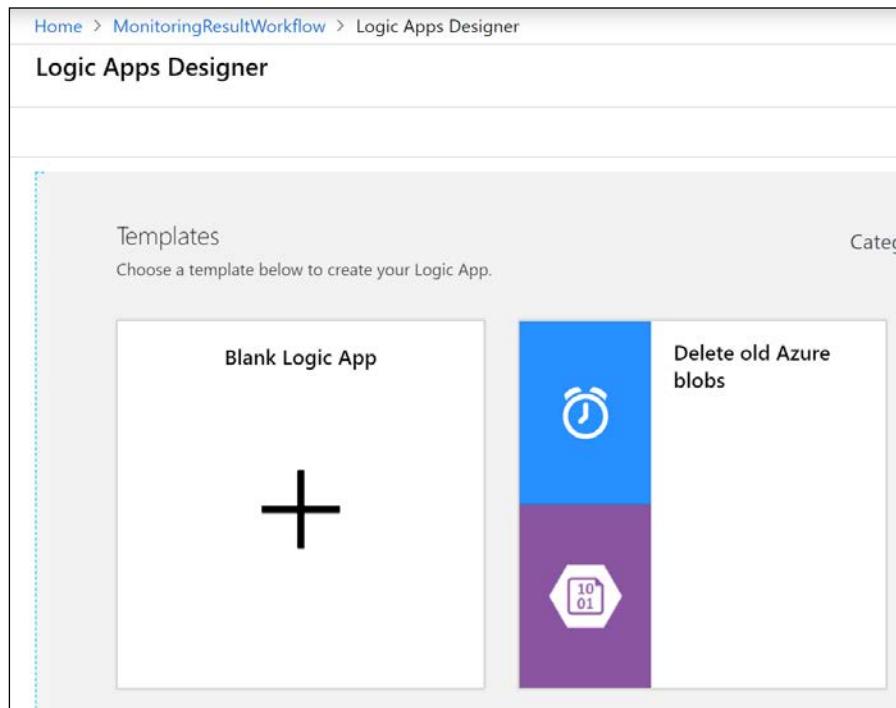
2. Fill up the resultant form after clicking on the **Create** button. We are provisioning the logic app in the same resource group as the other resources for this solution:

The screenshot shows the 'Logic App' creation form in the Azure portal. The form is titled 'Logic App' and includes the following fields:

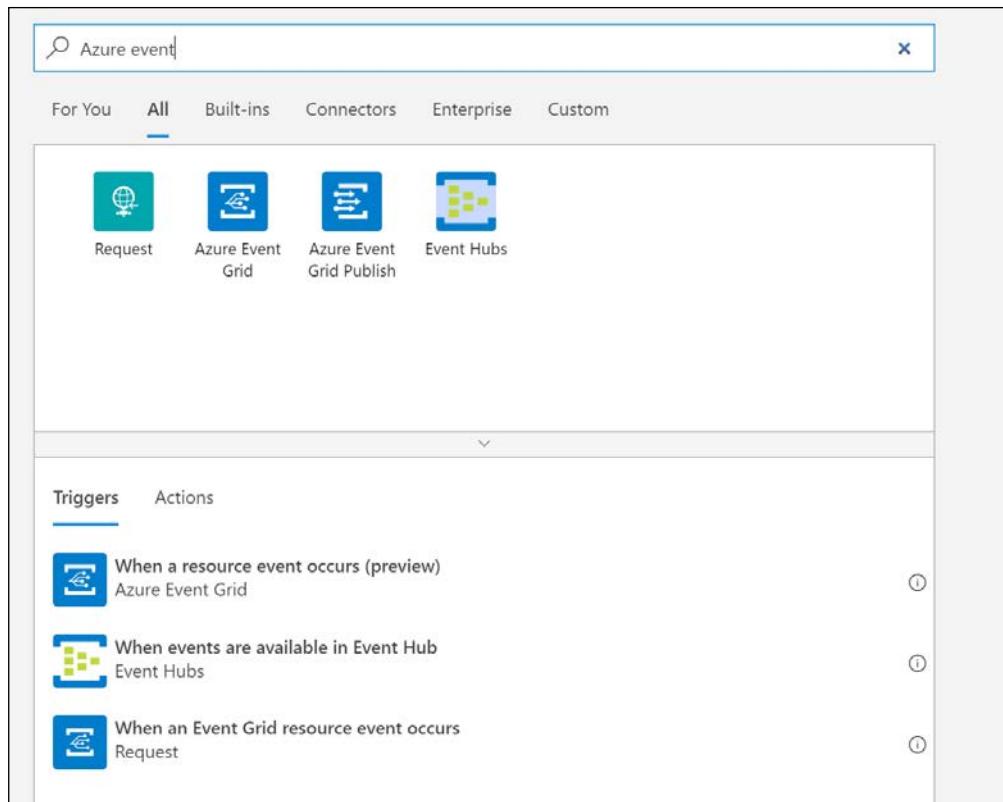
- Name:** MonitoringResultWorkflow (highlighted with a purple border)
- Subscription:** RiteshSubscription
- Resource group:** Use existing (VaultMonitoring selected)
- Location:** West Europe
- Log Analytics:** Off

A note at the bottom states: "You can add triggers and actions to your Logic App after creation." At the bottom right are 'Create' and 'Automation options' buttons.

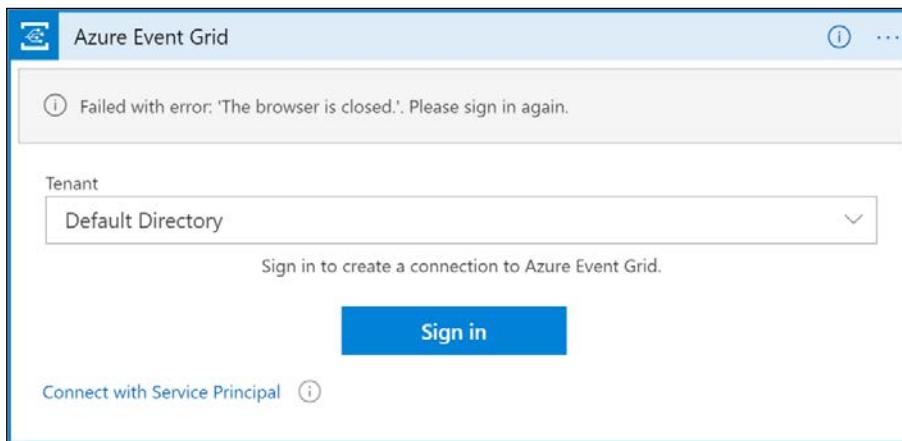
3. After the logic app is provisioned, it opens the designer window. Select **Blank Logic App** from the **Templates** section, as demonstrated in the following screenshot:



4. In the resultant window, add a trigger that can subscribe to Event Grid events. Logic Apps provides a trigger for Event Grid, and you can search for this to see whether it's available:

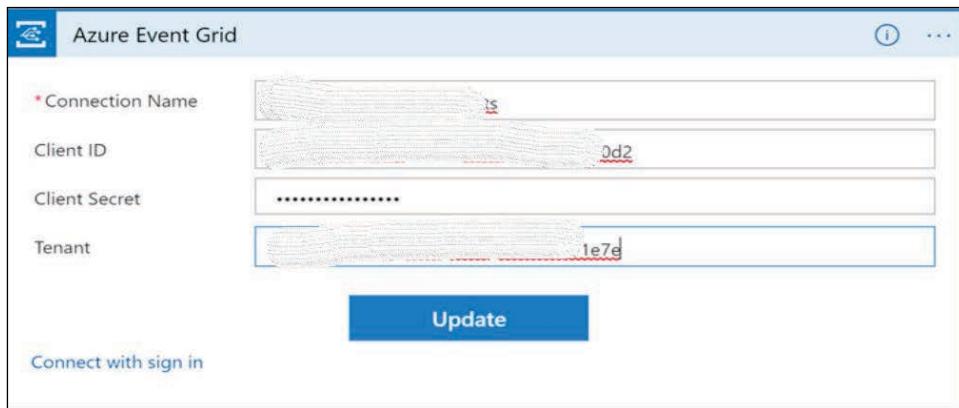


5. Next, select the **When a resource event occurs (preview)** trigger:

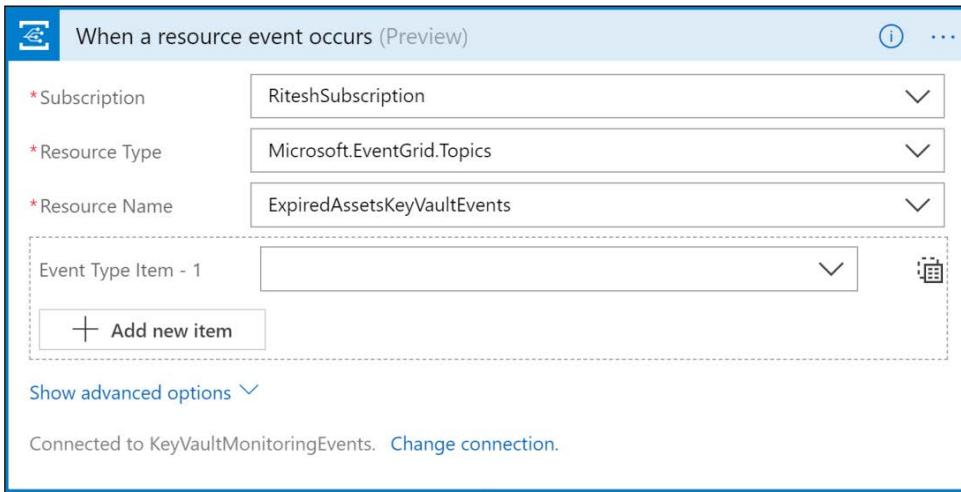


6. In the resultant window, select **Connect with Service Principal**.

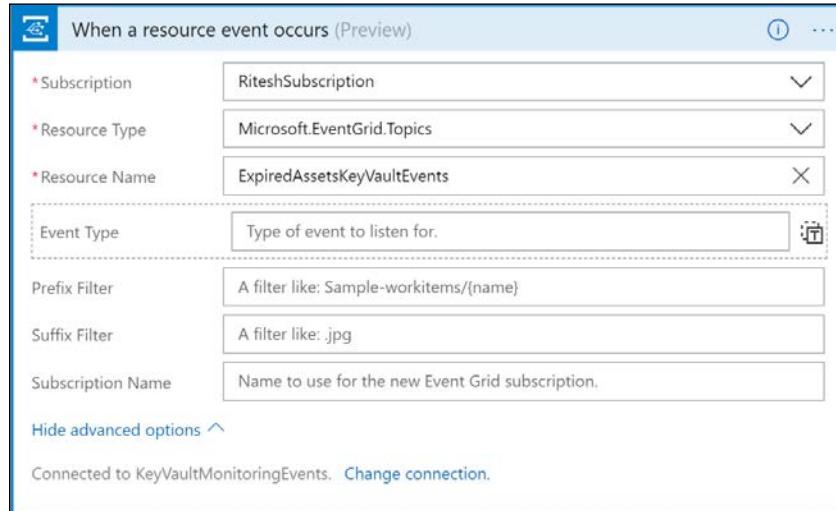
Provide the service principal details, including the application ID (**Client ID**), tenant ID, and password. This trigger does not accept a service principal that authenticates with the certificate – it accepts a service principal only with a password. Create a new service principal at this stage that authenticates with a password (the steps for creating a service principal based on password authentication was covered earlier in *Step 2*) and use the details of the newly-created application principal for Azure Event Grid configuration, as follows:



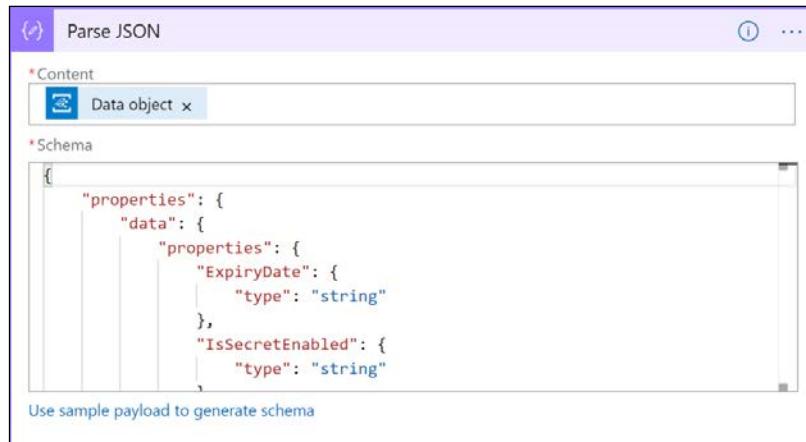
7. Select the subscription. Based on scope of service principal, this will get auto-filled. Select **Microsoft.EventGrid.Topics** as the **Resource Type** and set the name of the custom topic as **ExpiredAssetsKeyVaultEvents**:



8. The previous step will create a connector, and the connection information can be changed by clicking on **Change connection**.
9. The final configuration of Event Grid trigger should be similar to the following screenshot:



10. Add a new **Parse JSON** activity after the Event Grid trigger; this activity needs the JSON schema. Generally, the schema is not available, but this activity helps generate the schema if a valid JSON is provided to it:



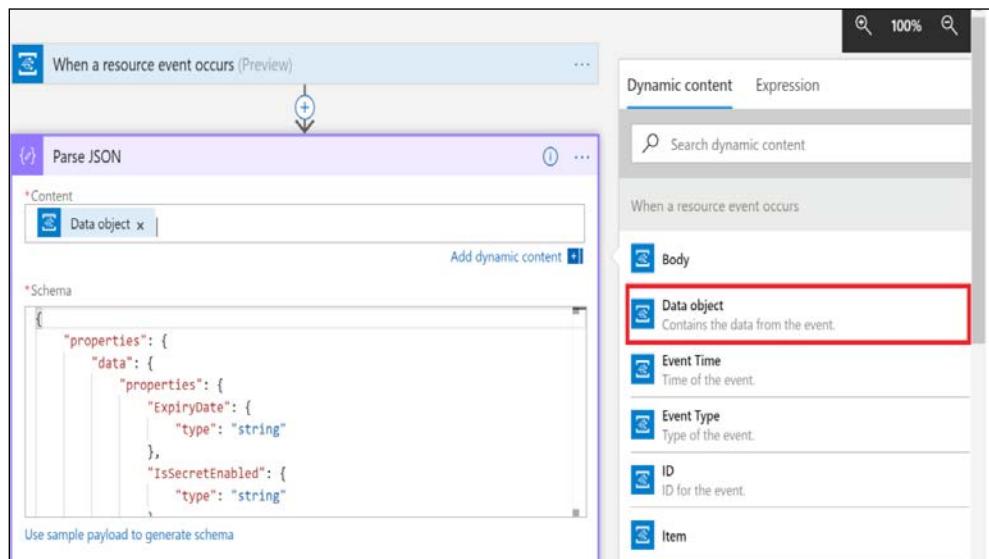
11. Click on **Use sample payload to generate schema** and provide the following data:

```
{  
    "ExpiryDate": "",  
    "SecretName": "",  
    "VaultName": "",  
    "SecretCreationDate": "",  
    "IsSecretEnabled": "",  
    "SecretId": ""  
}
```

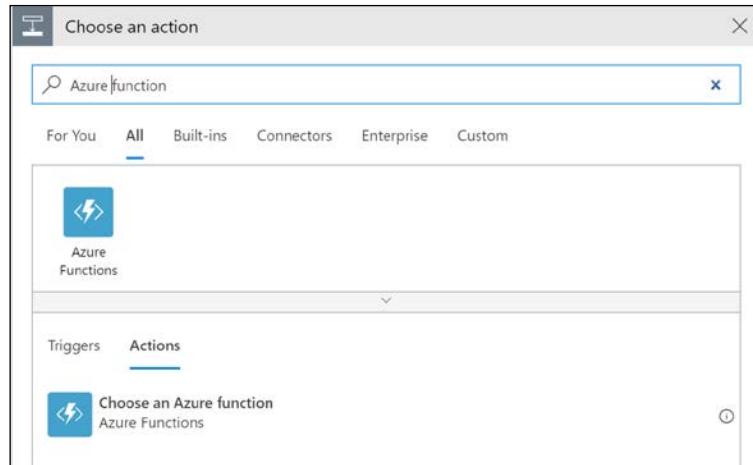
A question might arise here regarding the sample payload. How do you know at this stage what the payload is that's generated by the Event Grid publisher? The answer to this lies in the fact that this sample payload is exactly the same as is used in the data element in the Azure Automation runbook; you can take a look at that code snippet again:

```
data = @{  
    "ExpiryDate" = $certificate.Expires  
    "CertificateName" = $certificate.Name.ToString()  
    "VaultName" = $certificate.VaultName.ToString()  
    "CertificateCreationDate" = $certificate.Created.ToString()  
    "IsCertificateEnabled" = $certificate.Enabled.ToString()  
    "CertificateId" = $certificate.Id.ToString()  
}
```

12. The **Content** textbox should contain dynamic content coming out from the previous trigger, as demonstrated in the following screenshot:



13. Add another **Azure Functions** action after **Parse JSON**, and then select **Choose an Azure function**. Select the Azure function apps called `NotificationFunctionAppBook` and `SMSAndEmailFunction`, which were created earlier:

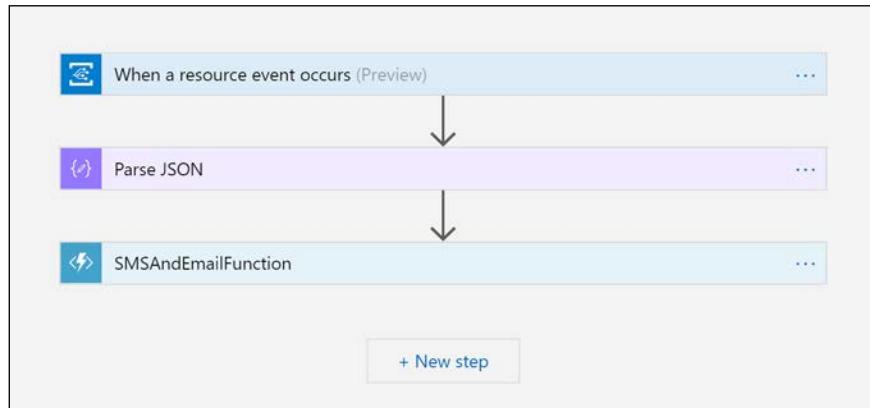


14. Click on the **Request Body** text area and fill it with the following code listing. This is done to convert the data into JSON before sending it to the Azure function:

```
{
  "alldata" :
}
```

15. Place the cursor after : in the preceding code and click on **Add dynamic content | Body from the previous activity**:

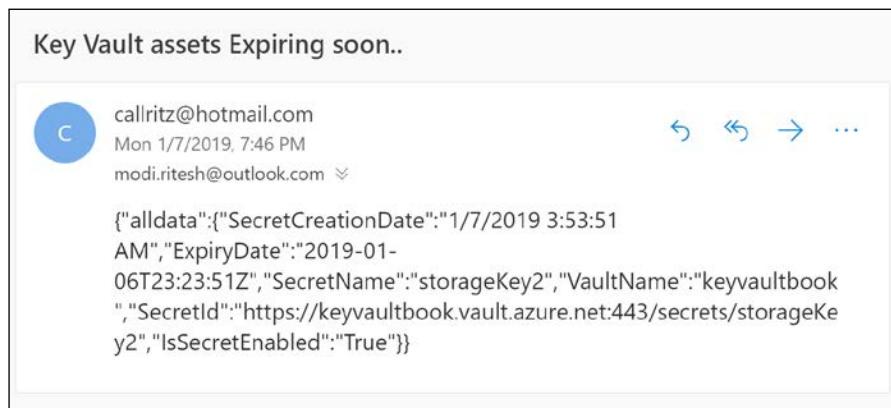
16. Save the entire logic app; the entire logic app should look as follows:



Testing

Upload some secrets and certificates that have expiry dates to Azure Key Vault and execute the Azure Automation runbook. The runbook is scheduled to run on a schedule. Additionally, the runbook will publish events to Event Grid. The logic app should be enabled, and it will pick the event and finally invoke the Azure function to send email and SMS notifications.

The email should look as follows:



Summary

This was a big chapter – this chapter introduced Event Grid by providing a couple of end-to-end solutions using storage resources and custom topics. It also introduced Azure Logic Apps as automated serverless workflows. The chapter focused heavily on creating an architecture that integrated multiple Azure services to create an end-to-end solution. The services used in the solution were Azure Automation, Azure Logic Apps, Azure Event Grids, Azure Functions, SendGrid, and Twilio. These services were implemented through the Azure portal and PowerShell using service principals as service accounts. It also showed a number of ways in which to create service principals with password and certificate authentication.

The next chapter is an important chapter from the Azure cost management perspective. It is quite easy to spin new resources on Azure without knowing the actual cost of running them. In the next chapter, we will provide details about cost management.

8

Cost Management

The primary reason that corporations are moving to the cloud is to save costs. There is no upfront cost for having an Azure subscription. Azure provides a **pay-as-you-go** payment mechanism, meaning that payment is based on consumption; Azure measures usage and provides monthly invoices based on your consumption. There is no upper limit for how much you can consume – Azure provides unlimited resources, and anybody with access to Azure can create as many resources as they want. Of course, in such circumstances, it is important for companies to keep a close watch on their Azure consumption. Although they can create policies to set organizational standards and conventions, there is also a need to have Azure billing and consumption information readily available. Moreover, companies should look to employ best practices for consuming Azure resources such that the returns are maximized. For this, architects need to be fully aware of Azure resources and features, their corresponding costs, and to perform cost-benefit comparisons.

In this chapter, we will cover the following topics:

- Understanding Azure billing
- Invoicing
- Enterprise Agreement customers
- Usage and quotas
- Resource providers
- Usage and billing APIs
- The Azure pricing models and calculator
- Best practices

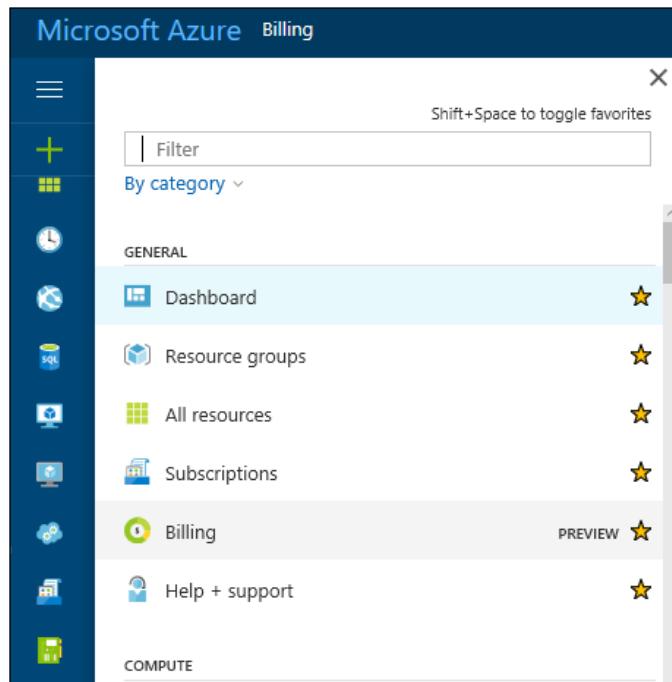
Understanding billing

Azure is a service utility that offers the following benefits:

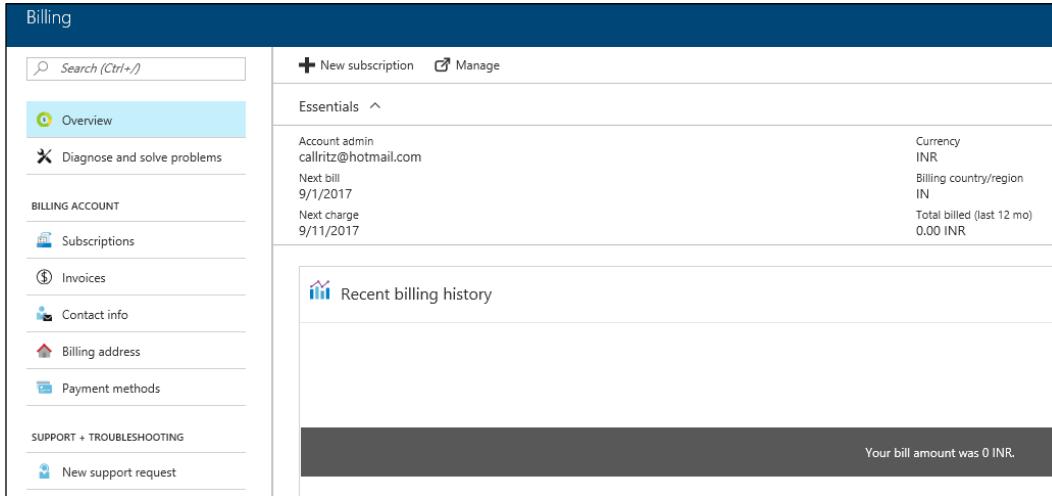
- No upfront costs
- No termination fees
- Billing on a per-minute basis
- Payment based on consumption

In such circumstances, it is very difficult to estimate the cost of consuming Azure resources. Every resource in Azure has its own cost model and charge based on storage, usage, and timespan. It is very important for the management, administration, and finance departments of a company to keep track of usage and costs. Azure provides the necessary usage and billing reports, such that an organization's management and administrators can generate cost and usage reports based on all sorts of criteria.

The Azure portal provides detailed billing and usage information through the **Billing** feature, which can be accessed from the master navigation blade:

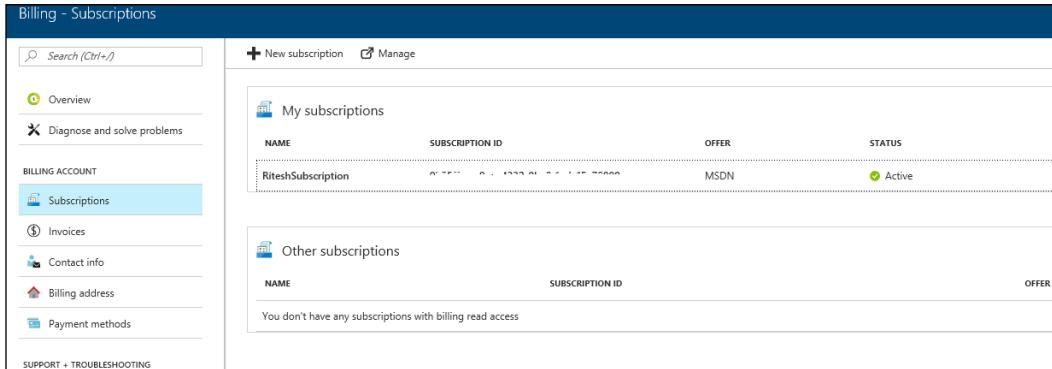


It provides a sub-menu for generating reports on both costs and billing:



The screenshot shows the 'Billing' blade with the 'Overview' section selected. The left sidebar includes links for 'Search (Ctrl+)', 'Overview', 'Diagnose and solve problems', 'BILLING ACCOUNT' (Subscriptions, Invoices, Contact info, Billing address, Payment methods), and 'SUPPORT + TROUBLESHOOTING' (New support request). The main content area displays 'Essentials' information: Account admin (callnitz@hotmail.com), Next bill (9/1/2017), Next charge (9/11/2017), Currency (INR), Billing country/region (IN), and Total billed (last 12 mo) (0.00 INR). A 'Recent billing history' section is present, and a note at the bottom states 'Your bill amount was 0 INR.'

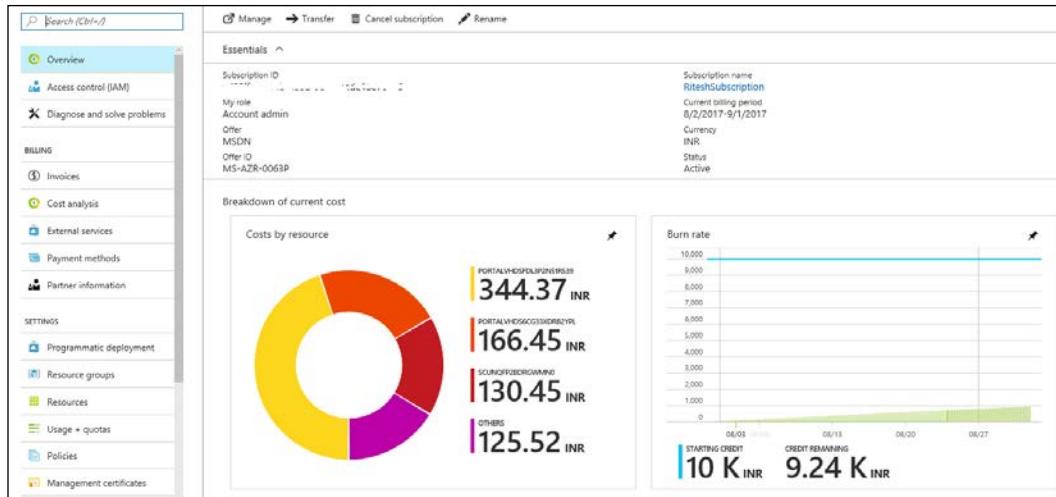
Clicking on the **Subscriptions** menu on this blade provides a list of all the subscriptions that the user has access to for generating reports:



The screenshot shows the 'Billing - Subscriptions' blade with the 'Subscriptions' section selected. The left sidebar is identical to the previous screenshot. The main content area shows 'My subscriptions' with one entry: RiteshSubscription, Offer: MSDN, Status: Active. Below it is a 'Other subscriptions' section which states 'You don't have any subscriptions with billing read access'.

Cost Management

Clicking on the subscription name in this list will provide a dashboard through which the complete billing, invoice, usage, consumption, policies, resource groups, and resources information can be found. The chart in this **Overview** section provides a cost percentage by resource and also a burn rate:



Clicking on the chart shows details of costs based on each resource. Here, there are multiple Storage accounts provisioned on Azure, and the cost for each is displayed. Using this screen, many types of report can be generated by providing different criteria, such as any combination of the following:

- Resource types
- Resource group
- Timespan
- Tags

Tags are particularly interesting. Queries based on tags such as department, project, owner, cost center, or any other name-value pair can be used to display cost information. You can also download the cost report as a CSV file using the **Download** button:

Costs by resource

Subscription: Rhoes0subscription | Resource type: All resource types | Resource group: All resource groups

Timespan: Current period | Tag: All tags

Total cost: **766.80** INR

NAME	TYPE	RESOURCE GROUP	COST (INR)	TAGS
portalvhddfd13p2n51539	Storage account (classic)	Default-Storage-WestUS	344.37	---
portalvhddfcg33xdth2ypl	Storage account (classic)	Default-Storage-EastUS	166.45	---
scumpip2ldrgwmr0	Storage account	abra	130.45	---
scumpip2ldrgwte	Storage account	abra	64.74	---
portalvhdsn0bxtwdj38ym	Storage account (classic)	Default-Storage-EastAsia	60.79	---

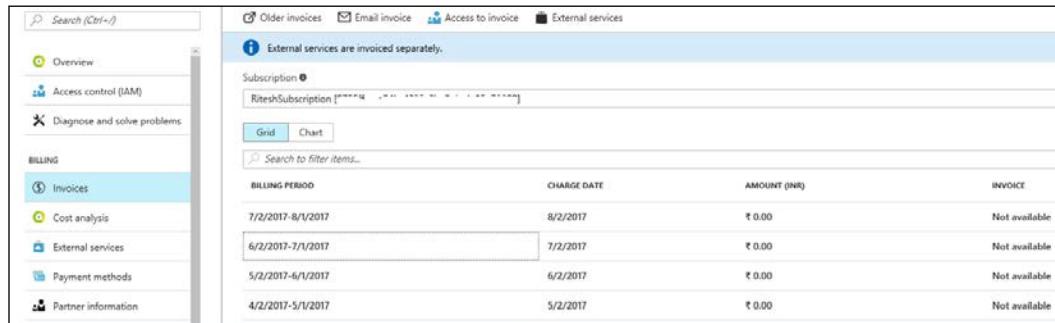
Clicking on an individual resource provides the daily cost consumption data for the resource:

DATE	AMOUNT (INR)
8/2/2017	13.73
8/3/2017	13.73
8/4/2017	13.73
8/5/2017	13.73
8/6/2017	13.73
8/7/2017	13.73
8/8/2017	13.73
8/9/2017	13.73
8/10/2017	13.73
8/11/2017	13.73
8/12/2017	13.73
8/13/2017	13.73
8/14/2017	13.73
8/15/2017	13.73
8/16/2017	13.73

Invoicing

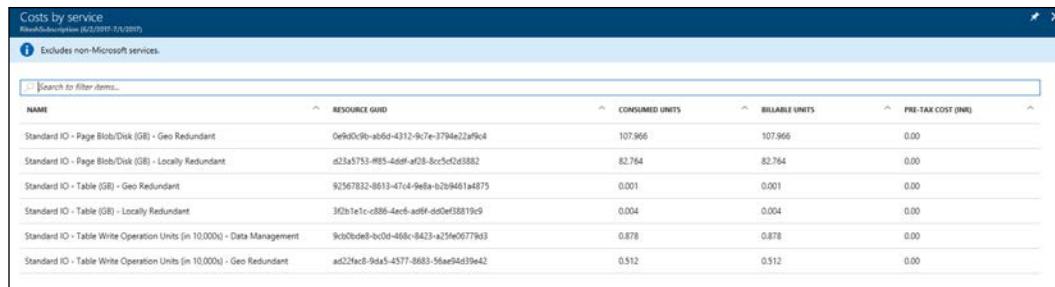
The Azure Billing feature also provides information about monthly invoices.

Clicking on the **Invoices** menu provides a list of all invoices generated:



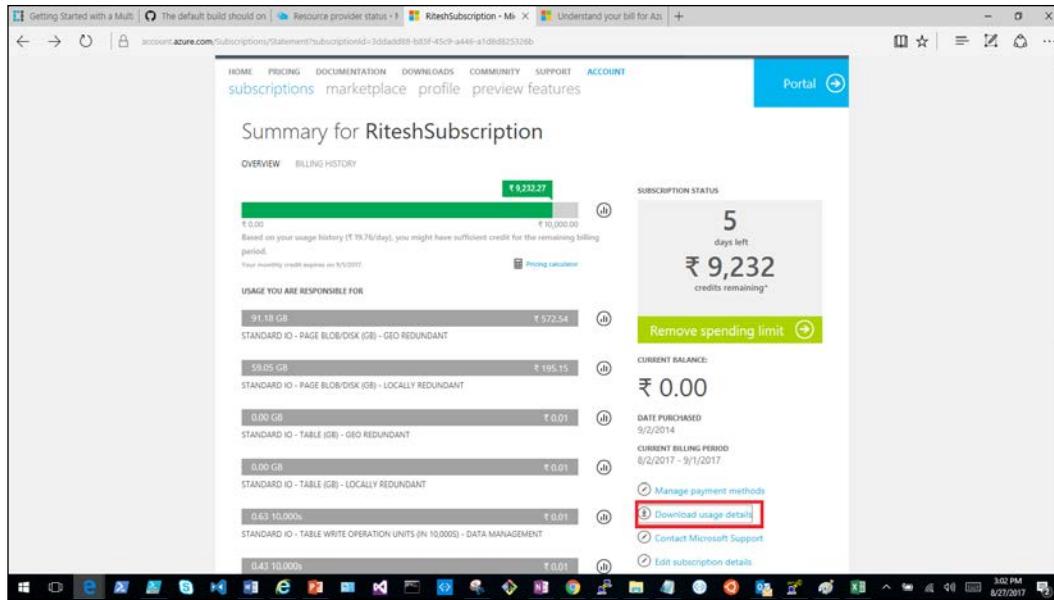
BILLING PERIOD	CHARGE DATE	AMOUNT (INR)	INVOICE
7/2/2017-8/1/2017	8/2/2017	₹ 0.00	Not available
6/2/2017-7/1/2017	7/2/2017	₹ 0.00	Not available
5/2/2017-6/1/2017	6/2/2017	₹ 0.00	Not available
4/2/2017-5/1/2017	5/2/2017	₹ 0.00	Not available

Clicking on any of the invoices provides details about that invoice:



NAME	RESOURCE GUID	CONSUMED UNITS	BILLABLE UNITS	PRE-TAX COST (INR)
Standard IO - Page Blob/Disk (GB) - Geo Redundant	0e9d0c9b-ab5d-4312-9c7e-3794e22a9c4	107.966	107.966	0.00
Standard IO - Page Blob/Disk (GB) - Locally Redundant	d23a5753-#85-4d6f-a728-8cc5cf2e3382	82.764	82.764	0.00
Standard IO - Table (GB) - Geo Redundant	92567832-#613-474-9eb-a299461aa875	0.001	0.001	0.00
Standard IO - Table (GB) - Locally Redundant	3fb1e1c-c886-4ec-adsf-d00ef8819c9	0.004	0.004	0.00
Standard IO - Table Write Operation Units (in 10,000s) - Data Management	9cb6bde8-bc0d-46bc-8423-a25fe0677963	0.878	0.878	0.00
Standard IO - Table Write Operation Units (in 10,000s) - Geo Redundant	ad32fec8-9da5-4577-8683-56ae94d39e42	0.512	0.512	0.00

There is also an alternative way to download invoice details. The invoice details are available by logging into <https://account.azure.com> and downloading the invoice details:



Enterprise Agreement customers

Enterprise customers that have an Enterprise Agreement can utilize <https://ea.azure.com> to download their usage and billing reports. Also, a new Power BI content pack was released recently that can be utilized to view Azure usage and costs through reports and a dashboard in Power BI.

More information about Enterprise Agreements usage and costs is available at <https://azure.microsoft.com/en-us/blog/new-power-bi-content-pack-for-azure-enterprise-users/>.

Usage and quotas

Each subscription has a limited quota for each resource type. For example, there could be a maximum of 60 public IP addresses provisioned with an MSDN Microsoft account. Similarly, all resources have a maximum default limit for each resource type. These resource type numbers for a subscription can be increased by contacting Azure support or clicking on the **Request Increase** button.

The usage and quota information is available from the **Usage + quotas** sub-menu of the **Subscription** menu:

QUOTA	PROVIDER	LOCATION	USAGE
Storage Accounts (Classic)	Microsoft.ClassicStorage	Global	3 %

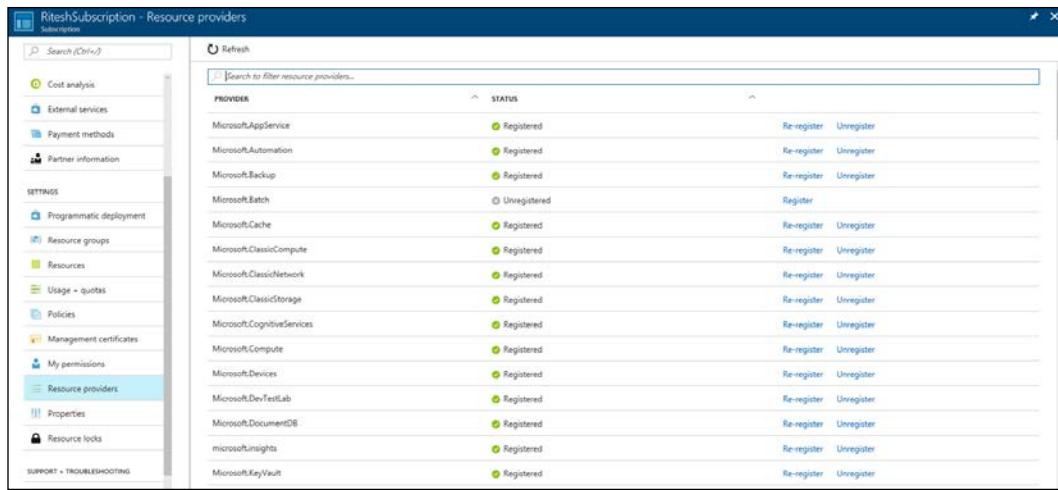
This blade shows all resource types provisioned in a subscription, along with their locations and counts. Here, the quota for classic Storage accounts is 100, and, currently, three classic Storage accounts have been consumed.

You can filter by location, provider, usage, and quota. Custom reports based on these filter criteria can be generated.

Resource providers

Resources are based on resource types and are available from resource providers. There are numerous providers available in Azure, which provide the resource types needed by users to create their instances. For example, the **Microsoft.Compute** resource provider provides virtual machine resource types. With virtual machine resource types, instances of virtual machines can be created.

Resource providers are required to be registered with Azure subscriptions. Resource types will not be available in a subscription if resource providers are not registered. To get a list of the providers that are available, or to see the ones that are registered and the ones that aren't, or even to register non-registered providers, this dashboard can be used:



The screenshot shows the 'Resource providers' section of the Azure portal. The left sidebar includes links for Cost analysis, External services, Payment methods, Partner information, Programmatic deployment, Resource groups, Resources, Usage + quotas, Policies, Management certificates, My permissions, and Resource providers (which is selected). The main area displays a table with columns for PROVIDER and STATUS. Most providers are listed as Registered, while Microsoft.Batch is Unregistered. Each row has 'Re-register' and 'Unregister' buttons.

PROVIDER	STATUS		
Microsoft.AppService	Registered	Re-register	Unregister
Microsoft.Automation	Registered	Re-register	Unregister
Microsoft.Backup	Registered	Re-register	Unregister
Microsoft.Batch	Unregistered	Register	
Microsoft.Cache	Registered	Re-register	Unregister
Microsoft.ClassicCompute	Registered	Re-register	Unregister
Microsoft.ClassicNetwork	Registered	Re-register	Unregister
Microsoft.ClassicStorage	Registered	Re-register	Unregister
Microsoft.CognitiveServices	Registered	Re-register	Unregister
Microsoft.Compute	Registered	Re-register	Unregister
Microsoft.Devices	Registered	Re-register	Unregister
Microsoft.DevTestLab	Registered	Re-register	Unregister
Microsoft.DocumentDB	Registered	Re-register	Unregister
microsoft.insights	Registered	Re-register	Unregister
Microsoft.KeyVault	Registered	Re-register	Unregister

The usage and billing APIs

Although the portal is a great way to find usage, billing, and invoice information manually, Azure also provides the following:

- **The Invoice Download API:** Use this API to download invoices.
- **The Resource Usage API:** Use this API to get estimated Azure resource consumption data.
- **The RateCard API:** Use this API to get a list of available Azure resources and estimated pricing information for each of them.

These APIs can be used to programmatically retrieve details and create customized dashboards and reports. Any programming or scripting language can use these APIs and create a complete billing solution.

Azure pricing models

Azure has multiple pricing models. It has a model for every type of customer. From free accounts for students and pay-as-you-go accounts for developers to enterprise agreements and cloud solution provider partner models. Apart from these account types, there are add-on pricing and discounts, such as reserved virtual machine instances and the Azure Hybrid Benefit.

Azure Hybrid Benefit

When a virtual machine is provisioned on Azure, there are two types of cost involved. These two costs are the resource cost for running the virtual machine and the operating system license cost. Although Enterprise Agreement customers get some discounts compared to other accounts, Azure provides another offer for them: the Azure Hybrid Benefit. In this scheme, existing Enterprise Agreement customers can use their on-premises operating system licenses to create their virtual machines on Azure, and Azure will not charge the cost of the license. The cost savings can be as high as 40 percent of the original cost of using this scheme. Enterprise Agreement customers should also have software assurance to enjoy this benefit, which is applicable for both Windows Standard and Datacenter editions. Each two-processor license or each set of 16-core licenses is entitled to two instances of up to eight cores, or one instance of up to 16 cores. The Azure Hybrid Benefit for Standard edition licenses can only be used in one instance, whether on-premises or in Azure. The Datacenter edition allows simultaneous usage both on-premises and in Azure.

Azure reserved virtual machine instances

Customers can reserve a fixed number of virtual machines in advance, for one year to three years for both the Windows and Linux operating systems. Azure provides up to a 72 percent discount on these virtual machines based on a pay-as-you-go pricing model. Although there is an upfront commitment, there is no obligation to use the instances. These reserved instances can be canceled at any point in time. This offering can even be clubbed with the Azure Hybrid Benefit scheme to further reduce the cost of these virtual machines.

Pay-as-you-go accounts

These are general Azure accounts and are billed monthly to customers. Customers do not commit any usage and are free to use any resource based on their needs. Resource costs are calculated based on usage and uptime. However, each resource has its own cost model. There is also no upfront cost associated with these accounts. Generally, there are no discounts available in this scheme.

Enterprise Agreements

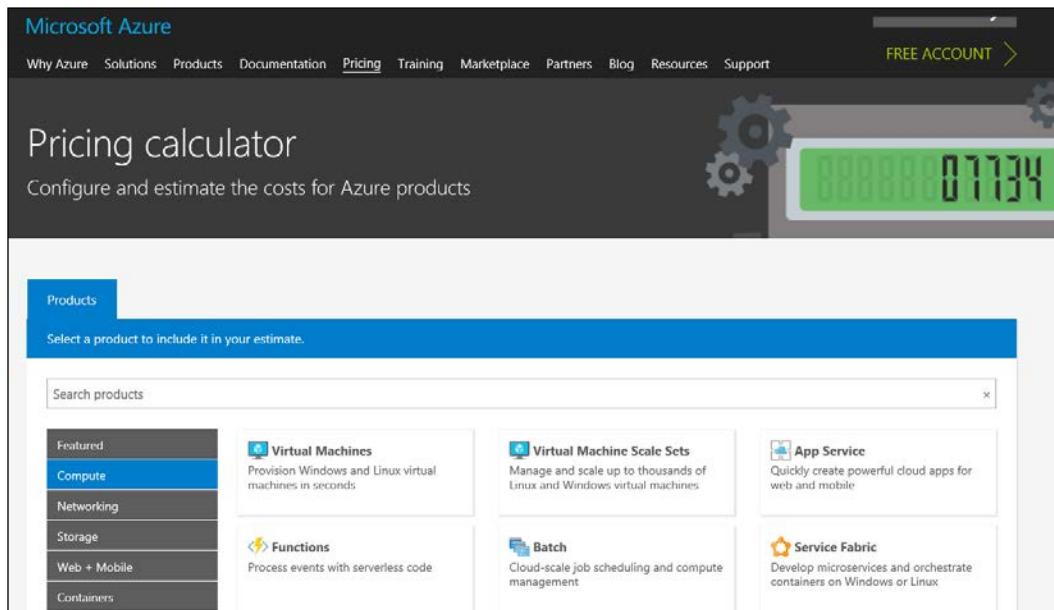
Customers who already have agreements with Microsoft can add their Azure tenants as part of Enterprise Agreements. Customers can enjoy great discounts if they are part of Enterprise Agreements. Customers just need to make an upfront annual monetary commitment and they can be added to this scheme. Customers are free to consume as they please. Please refer to <https://azure.microsoft.com/en-in/pricing/enterprise-agreement/> for more information.

The Cloud Solution Provider model

The **Cloud Solution Provider (CSP)** model is a model for Microsoft partners. CSP enables partners to have end-to-end ownership of the customer life cycle and relationship for Microsoft Azure. Partners can deploy their solutions to the cloud and charge customers using this scheme. Please refer to <https://azure.microsoft.com/en-in/offers/ms-azr-0145p/> for more information.

The Azure pricing calculator

Azure provides a cost calculator for users and customers to estimate their cost and usage. This calculator is available at <https://azure.microsoft.com/en-in/pricing/calculator/>:



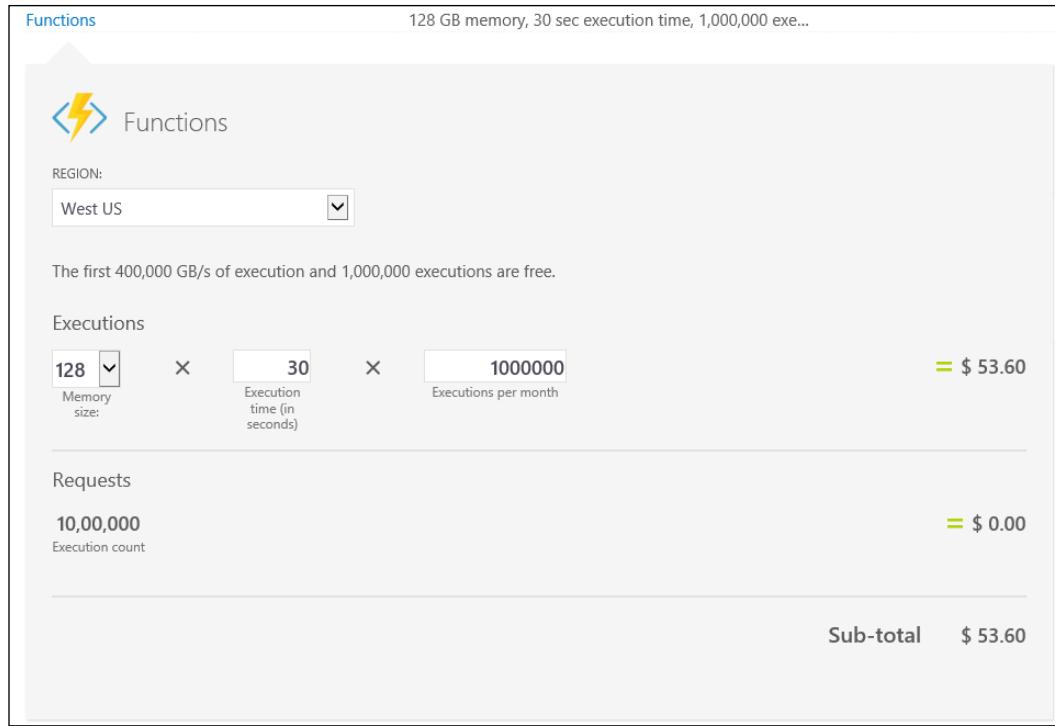
Cost Management

Users can select multiple resources from the left menu and they will be added to the calculator. In the following example, a virtual machine is added. Further configuration with regard to virtual machine region, operating system, type, tier, instance size, number of hours, and count can be done:

The screenshot shows the 'Your Estimate' section of the Azure Cost Management tool. At the top, it says 'Virtual Machines' and '1: D1: 1 cores, 3.5 GB RAM, 50 GB disk'. Below this is a 'Virtual Machines' configuration panel. It includes fields for REGION (West US), OPERATING SYSTEM (Windows), TYPE (OS Only), TIER (Standard), and INSTANCE (D1: 1 Core(s), 3.5 GB RAM, 50 GB Disk, \$ 0.140/hour). There is also an option to ADD MANAGED DISKS. At the bottom, there is a summary: 1 Virtual machines multiplied by 744 Hours equals a total cost of \$ 104.16.

Virtual machines	Hours	Total Cost
1	744	\$ 104.16

Similarly, the cost for Azure Functions in terms of virtual machine memory size, execution time, and execution per second is shown next:



Azure provides different levels and plans of support, which are as follows:

- **Default support:** Free
- **Developer support:** \$29 per month
- **Standard support:** \$300 per month
- **Professional direct:** \$1,000 per month

Finally, the overall estimated cost is displayed:

The screenshot shows the Azure Cost Management calculator interface. It displays two main sections: 'Support' and 'Programs and Offers'. Under 'Support', the 'Developer' SKU is selected at \$29.00. Under 'Programs and Offers', the 'Microsoft Online Services Program (MOSP)' is selected. A checkbox for 'SHOW DEV/TEST PRICING' is unchecked. The total 'Estimated monthly cost' is \$186.76, displayed in US Dollar (\$). A blue 'Export' button is visible.

It is important that architects understand how each Azure feature is used in the overall architecture and solution. The accuracy of the Azure calculator depends on what resources are selected and what their configuration is. Any misrepresentation would lead to bias and incorrect estimates, which would be different to the actual billing.

Best practices

Architects need to put in additional effort to understand their architecture and the Azure components being utilized. Based on active monitoring, audits, and usage, they should be fully aware of the SKU, size, and features of the architecture. This section will detail some of the best practices to be adopted from a cost optimization perspective.

Compute best practices

Compute refers to services that help in the execution of services. Some of the best practices related to compute are as follows:

- Choose the best location for your compute services, such as virtual machines. Choose a location where all Azure features and resources are available together in the same region. This will avoid egress traffic.
- Choose the optimal size for your virtual machines. A bigger virtual machine costs more than a smaller one, and a bigger virtual machine might not be required at all.

- Resize virtual machines according to demand. Azure releases new virtual machine sizes frequently. If a new size becomes available that is better suited to your needs, then it should be used.
- Shut down compute services when they are not needed. This particularly applies to non-production environments.
- Deallocate virtual machines rather than shutting them down. This will release all resources and consumption will stop.
- Use development/testing labs for development and testing purposes. They provide policies and auto-shutdown and auto-start features.
- With virtual machine scale sets, provision few virtual machines and increase their count based on demand.
- Choose the correct size (whether small, medium, or large) for application gateways. They are backed up by virtual machines and can help reduce costs if sized optimally. Also, choose the basic tier application gateway if a web application firewall is not needed.
- Choose the correct tiers for virtual private network gateways (including a basic virtual private network, standard, high performance, and ultra performance).
- Minimize network traffic between Azure regions by collocating resources in the same region.
- Use a load balancer with a public IP to access multiple virtual machines rather than assigning a public IP to each virtual machine.
- Monitor virtual machines and their performance and usage metrics. Based on those metrics, determine whether you want to upscale or scale out the virtual machine. Consultation of the metrics could also result in downsizing the virtual machines.

Storage best practices

Here are some best practices for optimizing storage for cost:

- Choose the appropriate storage redundancy type (whether GRS, LRS, or RA-GRS). GRS is costlier than LRS, for instance.
- Archive storage data to cool or archive the access tier. Keep data that is frequently accessed in the hot tier.
- Remove blobs that are not required.
- Delete virtual machine operating system disks explicitly after deleting the virtual machine, if they are not needed.

- Storage accounts are metered based on their size, write, read, list, and container operations.
- Prefer standard disks over premium disks. Use premium disks only if business requirements demand it.
- Use the **Content Delivery Network (CDN)** and caching for static files instead of fetching them from storage every time.

Platform as a Service (PaaS) best practices

Some of the best practices, if PaaS is the preferred deployment model, are listed here:

- Choose the appropriate Azure SQL tier (whether basic, standard, premium RS, or premium) and appropriate performance levels in terms of DTUs.
- Choose appropriately between single databases and elastic databases. If there are a lot of databases, it is more cost-efficient to use elastic databases compared to single databases.
- Ensure Azure SQL security – encrypt data at rest and in motion, data masking, threat protection are enabled.
- Ensure that backup strategy and data replication is set up according to business demands.
- Ensure there is redundancy for web apps with multi-region availability using traffic manager.
- Use Redis cache and CDN for faster delivery of data and pages.
- Re-architect your solutions to use PaaS solutions (such as serverless solutions and microservices in containers) rather than **Infrastructure as a Service (IaaS)** solutions. These PaaS solutions remove maintenance costs and are available on the consumption-per-minute basis. If you do not consume these services, there is no cost, even though your code and services will still be available round the clock.

General best practices

Some of other general best practices are listed here:

- Resource costs differ across regions. Try using a region with lower costs.
- Enterprise Agreements provide the best discounts. If you are not in an Enterprise Agreement, try to use one for the cost benefits.
- If Azure costs can be prepaid, then discounts for all kinds of subscription can be gained.
- Delete or remove unused resources. Figure out what resources are underutilized and reduce their SKU or size. If they are not needed, then delete them.
- Use Azure Advisor and take its recommendations seriously.

Summary

Cost management and administration is an important activity when dealing with the cloud. This is primarily because the monthly expense could be very low, but can be very high if proper attention is not given to it. Architects should design their applications in such a manner as to minimize cost as much as possible. They should use appropriate Azure resources, appropriate SKU, tier, and size, and should know when to start, stop, scale up, scale out, scale down, scale in, transfer data, and more. Proper cost management will ensure that actual expenses meet budgetary expenses.

The next chapter of this book deals with the monitoring and auditing capabilities of Azure.

9

Designing Policies, Locks, and Tags

Azure is a versatile cloud platform. Customers can not only create and deploy their applications; they can also actively manage and govern their environments. Clouds generally follow a pay-as-you-go paradigm, where a customer subscribes and can deploy virtually anything to the cloud. It could be as small as a basic virtual machine, or it could be thousands of virtual machines with higher SKUs. Azure will not stop any customer from provisioning the resources they want to provision. Within an organization, there could be a large number of people with access to the organization's Azure subscription. There needs to be a governance model in place so that only necessary resources are provisioned by people who have the right to create them. Azure provides resource management features, such as Azure **Role-Based Access Control (RBAC)**, policies, and locks, for managing and providing governance for resources.

Another major aspect of governance is cost, usage, and information management. An organization's management would always want to be kept updated about their cloud consumption and costs. They would like to identify what team, department, or unit is using what percentage of their total cost. In short, they want to have reports based on various dimensions about consumption and cost. Azure provides a tagging feature that can help provide this kind of information on the fly.

In this chapter, we will cover the following topics:

- Azure tags
- Azure policies
- Azure locks
- Azure RBAC
- Implementing Azure governance features

Azure tags

A tag is defined by the Oxford Dictionary (<https://en.oxforddictionaries.com/definition/tag>) as the following:

"a label attached to someone or something for the purpose of identification or to give other information."

Azure allows the tagging of resource groups and resources with name-value pairs. Tagging helps in the logical organization and categorization of resources. Azure also allows the tagging of 15 name-value pairs for a resource group and its resources. Although a resource group is a container for resources, tagging a resource group does not mean the tagging of its constituent resources. Resource groups and resources should be tagged based on their usage, which will be explained later in this section. Tags work at a subscription level. Azure accepts any name-value pairs, and so it is important for an organization to define both the names and their possible values.

But why is tagging important? In other words, what problems can be solved using tagging? Tagging has the following benefits:

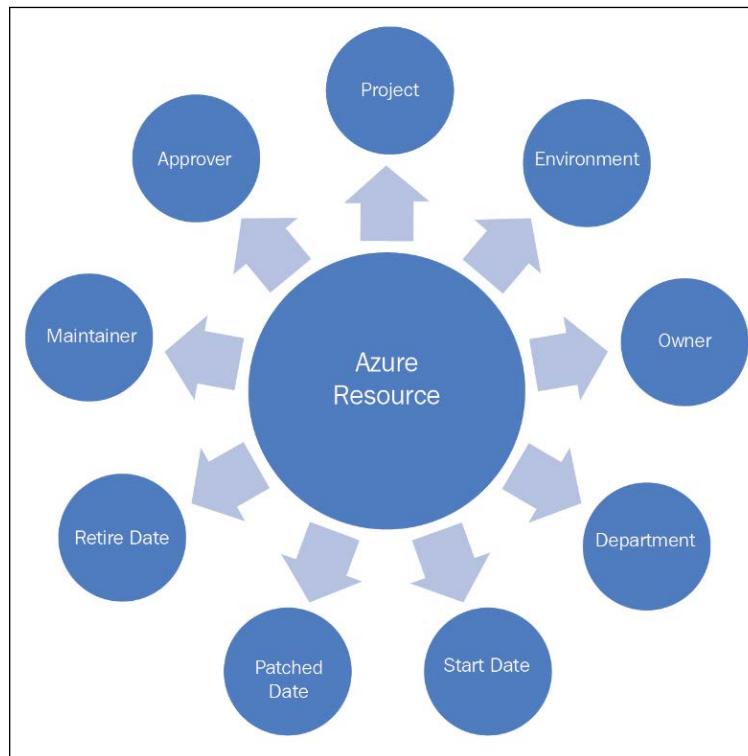
- **Categorization of resources:** An Azure subscription can be used by multiple departments within an organization. It is important for the management team to identify the owners of any resources. Tagging helps in assigning identifiers to resources that can be used to represent departments or roles.
- **Information management for Azure resources:** Again, Azure resources can be provisioned by anyone with access to the subscription. Organizations would like to have a proper categorization of resources in place to comply with information management policies. Such policies can be based on application life cycle management, such as management of the development, testing, and production environments. Such policies could be based on the usage, or based on any other priorities. Each organization has their own way of defining information categories, and Azure caters for this with tags.
- **Cost management:** Tagging in Azure can help in identifying resources based on their categorization. Queries can be executed against Azure to identify cost per category, for instance. For example, the cost of resources in Azure for the development of an environment for the finance department and the marketing department can be easily ascertained. Moreover, Azure also provides billing information based on tags. This helps in identifying the consumption rates of teams, departments, or groups.

Tags in Azure do have certain limitations, however:

- Azure allows a maximum of 15 tag name-value pairs to be associated with resource groups.
- Tags are non-inheritable. Tags applied to a resource group do not apply to the individual resources within it. However, it is quite easy to forget to tag resources when provisioning them. Azure policies are the mechanism to ensure that tags are tagged with the appropriate value during provision time. We will consider the details of such policies later in this chapter.

Tags can be assigned to resources and resource groups using PowerShell, Azure CLI 2.0, Azure Resource Manager templates, the Azure portal, and the Azure Resource Manager REST APIs.

An example of information management categorization using Azure tags is shown here. In this example, the **Department**, **Project**, **Environment**, **Owner**, **Approver**, **Maintainer**, **Start Date**, **Retire Date**, and **Patched Date** name-value pairs are used to tag resources. It is extremely easy to find all the resources for a particular tag or a combination of tags using PowerShell, the Azure CLI, or REST APIs:



Tags with PowerShell

Tags can be managed using PowerShell, Azure Resource Manager templates, the Azure portal, and REST APIs. In this section, PowerShell will be used to create and apply tags. PowerShell provides a cmdlet for retrieving and attaching tags to resource groups and resources:

- To retrieve tags associated with a resource using PowerShell, the `Find-AzureRMResource` cmdlet can be used:

```
(Find-AzureRmResource -TagName Dept -TagValue Finance).Name
```

- To retrieve tags associated with a resource group using PowerShell, the following command can be used:

```
(Find-AzureRmResourceGroup -Tag @{ Dept="Finance" }).Name
```

- To set tags to a resource group, the `Set-AzureRmResourceGroup` cmdlet can be used:

```
◦ Set-AzureRmResourceGroup -Name examplegroup -Tag @{  
    Dept="IT"; Environment="Test" }
```

- To set tags to a resource, the `Set-AzureRmResource` cmdlet can be used:

```
◦ Set-AzureRmResource -Tag @{ Dept="IT"; Environment="Test" }  
    -ResourceName examplevnet -ResourceGroupName examplegroup
```

Tags with Azure Resource Manager templates

Azure Resource Manager templates also help in defining tags for each resource. They can be used to assign multiple tags to each resource, as follows:

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

```
        "name": "Standard_LRS"
    },
    "kind": "Storage",
    "properties": {}
}
]
}
```

In the previous example, a couple of tags, `Dept` and `Environment`, are added to a Storage account resource using Azure Resource Manager templates.

Resource groups versus resources

It is a must for architects to decide the taxonomy and information architecture for Azure resources and resource groups. They should identify the categories by which resources will be classified based on the query requirements. However, they must also identify whether tags should be attached to individual resources or to resource groups.

If all resources within a resource group need the same tag, then it is better to tag the resource group, rather than tagging each resource. It is important to take the queries on tags into consideration before finalizing whether tags should be applied at the resource level or the resource group level. If the queries relate to individual resource types across a subscription and across resource groups, then assigning tags to individual resources makes more sense. However, if identifying resource groups is enough for your queries to be effective, then tags should be applied only to resource groups.

Azure policies

In the previous section, we talked about applying tags for Azure deployments. Tags are great for organizing resources; however, there is one more thing that was not discussed: how do organizations ensure that tags are applied for every deployment? There should be automated enforcement of Azure tags to resources and resource groups. There is no check from Azure to ensure that appropriate tags will be applied to resources and resource groups. Now, this is not just specific to tags –this applies to the configuration of any resource on Azure. For example, you may wish to restrict where your resources can be provisioned geographically (to only the US-East region, for instance).

You might have guessed by now that this section is all about formulating a governance model on Azure. Governance is an important element in Azure because it helps to bring costs under control. It also ensures that everyone accessing the Azure environment is aware of organizational priorities and processes. It helps in defining organizational conventions for managing resources.

Each policy can be built using multiple rules, and multiple policies can be applied to a subscription or resource group. Based on whether the rules are satisfied, policies can execute various actions. An action could be to deny an on-going transaction, to audit a transaction (which means writing to logs and allowing it to finish), or to append metadata to a transaction if it's found to be missing.

Policies could be related to the naming convention of resources, the tagging of resources, the types of resources that can be provisioned, the location of resources, or any combination of these rules.

Built-in policies

Azure provides infrastructure for the creation of custom policies; however, it also provides some out-of-box policies that can be used for governance. These policies relate to allowed locations, allowed resource types, and tags. More information for these built-in policies can be found at <https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-manager-policy>.

Policy language

Azure policies use JSON to define and describe policies.

There are two steps in policy adoption. The policy should be defined and then it should be applied and assigned. Policies have scope and can be applied at the subscription and resource group level.

Policies are defined using `if...` then blocks, similar to any popular programming language. The `if` block is executed to evaluate the conditions and based on the result of those conditions, the `then` block is executed:

```
{
  "if": {
    <condition> | <logical operator>
  },
  "then": {
    "effect": "deny | audit | append"
  }
}
```

Azure policies not only allow simple `if` conditions, but multiple `if` conditions can be joined together logically to create complex rules. These conditions can be joined using **AND**, **OR**, and **NOT** operators:

- The **AND** syntax requires all conditions to be true.
- The **OR** syntax requires one of the conditions to be true.
- The **NOT** syntax inverts the result of the condition.

The AND syntax is shown next. It is represented by the `allOf` keyword:

```
"if": {
  "allOf": [
    {
      "field": "tags",
      "containsKey": "application"
    },
    {
      "field": "type",
      "equals": "Microsoft.Storage/storageAccounts"
    }
  ]
},
```

The OR syntax is shown next. It is represented by the `anyOf` keyword:

```
"if": {
  "anyOf": [
    {
      "field": "tags",
      "containsKey": "application"
    },
    {
      "field": "type",
      "equals": "Microsoft.Storage/storageAccounts"
    }
  ]
},
```

The NOT syntax is shown next. It is represented by the `not` keyword:

```
"if": {
  "not": [
    {
      "field": "tags",
      "containsKey": "application"
    },
    {
      "field": "type",
      "equals": "Microsoft.Storage/storageAccounts"
    }
  ]
},
```

In fact, these logical operators can be combined together, as follows:

```
"if": {
  "allOf": [
    {
      "not": {
        "field": "tags",
        "containsKey": "application"
      }
    },
    {
      "field": "type",
      "equals": "Microsoft.Storage/storageAccounts"
    }
  ]
},
```

This is very similar to the use of `if` conditions in popular programming languages such as C# and Node.js:

```
If ("type" == "Microsoft.Storage/storageAccounts") {
  Deny
}
```

It is important to note that there is no `allow` action, although there is a `Deny` action. This means that policy rules should be written with the possibility of denial in mind. Rules should evaluate the conditions and return `Deny` if they return true.

Allowed fields

The fields that are allowed in policies are as follows:

- Name
- Kind
- Type
- Location
- Tags
- Tags.*
- Property aliases

Azure locks

Locks are mechanisms for stopping certain activities on resources. RBAC provides rights to users, groups, and applications within a certain scope. There are out-of-the-box RBAC roles, such as owner, contributor, and reader. With the contributor role, it is possible to delete or modify a resource. How can such activities be prevented despite the user having a contributor role? Enter Azure locks.

Azure locks can help in two ways:

- They can lock resources such that they cannot be deleted, even if you have owner access.
- They can lock resources in such a way that it can neither be deleted nor have its configuration modified.

Locks are typically very helpful for resources in production environments that should not be modified or deleted accidentally.

Locks can be applied at the levels of subscription, resource group, and individual resource. Locks can be inherited between subscriptions, resource groups, and resources. Applying a lock at the parent level will ensure that those at the child level will also inherit it. Even resources you add later will inherit the lock from the parent. The most restrictive lock in the inheritance takes precedence. Applying a lock at the resource level will also not allow the deletion of the resource group containing the resource.

Locks are applied only to operations that help in managing the resource, rather than operations that are within the resource. Users either need `Microsoft.Authorization/*` or `Microsoft.Authorization/locks/*` RBAC permissions to create and modify locks.

Locks can be created and applied through the Azure portal, Azure PowerShell, the Azure CLI, Azure Resource Manager templates, and REST APIs.

Creating a lock using an Azure Resource Manager template is done as follows:

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

```
{  
    "name": "[concat(parameters('lockedResource'), '/Microsoft.  
Authorization/myLock')]",  
    "type": "Microsoft.Storage/storageAccounts/providers/locks",  
    "apiVersion": "2015-01-01",  
    "properties": {  
        "level": "CannotDelete"  
    }  
}  
]  
}
```

Creating and applying a lock to a resource using PowerShell is done as follows:

```
New-AzureRmResourceLock -LockLevel CanNotDelete -LockName LockSite '  
-ResourceName examplesite -ResourceType Microsoft.Web/sites '  
-ResourceGroupName exempleresourcegroup
```

Creating and applying a lock to a resource group using PowerShell is done as follows:

```
New-AzureRmResourceLock -LockName LockGroup -LockLevel CanNotDelete '  
-ResourceGroupName exempleresourcegroup
```

Creating and applying a lock to a resource using the Azure CLI is done as follows:

```
az lock create --name LockSite --lock-type CanNotDelete \  
--resource-group exempleresourcegroup --resource-name examplesite \  
--resource-type Microsoft.Web/sites
```

Creating and applying a lock to a resource group using the Azure CLI is done as follows:

```
az lock create --name LockGroup --lock-type CanNotDelete \  
--resource-group exempleresourcegroup
```

Azure RBAC

Azure provides authentication using Azure Active Directory for its resources. Once an identity has been authenticated, the resources the identity should be allowed to access should be decided. This is known as authorization. Authorization evaluates the permissions that have been afforded to an identity. Anybody with access to an Azure subscription should be given just enough permissions so that their specific job can be performed, and nothing more.

Authorization is popularly also known as RBAC. RBAC in Azure refers to the assigning of permissions to identities within a scope. The scope could be a subscription, a resource group, or individual resources.

RBAC helps in the creation and assignment of different permissions to different identities. This helps in segregating duties within teams, rather than everyone having all permissions. RBAC helps in making people responsible for their job, because others might not even have the necessary access to perform it. It should be noted that providing permissions at a greater scope automatically ensures that child resources inherit those permissions. For example, providing an identity with read access for a resource group means that the identity will have read access to all the resources within that group, too.

Azure provides three general-purpose, built-in roles. They are as follows:

- The owner role, which has full access to all resources
- The contributor role, which has access to read/write resources
- The reader role, which has read-only permissions to resources

There are more roles provided by Azure, but they are resource-specific, such as the network contributor and security manager roles.

To get all roles provided by Azure for all resources, execute the `Get-AzureRmRoleDefinition` command in the PowerShell console.

Each role definition has certain allowed and disallowed actions. For example, the owner role has all actions permitted; none of the actions are prohibited:

```
PS C:\Users\rimodi> Get-AzureRmRoleDefinition -Name "owner"

Name          : Owner
Id            : 8e3af657-a8ff-443c-a75c-2fe8c4bcb635
IsCustom      : False
Description   : Lets you manage everything, including access to
resources.
Actions        : { * }
NotActions    : { }
AssignableScopes : { / }
```

Each role comprises multiple permissions. Each resource provides a list of operations. The operation supported by a resource can be obtained using the `Get-AzureRmProviderOperation` cmdlet. This cmdlet takes the name of the provider and resource to retrieve the operations:

```
Get-AzureRmProviderOperation -OperationSearchString "Microsoft.
Insights/*"
```

This will result in the following output:

```
PS C:\Users\rimodi> get-AzureRmProviderOperation  
-OperationSearchString "Microsoft.Insights/*" | select operation  
  
Operation  
-----  
Microsoft.Insights/Register/Action  
Microsoft.Insights/AlertRules/Write  
Microsoft.Insights/AlertRules/Delete  
Microsoft.Insights/AlertRules/Read  
Microsoft.Insights/AlertRules/Activated/Action  
Microsoft.Insights/AlertRules/Resolved/Action  
Microsoft.Insights/AlertRules/Throttled/Action  
Microsoft.Insights/AlertRules/Incidents/Read  
Microsoft.Insights/MetricDefinitions/Read  
Microsoft.Insights/eventtypes/values/Read  
Microsoft.Insights/eventtypes/digestevents/Read  
Microsoft.Insights/Metrics/Read  
Microsoft.Insights/LogProfiles/Write  
Microsoft.Insights/LogProfiles/Delete  
Microsoft.Insights/LogProfiles/Read  
Microsoft.Insights/Components/Write  
Microsoft.Insights/Components/Delete  
Microsoft.Insights/Components/Read  
Microsoft.Insights/AutoscaleSettings/Write  
Microsoft.Insights/AutoscaleSettings/Delete  
Microsoft.Insights/AutoscaleSettings/Read  
Microsoft.Insights/AutoscaleSettings/Scaleup/Action  
Microsoft.Insights/AutoscaleSettings/Scaledown/Action  
    Microsoft.Insights/AutoscaleSettings/providers/Microsoft.Insights/  
MetricDefinitions/Read  
Microsoft.Insights/ActivityLogAlerts/Activated/Action  
Microsoft.Insights/DiagnosticSettings/Write  
Microsoft.Insights/DiagnosticSettings/Delete  
Microsoft.Insights/DiagnosticSettings/Read  
Microsoft.Insights/LogDefinitions/Read  
Microsoft.Insights/Webtests/Write  
Microsoft.Insights/Webtests/Delete  
Microsoft.Insights/Webtests/Read  
Microsoft.Insights/ExtendedDiagnosticSettings/Write  
Microsoft.Insights/ExtendedDiagnosticSettings/Delete  
Microsoft.Insights/ExtendedDiagnosticSettings/Read
```

Custom Roles

Custom roles are created by combining multiple permissions. For example, a custom role can consist of operations from multiple resources, as follows:

```
$role = Get-AzureRmRoleDefinition "Virtual Machine Contributor"
$role.Id = $null $role.Name = "Virtual Machine Operator" $role.
Description = "Can monitor and restart virtual machines." $role.
Actions.Clear() $role.Actions.Add("Microsoft.Storage/*/read") $role.
Actions.Add("Microsoft.Network/*/read") $role.Actions.Add("Microsoft.
Compute/*/read") $role.Actions.Add("Microsoft.Compute/virtualMachines/
start/action") $role.Actions.Add("Microsoft.Compute/virtualMachines/
restart/action")
$role.Actions.Add("Microsoft.Authorization/*/read") $role.Actions.
Add("Microsoft.Resources/subscriptions/resourceGroups/read") $role.
Actions.Add("Microsoft.Insights/alertRules/*") $role.Actions.
Add("Microsoft.Support/*") $roleAssignableScopes.Clear() $role.
AssignableScopes.Add("/subscriptions/c276fc76-9cd4-44c9-99a7-
4fd71546436e") $role.AssignableScopes.Add("/subscriptions/e91d47c4-
76f3-4271-a796-21b4ecfe3624") New-AzureRmRoleDefinition -Role $role
```

How are locks different from RBAC?

Locks are not the same as RBAC. RBAC helps in allowing or denying permissions for resources. These permissions relate to performing operations, such as read, write, and update operations on resources. Locks, on the other hand, relate to disallowing permissions to configure or delete resources.

An example of implementing Azure governance features

In this section, we will go through a sample architecture implementation for a fictitious organization that wants to implement Azure governance and cost management features.

Background

Company Inc is a worldwide company that is implementing a social media solution on an Azure IaaS platform. They use web servers and application servers deployed on Azure virtual machines and networks. Azure SQL Server acts as the backend database.

RBAC for Company Inc

The first task is to ensure that the appropriate teams and application owners can access their resources. It is recognized that each team has different requirements. For the sake of clarity, Azure SQL is deployed in a separate resource group to the Azure IaaS artifacts.

The administrator assigns the following roles for the subscription:

Role	Assigned to	Description
Owner	Administrator	Manages all resource groups and the subscription.
Security manager	Security administrators	This role allows users to look at Azure Security Center and the status of the resources.
Contributor	Infrastructure management	Managing virtual machines and other resources.
Reader	Developers	Can view resources, but cannot modify them. Developers are expected to work in their development/testing environments.

Azure policies

The company should implement Azure policies to ensure that its users always provision resources according to the company guidelines.

Azure policies govern various aspects related to the deployment of resources. The policies will also govern updates after the initial deployment.

Some of the policies that should be implemented are given in the following section.

Deployments to certain locations

Azure resources and deployments can only be executed for certain chosen locations. It would not be possible to deploy resources in regions outside of the policy. For example, the regions that are allowed are West Europe and East US. It should not be possible to deploy resources in any other region.

Tags of resources and resource groups

Every resource in Azure, including the resource groups, will mandatorily have tags assigned to it. The tags will include, as a minimum, details about the department, environment, creation data, and project name.

Diagnostic logs and Application Insights for all resources

Every resource deployed on Azure should have diagnostic logs and application logs enabled wherever possible.

Azure locks

The company should implement Azure locks to ensure that important and crucial resources are not deleted accidentally.

Every resource that is crucial for the functioning of a solution needs to be locked down. It means that even the administrators of the services running on Azure do not have the capability to delete these resources. The only way to delete the resource is to remove the lock first.

All production and pre-production environments, apart from the development and testing environment, would be locked for deletion.

All development and testing environments that have single instances would also be locked for deletion.

All resources related to <https://www.eversource.com/content/> would be locked for deletion for all environments from development to production.

All shared resources will be locked for deletion irrespective of the environment.

Summary

Governance and cost management are among the top priorities for companies moving to the cloud. Having an Azure subscription with a pay-as-you-go scheme can harm the company budget, because anyone with access to the subscription can provision as many resources as they like. Some resources are free, but others are expensive. It is important for organizations to remain in control of their cloud costs. Tags help in generating billing reports. These reports could be based on departments, projects, owners, or any other criteria. While cost is important, governance is equally important. Azure provides locks, policies, and RBAC to implement proper governance. Policies ensure that resource operations can be denied or audited, locks ensure that resources cannot be modified or deleted, and RBAC ensures that employees have the right permissions to perform their jobs. With these four features, companies can have sound governance and cost control for their Azure deployments.

DevOps on Azure will be the focus of the next chapter, in which some of its most important concepts and implementation techniques will be discussed.

10

Azure Solutions Using Azure Container Services

Containers are the most talked-about technologies these days. Corporations are talking about containers rather than virtual machines and want to see them in almost all their solutions. The solutions could be microservices, serverless functions, web applications, web APIs, or just about anything.

In this chapter, we will delve into most of the Azure features that are to do with containers, including the following:

- Azure Kubernetes Service
- Containers on virtual machines
- Azure Container Registry
- Azure Container Instance
- Web Apps for Containers
- Hosting images on Docker Hub

We will not go into the details and fundamentals of containers, and will concentrate more on Azure-specific container services.

Every container needs an image as a starting point. An image can be created using a Dockerfile or can be downloaded from a container registry. We will be using a single image for all our samples in this chapter, and this image will be created from a Dockerfile.

The content of the Dockerfile used for all examples in this chapter is listed here. The entire code and `samplewebapp` solution is available with the accompanying source code:

```
FROM microsoft/dotnet:2.1-aspnetcore-runtime-nanoserver-sac2016 AS
base
WORKDIR /app
EXPOSE 80

FROM microsoft/dotnet:2.1-sdk-nanoserver-sac2016 AS build
WORKDIR /src
COPY [ "samplewebapp/samplewebapp.csproj" , "samplewebapp/" ]
RUN dotnet restore "samplewebapp/samplewebapp.csproj"
COPY . .
WORKDIR "/src/samplewebapp"
RUN dotnet build "samplewebapp.csproj" -c Release -o /app

FROM build AS publish
RUN dotnet publish "samplewebapp.csproj" -c Release -o /app

FROM base AS final
WORKDIR /app
COPY --from=publish /app .
ENTRYPOINT [ "dotnet" , "samplewebapp.dll" ]
```

Azure Container Registry

Prior to Azure Container Registry, Docker Hub was the most well-known registry service for Docker images.

Azure Container Registry is an alternative repository to Docker Hub. A registry is a location on the internet that provides listings of images, along with facilities to upload and download the images on demand. There are two types of registries:

- Public
- Private

A public repository, as the name suggests, is public in nature and images from it can be downloaded and used by anyone. However, the upload of images to a public repository is discretionary and, depending on the provider, may or may not allow the upload of images.

On the other hand, private repositories are meant only for people who have access to the repository. They need to authenticate before they can upload or download images.

Docker Hub provides the ability to create user accounts, and these accounts can have public as well as private repositories.

In fact, Microsoft has accounts on Docker Hub as well, where all well-known Windows images are available. You can find all Microsoft images by executing the `docker search Microsoft` command, as shown in the following screenshot:

C:\Users\citynextadmin>docker search Microsoft	NAME	DESCRIPTION	STARS
	microsoft/dotnet	Official images for .NET Core and ASP.NET Co...	1395
	microsoft/mssql-server-linux	Official images for Microsoft SQL Server on ...	1078
	microsoft/aspnet	Microsoft IIS images	819
	microsoft/windowsservercore	The official Windows Server Core base image	644
	microsoft/aspnetcore	Official images for running compiled ASP.NET...	582
	microsoft/nanoserver	The official Nano Server base image	473
	microsoft/iis	Microsoft IIS images	356
	microsoft/mssql-server-windows-developer	Official Microsoft SQL Server Developer Edit...	289
	microsoft/mssql-server-windows-express	Official Microsoft SQL Server Express Editio...	281
	microsoft/aspnetcore-build	Official images for building ASP.NET Core ap...	270
	microsoft/azure-cli	Official images for Microsoft Azure CLI	156
	microsoft/powershell	PowerShell for every system!	145
	microsoft/vsts-agent	Official images for the Visual Studio Team S...	116
	microsoft/dynamics-nav	Official images for Microsoft Dynamics NAV o...	108
	microsoft/dotnet-samples	.NET Core Docker Samples	70
	microsoft/bcsandbox	Business Central Sandbox	53
	microsoft/mssql-tools	Official images for Microsoft SQL Server Com...	51
	microsoft/oms	Monitor your containers using the Operations...	41
	microsoft/cntk	CNTK images from github.com/Microsoft/CNTK-d...	38
	microsoft/wcf	Microsoft WCF images	28
	microsoft/dotnet-nightly	Preview images for .NET Core and ASP.NET Cor...	23
	microsoft/dotnet-framework-build	The .NET Framework build images have moved t...	17
	microsoft/mmlspark	Microsoft Machine Learning for Apache Spark	7
	microsoft/aspnetcore-build-nightly	Images to build preview versions of ASP.NET ...	4
	microsoft/cntk-nightly	CNTK nightly image from github.com/Microsoft...	2

As mentioned before, Microsoft also provides an alternate registry as an Azure service, known as **Azure Container Registry**. It provides similar functionality to that of Docker Hub.

One advantage of using Azure Container Registry over Docker Hub is that, if you are using an image from Azure Container Registry that is in the same Azure location that is hosting containers based on the image, the image can be downloaded faster using the Microsoft backbone network instead of going through the internet. For example, hosting a container on Azure Kubernetes Service whose source image is within Azure Container Registry will be much faster than Docker Hub.

Azure Container Registry is a managed service that will take care of all operational needs concerning images, such as storage and availability. Images and their layers are stored in the registry and are downloaded to the destination host for creation of containers on demand. It is the responsibility of Azure to take care of these images and their layers.

The beauty of Azure Container Registry is that it works with the Azure CLI, PowerShell, and the Azure portal. And it does not stop there: it also has high fidelity with the Docker command line. You can use Docker commands to upload and download images from Azure Container Registry instead of Docker Hub.

Let's now focus on how to work with Azure Container Registry using the Azure CLI:

1. Obviously, the first task is to log into Azure using the `az login` command:

```
C:\Users\citynextadmin>az login
```

2. If you have multiple subscriptions associated with the same login, select an appropriate subscription for working with Azure Container Registry:

```
C:\Users\citynextadmin>az account set --subscription [REDACTED]
```

3. Create a new resource group for hosting a new instance of Azure Container Registry:

```
C:\Users\citynextadmin>az group create --name rg03 --location "west europe" --verbose
{
  "id": "/subscriptions/[REDACTED]/resourceGroups/rg03",
  "location": "westeurope",
  "managedBy": null,
  "name": "rg03",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null
}
```

4. The Azure CLI provides `acr` commands. Create a new instance of Azure Container Registry using the following command:
`az acr create`
5. You need to provide the SKU of Azure Container Registry, a name, and the name of a resource group. It can also optionally enable admin privileges.

6. The SKUs available for Azure Container Registry are as follows:
 - **Basic:** This is not recommended for production scenarios and is more suitable for development/test work. This is because the amount of resources available for storage and bandwidth is constrained compared to higher SKUs.
 - **Standard:** This has all features of Basic along with higher configuration and availability of resources. This is suitable for production deployments.
 - **Premium:** This, again, has all the features of the Standard SKU along with higher resource availability. This has additional features such as geo-replication.
 - **Classic:** This is not a managed service and the images stored using this SKU are stored on user-provided storage accounts. Users need to actively manage these storage accounts in terms of security, administration, and governance.
7. The output from the following contains important information that is used quite frequently for uploading and downloading images. These include the `loginServer` and `name` properties. The username of Azure Container Registry is the same as its name:

```
C:\Users\citynextadmin>az acr create --sku basic --resource-group rg03 --name sampleappbookacr --verbose --admin-enabled
{
  "adminUserEnabled": true,
  "creationDate": "2018-12-31T07:15:48.719352+00:00",
  "id": "/subscriptions/_/resourceGroups/rg03/providers/Microsoft.ContainerRegistry/registries/sampleappbookacr",
  "location": "westeurope",
  "loginServer": "sampleappbookacr.azurecr.io",
  "name": "sampleappbookacr",
  "provisioningState": "Succeeded",
  "resourceGroup": "rg03",
  "sku": {
    "name": "Basic",
    "tier": "Basic"
  },
  "status": null,
  "storageAccount": null,
  "tags": {},
  "type": "Microsoft.ContainerRegistry/registries"
}
```

8. The service on the portal is configured as shown in the following screenshot:

The screenshot shows the 'Access keys' page for a container registry named 'samplewebappbookacr'. The left sidebar lists various options like Overview, Activity log, Access control (IAM), Tags, Quick start, Events, Settings, Services, Repositories, and Webhooks. The 'Access keys' option is selected. The main area shows the following details:

- Registry name: samplewebappbookacr
- Login server: samplewebappbookacr.azurecr.io
- Admin user: Status is 'Enable' (button is blue)
- Username: samplewebappbookacr
- Two sets of credentials:

NAME	PASSWORD
password	TcU1V3G8vwNl=f6JkxBCQgFDs5=kijfb
password2	d5Jkcu08phgoVgFw/oAhShYLGzWD02qX

9. If admin is enabled, the credentials for the admin can be fetched using a command as shown in the following screenshot. There are two passwords generated, which can be used to swap when necessary. Login is needed for pushing images and is not required for pulling images from Azure Container Registry:

```
C:\Users\citynextadmin>az acr credential show --name sampleappbookacr --resource-group rg03 --verbose
{
  "passwords": [
    {
      "name": "password",
      "value": "igzgtUx1ehYn2/92KvNk=+X4UtbtUQim"
    },
    {
      "name": "password2",
      "value": "XmbIrrHpl0H9DQmLu/ReXmYBietU8ht3"
    }
  ],
  "username": "sampleappbookacr"
}
```

10. Using the username and password data (which we have by now), it is possible to log into Azure Container Registry:

```
C:\Users\citynextadmin>az acr login --name sampleappbookacr --password "igzgtUxlehYn2/92KvNk+=X4Utb1UQim" --resource-group rg03 --username sampleappbookacr
Login Succeeded
WARNING! Your password will be stored unencrypted in C:\Users\citynextadmin\.docker\config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
```

11. You will need to log into Azure Container Registry if it is protected by password. The next task is to prepare local images so that they can be uploaded to Azure Container Registry. This preparation needs to tag the local images with the server name of the registry.
12. The following screenshot shows a command in which a sample image named `samplewebapp` is tagged using the `tag` command to `sampleappbooacr.azurecr.io/samplewebapp:latest`. Here, `sampleappbooacr.azurecr.io` refers to the server name that was created earlier as part of the creation of the registry. Note that Docker commands are used for tagging the images:

```
C:\Users\citynextadmin>docker tag samplewebapp:latest sampleappbookacr.azurecr.io/samplewebapp:latest
```

13. After tagging using the `docker push` command, the image can be pushed to Azure Container Registry:

```
C:\Users\citynextadmin>docker push sampleappbookacr.azurecr.io/samplewebapp:latest
The push refers to repository [sampleappbookacr.azurecr.io/samplewebapp]
036d27f25d59: Pushed
f6ac38202aab: Pushed
7605499be4ca: Pushed
1bd33b5316f6: Pushed
f3eff1ccb92a: Pushed
cbe35d8d94e0: Pushed
e36c5fabaebe: Pushed
8b28210b2d46: Pushed
cc80cb0b7044: Pushed
e4a9564a3ea3: Pushed
f5c673c58dcc: Skipped foreign layer
6c357baed9f5: Skipped foreign layer
latest: digest: sha256:da53d5ab40890824b24712e37129819f2d4b65d8eb1df5c5df8b2bc65786d1fd size: 3026
```

14. From now, it is possible for anyone who has knowledge about the `sampleappbookacr` registry to consume this image.

The registry on the portal is shown in the following screenshot:

The screenshot shows the Azure Container Registry interface. On the left, there's a sidebar with options like Overview, Activity log, Access control (IAM), Tags, Quick start, Events, Settings (Access keys, Locks, Automation script), and Services (Repositories). The main area has two tabs: 'samplewebappbookacr - Repositories' and 'samplewebapp'. The 'samplewebapp' tab is active, showing a search bar, a 'Refresh' button, and a table with columns for REPOSITORIES and TAGS. Under REPOSITORIES, there's one entry: 'samplewebapp'. Under TAGS, there's one entry: 'latest'. Both entries have a '...' button next to them.

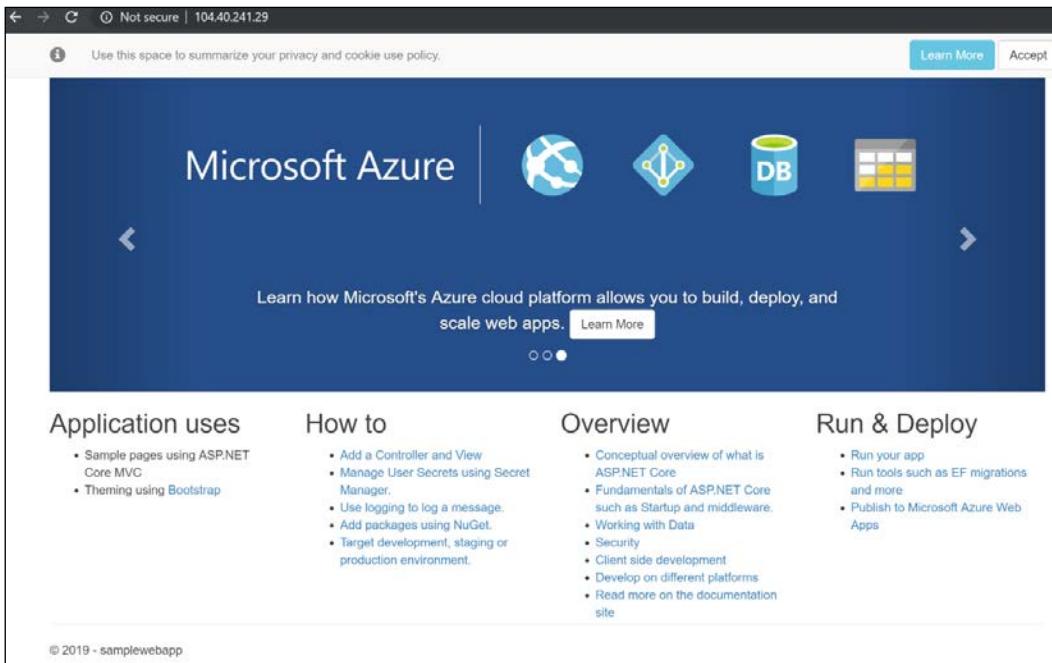
15. Create a container from the image that was just uploaded to Azure Container Registry. For this exercise, create a new virtual machine based on Windows Server 2016 or Windows 10 that has containers enabled, and execute the command, as shown in the following screenshot, to create a new container using the samplewebapp image from the samplewebappbookacr registry:

```
C:\Users\citynextadmin>docker run -it -p 80:80 samplewebappbookacr.azurecr.io/samplewebapp:latest powershell
```

16. It is important to note that, even if the image is not locally available, the command will download the images while executing the command and then create a container out of it. The output from running the container is shown in the following screenshot:

```
Administrator: Command Prompt - docker run -it -p 80:80 samplewebapp:latest <enter>
Hosting environment: Production
Content root path: C:\app
Now listening on: http://[::]:80
Application started. Press Ctrl+C to shut down.
```

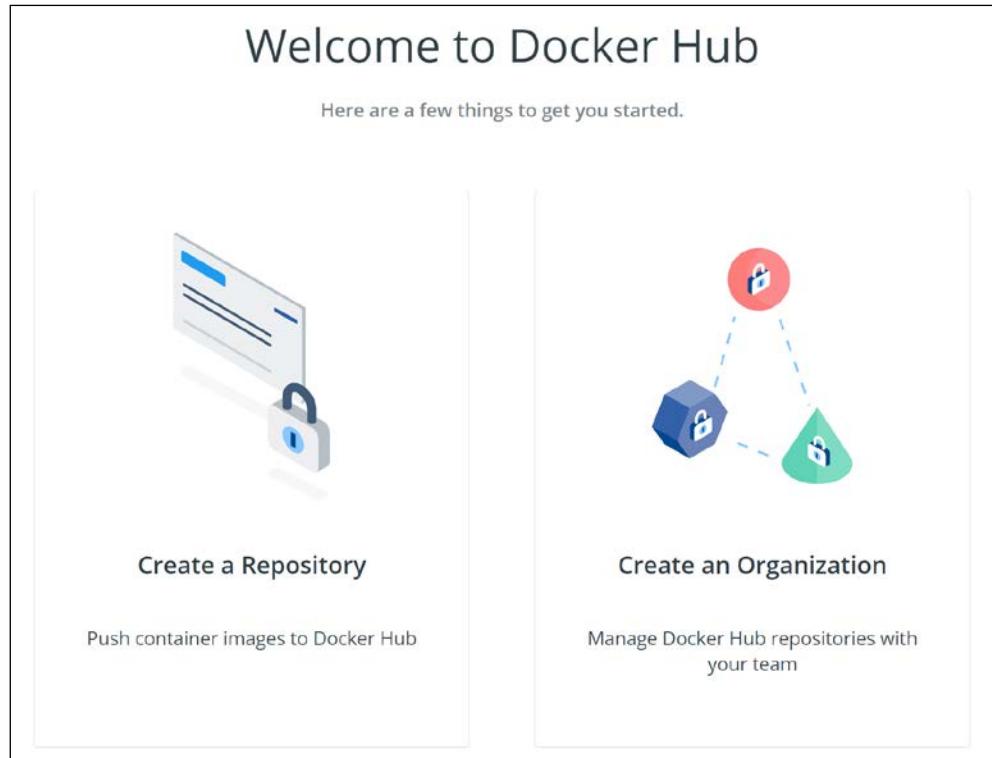
17. The result shown in the following screenshot displays that the web application is currently running:



The Dockerfile and solution file used to create the local image are available as part of the source code. It is also possible to upload the same image to Docker Hub.

18. The first step is to have an account on Docker Hub. For this, a new account or an existing account can be used.
19. After creating the account, log in to your account.

20. Create a new repository by selecting **Create a Repository**, as shown in the following screenshot:



21. Provide a name and select a **Public** repository:

The screenshot shows the 'Create Repository' page on Docker Hub. At the top, there's a header with 'Repositories' and 'Create'. Below it, a 'Pro tip' section with the text: 'You can push a new image to this repository using the CLI' and the command: 'docker tag local-image:tagname new-repo:tagname' followed by 'docker push new-repo:tagname'. A note says 'Make sure to change tagname with your desired image repository tag.' In the main form, the 'Repository name' field contains 'samplewebapp'. The 'Visibility' section has two options: 'Public' (selected) and 'Private'. The 'Public' option is described as 'Public repositories appear in Docker Hub search results'. The 'Private' option is described as 'Only you can see private repositories'. Below this, there's a 'Build Settings (optional)' section with a note about 'Autobuild triggers a new build with every git push to your source code repository' and a link to 'Learn More'. It also includes a note about re-linking GitHub or Bitbucket accounts with a link to 'Learn More'. At the bottom, there are three buttons: 'Cancel' (red), 'Create' (blue), and 'Create & Build' (blue).

22. The next screenshot shows the command to be executed for pushing images to this repository:

The screenshot shows a Docker Hub repository page for 'azureforarchitects / samplewebapp'. At the top, it says 'Using 0 of 1 private repositories. [Get more](#)'. Below that is a navigation bar with tabs: General (which is selected), Tags, Builds, Timeline, Collaborators, Webhooks, and Settings. The main content area starts with a repository summary: 'azurerefarchitects / samplewebapp', 'this repos contains image for samplewebapp for book', and 'Last pushed: a few seconds ago'. To the right of this is a 'Docker commands' section with the text 'To push a new tag to this repository.' and the command 'docker push azureforarchitects/samplewebapp:tagname'. Below this is a 'Tags' section which says 'This repository contains 1 tag(s)' and lists 'latest' with a timestamp 'a few seconds ago'. There is also a link 'See all'. At the bottom is a 'Full Description' section with the note 'Repository description is empty. Click [here](#) to edit.'

23. From your virtual machine containing the local image for `samplewebapp`, create a new tag. This is a similar exercise to what we did for Azure Container Registry; however, this time, it is for Docker Hub:

```
C:\Users\citynextadmin>docker tag samplewebapp:latest azurerefarchitects/samplewebapp:latest
```

24. Using the Docker CLI, log into the Docker Hub:

```
C:\Users\citynextadmin>docker login -u azurerefarchitects -p Dsc123..  
WARNING! Using --password via the CLI is insecure. Use --password-stdin.  
WARNING! Your password will be stored unencrypted in C:\Users\citynextadmin\.docker\config.json.  
Configure a credential helper to remove this warning. See  
https://docs.docker.com/engine/reference/commandline/login/#credentials-store  
Login Succeeded
```

25. Finally, push the newly tagged image to Docker Hub using the docker push command:

```
C:\Users\citynextadmin>docker push azureforarchitects/samplewebapp:latest
The push refers to repository [docker.io/azureforarchitects/samplewebapp]
036d27f25d59: Pushed
f6ac38202aab: Pushed
7605499be4ca: Pushed
1bd33b5316f6: Pushed
f3eff1ccb92a: Pushed
cbe35d8d94e0: Pushed
e36c5fabaebe: Pushed
8b28210b2d46: Pushed
cc80cb0b7044: Pushed
e4a9564a3ea3: Pushed
f5c673c58dcc: Skipped foreign layer
6c357baed9f5: Skipped foreign layer
latest: digest: sha256:da53d5ab40890824b24712e37129819f2d4b65d8eb1df5c5df8b2bc65786d1fd size: 3026
```

Now, this new image can be used in different hosts, including Azure App Services.

Azure Container Instances

In the last section, we created a container based on an Azure Container Registry image on a virtual machine. In this section, we will investigate Azure Container Instances. Azure Container Instances is a managed service provided by Azure to host containers. It is a service that helps in executing and running containers by providing a platform that is completely managed by Azure.

Using Azure Container Instances completely alleviates the need for any **Infrastructure as a Service (IaaS)** components, such as creating virtual machines to host containers. We will again use the Azure CLI to create Azure Container Instances as well.

Proceed with the steps as follows:

1. If you have not yet logged into Azure using the CLI, log into Azure using the az login command and select an appropriate subscription.
2. The command to log into Azure is shown next. It will open up a browser window asking for credentials to get started with the login process:

```
C:\Users\citynextadmin>az login
```

3. After successfully getting connected to Azure, the command for selecting an appropriate subscription should be executed as shown next:

```
C:\Users\citynextadmin>az account set --subscription [REDACTED]
```

4. Create a new resource group to host an Azure Container Instance using the following command:

```
C:\Users\citynextadmin>az group create --name rg04 --location "west_europe" --verbose [REDACTED]
```

5. Create an Azure Container Instance using the following command:

```
az container create
```

Pass the necessary information outlined in the following bullet points:

- The DNS name through which the container can be accessed.
- The number of CPUs allocated to the container.
- The fully qualified image name – this is the same name that we created in previous section (`samplewebappbookacr.azurecr.io/samplewebapp:latest`).
- The IP address type, whether public or private – a public IP address will ensure that the container is available to be consumed from the internet.
- The location of the container – in this case, it is the same as the location of the container registry. This helps in the faster creation of containers.
- The memory in GB to be allocated to the container.
- The name of the container.
- The OS of the container using the `-os-type` option. In this case it is Windows.
- The protocol and ports to be open. We open port 80 as our application will be hosted on that port.
- The restart policy is set to always. This will ensure that, even if the host that is running the container restarts, the container will run automatically.
- Azure Container Registry information from where the image will be pulled, including the registry login server, password, and username:

```
C:\Users\citynextadmin>az container create --resource-group rg03 --cpu 4 --dns-name-label mysamplewebappcontainer --image sampleappbookacr.azurecr.io/samplewebapp:latest --ip-address Public --location "West Europe" --memory 7 --name mysamplewebappcontainer --os-type Windows --protocol tcp --ports 8 --restart-policy Always --registry-login-server sampleappbookacr.azurecr.io --registry-password "lgzgtuxlehn2/92KvNk==X4UtbIuQim" --registry-username [REDACTED]
```

The command should run successfully to create a new container. This would create a new container on the Azure Container Instance. The portal configuration is as follows:

Setting	Value
OS type	Windows
IP address	51.145.176.253
FQDN	mysamplewebappcontainer.westeurope.azurecontainer.io
Container count	1

The container configuration is as follows:

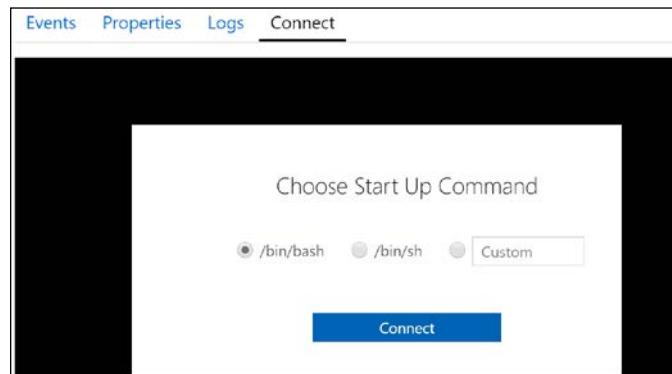
NAME	IMAGE	STATE	START TIME	RESTART COUNT
mysamplewebappcontainer	samplewebappbookacr.azurecr.io...	Running	2019-01-24T09:07:20Z	0

NAME	TYPE	FIRST TIMESTAMP	LAST TIMESTAMP	MESSAGE	COUNT
Started	Normal	1/24/2019, 4:07 AM EST	1/24/2019, 4:07 AM EST	Started container with dock...	1
Pulled	Normal	1/24/2019, 4:07 AM EST	1/24/2019, 4:07 AM EST	Successfully pulled image *s...	1
Created	Normal	1/24/2019, 4:07 AM EST	1/24/2019, 4:07 AM EST	Created container with dock...	1
Pulling	Normal	1/24/2019, 4:06 AM EST	1/24/2019, 4:06 AM EST	pulling image *sampleweb...	1

The properties are the same as we provided when creating the container instance:

Properties	
Name	mysamplewebappcontainer
Image	samplewebappbookacr.azurecr.io/samplewebapp:latest
Ports	80
Memory	7 GB
Cores	4
Commands	(no commands)
Environment variables	

It is also possible to connect to the container using the **Connect** tab:



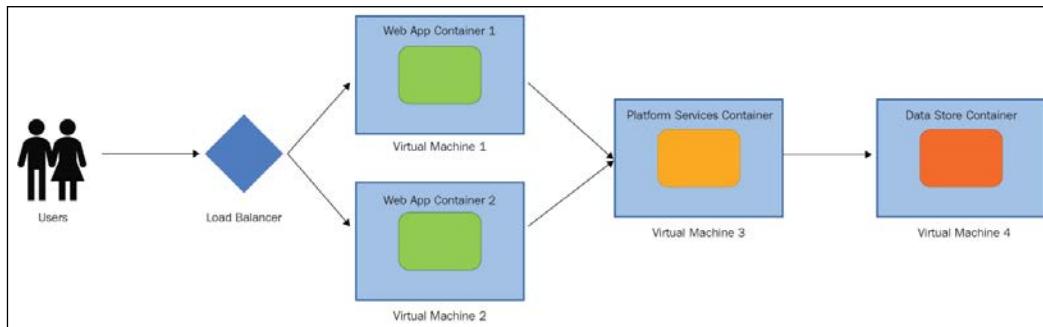
The final page output is shown in the following screenshot:

The screenshot shows a web browser window with the URL mysamplewebappcontainer.westeurope.azurecontainer.io. The page has a purple header with the Visual Studio logo and icons for HTML5, CSS3, and JS. Below the header, there's a message about new features for building modern web apps. The main content area is divided into four sections: Application uses, How to, Overview, and Run & Deploy. The Application uses section lists items like Sample pages using ASP.NET Core MVC and Theming using Bootstrap. The How to section lists steps such as Add a Controller and View, Manage User Secrets using Secret Manager, and Use logging to log a message. The Overview section provides a conceptual overview of ASP.NET Core. The Run & Deploy section lists tasks like Run your app, Run tools such as EF migrations and more, and Publish to Microsoft Azure Web Apps. At the bottom, there's a footer with the text "© 2019 - samplewebapp".

Azure Kubernetes Service

Azure Kubernetes Service is one of the leading managed container environments in the world. Before getting into Azure Kubernetes Service, let's understand what Kubernetes is and why it is required.

Kubernetes is a container orchestration engine and service. Orchestration of containers is a must for any enterprise solution deployed on top of containers. Imagine that there are four containers for a solution, which are hosted on four different virtual machines, as shown in the following diagram:



Containers, by their very nature, are quite fleeting. They can be created easily and quite quickly, and can be torn down just as easily. For example, if virtual machine three is restarted, the container running on it will also be destroyed. If any container, or any host hosting a container, goes down for any reason, the whole application and solution also goes down with it. This is because there is no active monitoring of these containers and hosts.

Although traditional monitoring systems can be used to monitor the hosts, there is no single solution that manages and monitors the hosts and containers together. Moreover, the problem is not only related to availability. The health of the hosts and containers can also have a significant impact for the functioning of the application. Even if one container is not able to perform as expected, the entire solution will have a downgraded performance.

Container orchestrators solve these aforementioned problems with containers. The orchestrators take away the burden of active monitoring and management from users and automate the entire set of activities and its impact. An orchestrator can identify whether any host or any container is suffering from degraded performance. It can immediately spin up new containers on different hosts. Not only this, but even if one of the containers goes down, a new container instance can be created on another host. This does not stop here. If a complete host goes down, all the containers running on that host can be recreated on a different host. These are all the responsibility of container orchestrators.

A container orchestrator should be told the desired and expected amount of each type of container. For our example, the desired amount of web application containers is two, for platform services it's one, and for the data store it's one. If this information is fed into an orchestrator, it will always try to maintain the environment to have the desired amount of containers. If containers go down in any host, the orchestrator will notice that and create a new container on another host to compensate.

Kubernetes is a container orchestrator, and it provides all of these features.

Kubernetes can be deployed on virtual machines directly via IaaS deployment, and at the same time it can be provisioned as Azure Kubernetes Service. The difference between the two different deployment paradigms is that IaaS deployment is controlled and managed by the user, while Azure Kubernetes Service is a managed service and all hosts and containers are managed by Azure without any effort from the user deploying it. Managing an IaaS-based Kubernetes deployment is not an easy task. Only people with in-depth knowledge of Kubernetes are able to manage such an environment, and it is not easy to find such professionals. Moreover, the overall deployment of Kubernetes and its management is a complex and cumbersome exercise that few organizations are willing to undertake. For such reasons, Azure Kubernetes Service is popular among organizations that are outsourcing the entire management of Kubernetes to Azure.

Kubernetes, being based on orchestrator engines, works seamlessly with the Docker runtime. It can, however, even work with other container runtimes and network providers.

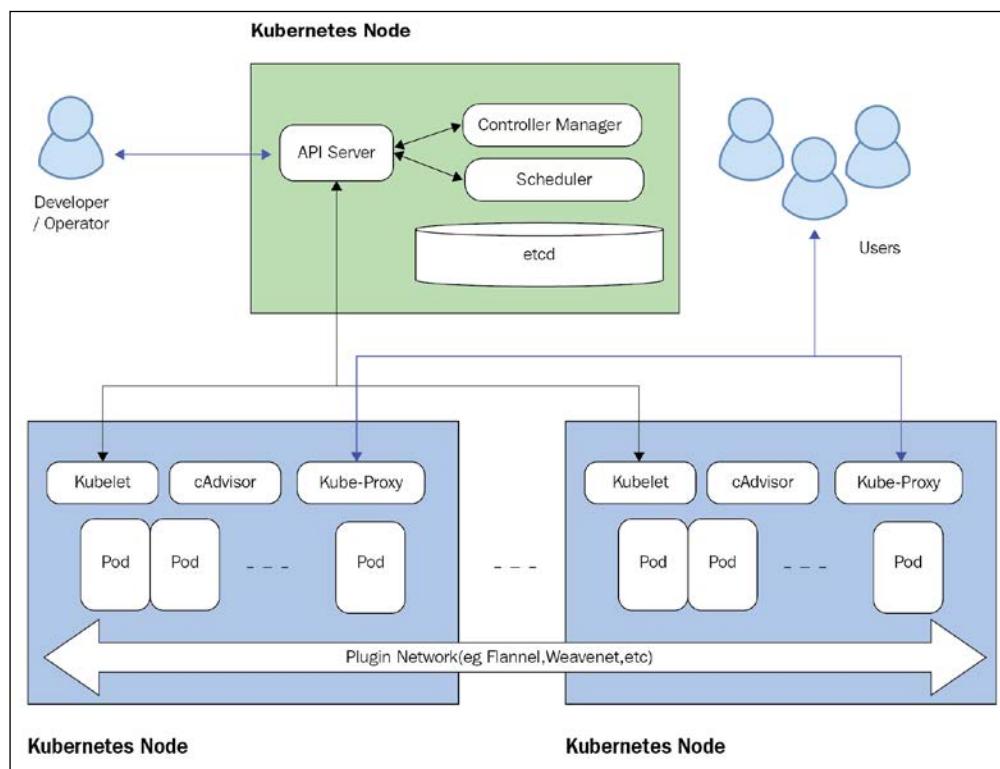
Kubernetes architecture

Kubernetes comprises a set of servers, and these servers are together called clusters. Kubernetes has two types of nodes:

- Master nodes
- Pod nodes

Master nodes

Master nodes contain all the management components of Kubernetes, such as the **API server**, **scheduler**, distributed configuration services, such as etcd, and the replication controller, while the pod nodes contain the **Kube-Proxy** and **Kubelet** components. The containers are deployed on pod nodes:



Pods

The first component to understand in Kubernetes is the concept of pods. Pods are collections of containers that are deployed, created, and managed as a single unit. Containers are not the smallest unit of management in Kubernetes; those would be pods.

Pods are created either using Kubernetes-provided CLI tools, such as Kubectl, or using YAML files. These YAML files contain the pod metadata and configuration information. When these YAML files are submitted to Kubernetes, the main Kubernetes engine schedules the creation of pods on pod nodes.

API server

The API server is the brain for the entire Kubernetes service. All requests from users actually get submitted to the API server. The API server is a REST-based endpoint that accepts requests and acts on them. For example, requests to create pods are submitted to the API server. The API server stores this information and informs another component, known as a scheduler, to provision pods based on the YAML files. The health information of the pods and nodes is also sent to the API server by Kubelets from pod nodes.

Kubelets

Kubelets are components that are provisioned on pod nodes. Their job is to monitor the pod nodes and the pods on those nodes and inform the API server about their health. Kubelets are also responsible for configuring the Kube-Proxy component, which plays a crucial role in request flow among pods and nodes arising from both external and internal sources.

Kube-Proxy

Kube-Proxy is responsible for maintaining routes on the pod nodes. It is also responsible for changing the data packet headers with the appropriate **Source Network Address Translation (SNAT)** and **Destination Network Address Translation (DNAT)** information and enables the flow of traffic among pods and nodes within the cluster.

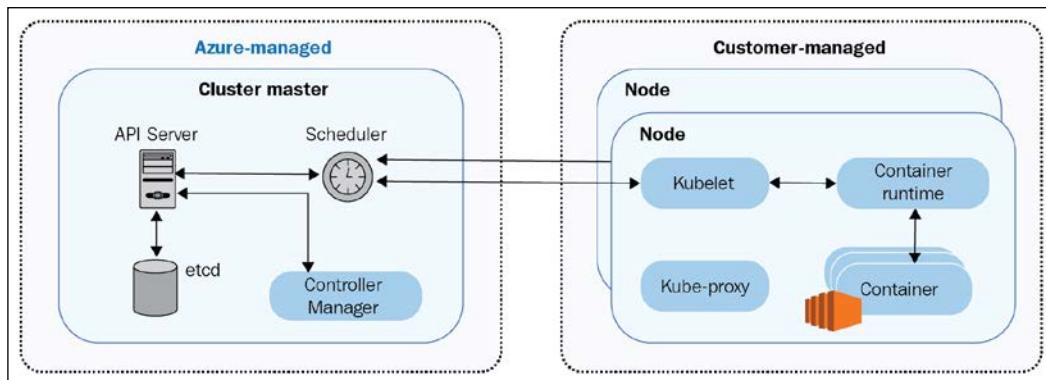
Replication controller/controller manager

The replication controller is responsible for maintaining the desired number of pod instances. If pods or nodes go down, the Kubelet informs the API server and the API server informs the replication controller. The controller will immediately schedule the creation of new pods on the same or different nodes, based on the current health of the node.

Azure Kubernetes architecture

When we provision Azure Kubernetes Service, we get an environment consisting of an entire Kubernetes cluster comprising both master and pod nodes. These pod nodes already have the Docker runtime and network plugins installed and configured.

The master has all the components installed and configured. The master is managed by Azure Kubernetes Service while the nodes are managed by the customers. This is shown in the following diagram:



Provisioning Azure Kubernetes Service

In this section, we will provision Azure Kubernetes Service using Azure CLI. Azure Kubernetes Service can also be provisioned from the Azure portal and Azure PowerShell:

1. Log into Azure using the following command:

```
az login
```

2. Select an appropriate subscription using the following command:

```
az account set -subscription <<xxxxxxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxx>>
```

3. The Azure Kubernetes Service features are available from the `az aks` command.

4. Create a new resource group to host Azure Kubernetes Service using the following command:

```
az group create --name "akdemo" --location "west Europe"
```

5. Once the resource group is created, we can create the Kubernetes cluster using the following command:

```
az aks create --resource-group akdemo --name AKSPortalCluster  
--node-count 2 --enable-addons monitoring --generate-ssh-keys
```

6. This command takes a while to create the Kubernetes cluster.
7. After the cluster is provisioned, it is possible to download the Kubectl CLI tool using the Azure CLI and configure it to work without the cluster.
8. The command to download the Kubectl tool is `az aks install-cli`:

```
C:\Users\citynextadmin>az aks install-cli  
Downloading client to "C:\Users\citynextadmin\.azure-kubectl\kubectl.exe" from "https://storage.googleapis.com/kubernetes-release/release/v1.13.2/bin/windows/amd64/kubectl.exe"  
Please add "C:\Users\citynextadmin\.azure-kubectl" to your search PATH so the 'kubectl.exe' can be found. 2 options:  
  1. Run "Set PATH=%PATH%;C:\Users\citynextadmin\.azure-kubectl" or "$env:path += 'C:\Users\citynextadmin\.azure-kubectl'" for PowerShell. This is good for the current command session.  
  2. Update system PATH environment variable by following "Control Panel->System->Advanced->Environment Variables", and re-open the command window. You only need to do it once
```

It is important to note that the Kubectl tool should be in the search path to execute it on the command line by its name. If it's not in the path, then the entire path of the Kubectl tool should be provided every time it needs to be executed. An ideal approach is to add it to the search path using the following command:

```
set PATH=%PATH%;C:\Users\citynextadmin\.azure-kubectl
```

After executing this command, it is possible to execute the `kubectl` command:

```
C:\Users\citynextadmin>kubectl  
kubectl controls the Kubernetes cluster manager.  
  
Find more information at: https://kubernetes.io/docs/reference/kubectl/overview/  
  
Basic Commands (Beginner):  
  create      Create a resource from a file or from stdin.  
  expose     Take a replication controller, service, deployment or pod and expose it as a new Kubernetes Service  
  run        Run a particular image on the cluster  
  set        Set specific features on objects  
  
Basic Commands (Intermediate):  
  explain    Documentation of resources  
  get        Display one or many resources  
  edit       Edit a resource on the server  
  delete    Delete resources by filenames, stdin, resources and names, or by resources and label selector
```

At this stage, the Kubectl tool is able to be executed, but it is still not connected to our recently provisioned cluster. Kubectl needs credentials configured so that it can be authenticated by the cluster. Please note that we did not provide any credential information when creating the cluster, and the credentials were generated by Azure Kubernetes Service when provisioning the cluster.

These credentials can be downloaded, and Kubectl can be configured using the command provided by Azure Kubernetes Service on the Azure CLI:

```
C:\Users\citynextadmin>az aks get-credentials --resource-group akdemo --name AKSPortalCluster  
Merged "AKSPortalCluster" as current context in C:\Users\citynextadmin\.kube\config
```

The next step is to create pods on the cluster. For this, we will create a new YAML file and submit it to the API server using the commands available in the Kubectl CLI tool.

The YAML file should be saved on a local machine so that it can be referenced from the command line. It is not possible to explain the YAML file in this book, but there are other books that cover YAML and Kubernetes configurations in detail. The sample YAML is listed here:

```
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: azure-vote-back  
spec:  
  replicas: 1  
  selector:  
    matchLabels:  
      app: azure-vote-back  
  template:  
    metadata:  
      labels:  
        app: azure-vote-back  
    spec:  
      containers:  
      - name: azure-vote-back  
        image: redis  
        resources:  
          requests:  
            cpu: 100m  
            memory: 128Mi  
          limits:  
            cpu: 250m  
            memory: 256Mi  
        ports:  
        - containerPort: 6379  
          name: redis  
---  
apiVersion: v1
```

```
kind: Service
metadata:
  name: azure-vote-back
spec:
  ports:
  - port: 6379
  selector:
    app: azure-vote-back
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-front
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-front
  template:
    metadata:
      labels:
        app: azure-vote-front
    spec:
      containers:
      - name: azure-vote-front
        image: microsoft/azure-vote-front:v1
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 250m
            memory: 256Mi
        ports:
        - containerPort: 80
        env:
        - name: REDIS
          value: "azure-vote-back"
---
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-front
spec:
```

```
type: LoadBalancer
ports:
- port: 80
selector:
  app: azure-vote-front
```

The file was saved as `aksdemo.yaml`. This YAML file is responsible for creating two types of pod. Each of these types should be used to create a single instance. In short, there should be two pods created, one for each type. Both of the pod specifications are available in the YAML file within the `template` section.

The YAML file can be submitted using the `apply` command:

```
C:\Users\citynextadmin>kubectl apply -f aksdemo.yaml
deployment.apps/azure-vote-back created
service/azure-vote-back created
deployment.apps/azure-vote-front created
service/azure-vote-front created
```

At this stage, nodes can be queried as follows:

```
C:\Users\citynextadmin>kubectl get nodes
NAME           STATUS   ROLES   AGE   VERSION
aks-nodepool1-39448715-0   Ready    agent   20m   v1.9.11
aks-nodepool1-39448715-1   Ready    agent   20m   v1.9.11
```

Pods can be queried as shown in the following screenshot:

```
C:\Users\citynextadmin>kubectl get pods
NAME                           READY   STATUS            RESTARTS   AGE
azure-vote-back-f7c88f6f5-2zxp7   1/1    Running          0          67s
azure-vote-front-7c79dddc4c-f99hl  0/1    ContainerCreating   0          67s
```

Once the pods are created, they should show a `Running` status. However, it's not only the pods that need to be created. There are more resources to create on the cluster, as shown in the next screenshot.

This includes two services. One service, of the `ClusterIP` type, is used for internal communication between pods, and the other one, of the `LoadBalancer` type, is used for enabling external requests to reach the pod. The configuration of services is available in the YAML file as `kind: Service`. There are two deployments as well; deployments are resources that manage the deployment process. The configuration of services is available in the YAML file as `kind: Deployment`.

Two replica sets are also created, one for each pod type. Replica sets were previously known as replication controllers, and they ensure that the desired number of pod instances are always maintained within the cluster. The configuration of services is available in the YAML file as the `spec` element:

```
C:\Users\citynextadmin>kubectl get all
NAME                                         READY   STATUS    RESTARTS   AGE
pod/azure-vote-back-f7c88f6f5-2zxp7        1/1     Running   0          4m11s
pod/azure-vote-front-7c79dddc4c-f99h1      1/1     Running   0          4m11s

NAME                TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)        AGE
service/azure-vote-back  ClusterIP   10.0.180.11   <none>           6379/TCP      4m11s
service/azure-vote-front LoadBalancer  10.0.56.53    52.236.179.79  80:32110/TCP  4m11s
service/kubernetes       ClusterIP   10.0.0.1       <none>           443/TCP       27m

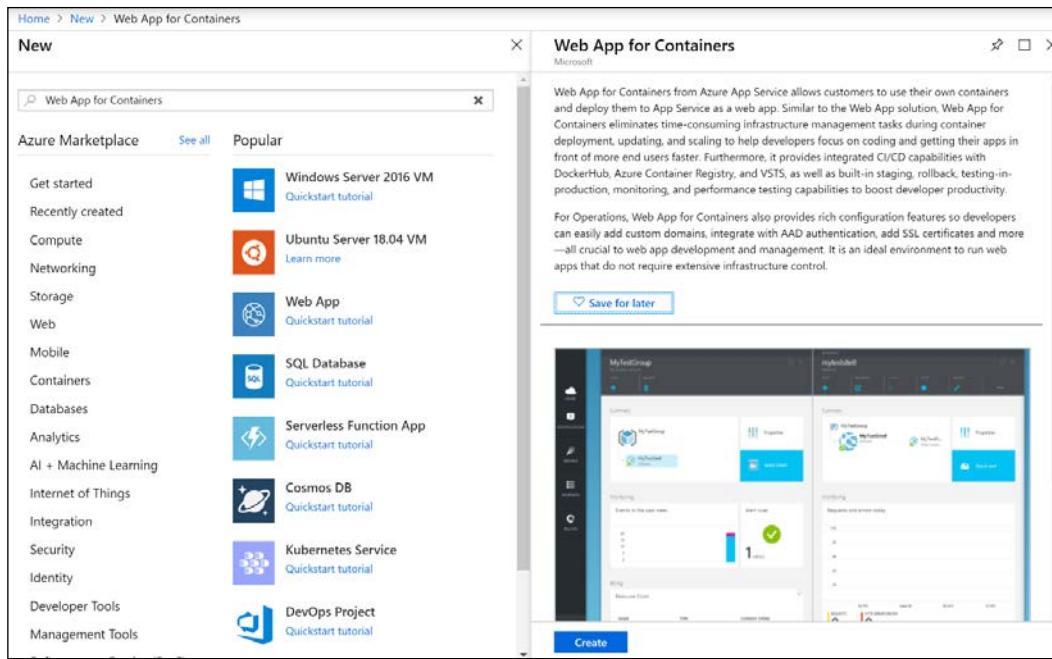
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/azure-vote-back  1/1     1           1           4m11s
deployment.extensions/azure-vote-front 1/1     1           1           4m11s

NAME                         DESIRED   CURRENT   READY   AGE
replicaset.extensions/azure-vote-back-f7c88f6f5  1         1         1         4m11s
replicaset.extensions/azure-vote-front-7c79dddc4c  1         1         1         4m11s
```

App Service containers

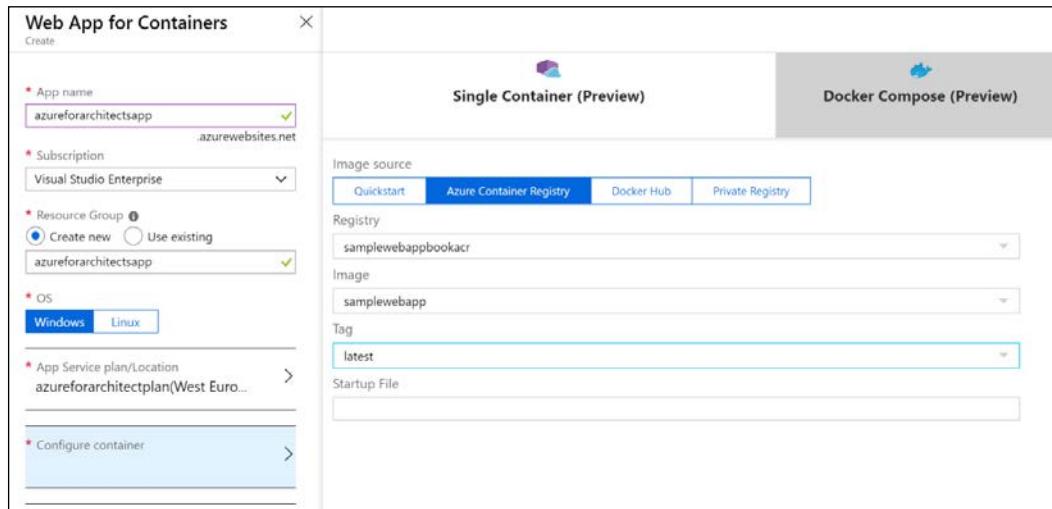
Previously, it was not possible to use containers along with Azure App Service. App Service provided the ability to host applications on virtual machines. There is now a new preview feature that has been recently added to Azure App Service. This feature allows you to host containers on App Service and have containers contain the application binaries and dependencies together:

1. To deploy a container on App Service, go to the Azure portal and select **Web App for Containers** and create one as follows:

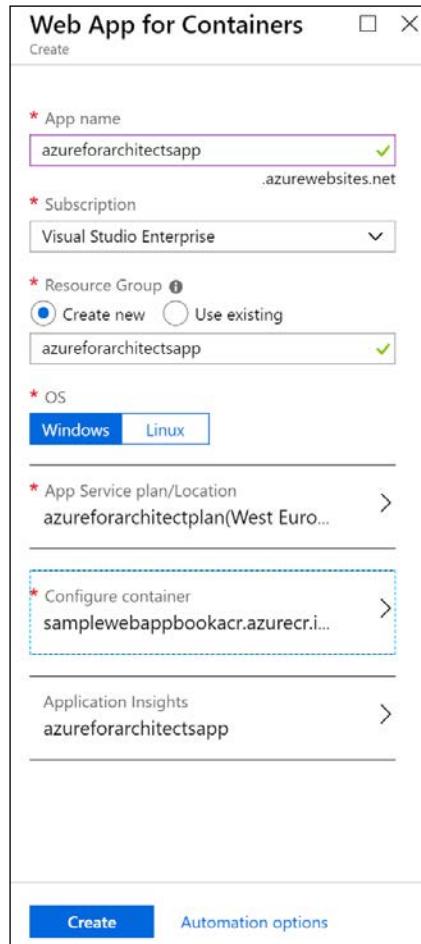


2. Provide a name for it, along with subscription and resource group information. Select an appropriate operating system. Both Linux and Windows are available as options.
3. Either create a new **App Service plan**, as shown in the following screenshot, or use an existing plan:

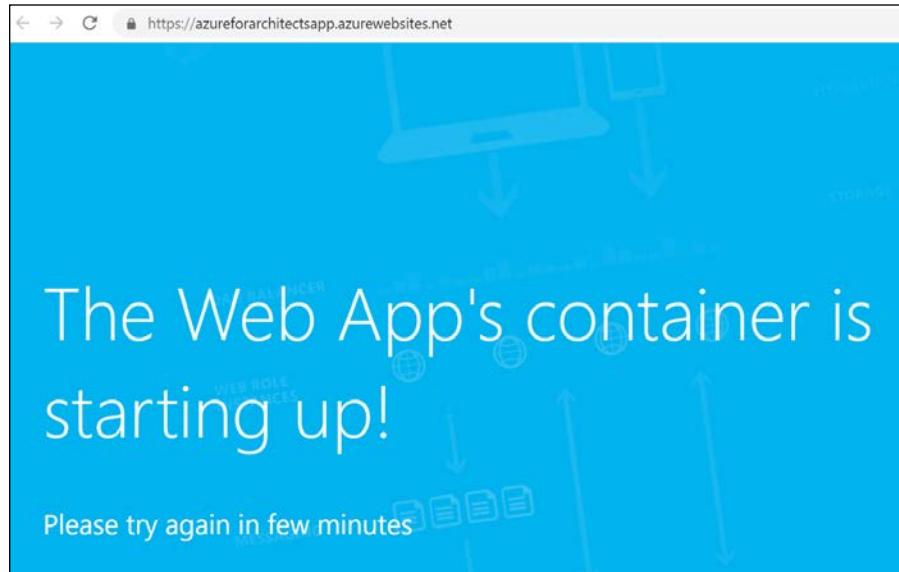
4. Provide the **Configure container** configuration information. Here, we already have our image on Azure Container Registry, which has been chosen for this example. The other choices are Docker Hub and private registries. Provide the information about the registry hosting the image. Since we've already uploaded an image to Docker Hub, it is possible to use the Docker Hub image instead of Azure Container Registry:



5. The complete configuration for **Web App for Containers** should be similar to that shown in the following screenshot. Creating the App Service will download the image from the container registry and create a new container out of it:



6. It takes a while for the container to be up and running. Meanwhile, if you navigate to the URL generated for App Service, it will show the following page:



7. If you navigate to the **Container settings** menu on the left, it will show the diagnostic logs for the container:

A screenshot of the Azure portal interface, specifically the "Container settings" page for an app service named "azureforarchitectsapp". The left sidebar shows navigation options like Home, Deployment, Settings, and Application Insights. The "Container settings" option is selected and highlighted in blue. On the right, there are sections for "Deployment" (with fields for "Image and optional tag" set to "azureforarchitects/samplewebapp:latest" and "Startup File" empty), "Continuous Deployment" (with a toggle switch set to "On"), and "Logs". The "Logs" section displays two entries of diagnostic log output. The first entry is from "31/12/2018 10:09:57.075 INFO - Site: azureforarchitectsapp - Image: azureforarchitects/samplewebapp:latest" and the second is from "31/12/2018 10:09:57.420 INFO - Site: azureforarchitectsapp - Image: azureforarchitects/samplewebapp:latest". Both logs mention a "Custom Registry" at "https://index.docker.io" and a "Status: 6f2071ddc729 Extracting". At the bottom of the logs, there are "Download" and "Refresh" buttons, along with "Save" and "Discard" buttons for making changes.

Comparing all container options

It's time to compare all container hosting options in Azure. This is one of the most crucial skills that architects should possess to decide the optimal resource for their requirements. The options for hosting containers are as follows:

- Containers on virtual machines
- Containers on virtual machines with Kubernetes as the orchestrator
- Azure Kubernetes Service
- Containers in Azure App Service
- Containers in Azure Container Instances
- Containers in Azure Functions
- Containers in Service Fabric

Given the large number of options for hosting containers, it is important to understand their nature, including their cost and benefits, before finalizing a resource or group of resources. In this section, we will evaluate and compare resources based on the following:

- Cost
- Complexity
- Control
- Agility and flexibility
- Scalability and availability

Containers on virtual machines

When deploying containers on virtual machines without an orchestrator, the entire burden of ensuring that the containers are running is on the customer. The customer may have to purchase additional monitoring services that can monitor hosts as well as containers. This, however, is also not an ideal situation. This is because, while monitoring can generate alerts and inform the administrators, the customer must spend more to create automation on top of those alerts.

This option is also costly compared to others, as the cost will involve the cost of the virtual machines, along with the cost of other Azure resources, such as Azure Storage and Load Balancer.

For production environments and mission-critical application, this option is not advisable.

Containers on virtual machines with Kubernetes as the orchestrator

When deploying containers on virtual machines with an orchestrator, the cost will be higher compared to that without an orchestrator because of the need for additional virtual machines to act as management servers hosting the API server, schedulers, and controllers. If the master is highly available, then additional virtual machines will be required. The benefit of this solution compared to previous solutions is that the applications will be continually available without downtime, as Kubernetes will ensure that if a host or pod goes down, the pods will be created on other hosts.

This option is ideal when an organization wants full control over the environment and solution and has skilled people to manage the environment. The option also helps with rolling upgrades for the application. The scaling of hosts will be a manual activity, which could make operations complex and time-consuming unless virtual machine scale sets with scaling rules are used for scaling.

Azure Kubernetes Service

Azure Kubernetes Service is a **Platform as a Service (PaaS)**. This means that when deploying Azure Kubernetes Service, customers do not generally access the master and agent nodes directly; however, if required, they can log onto agent nodes for troubleshooting purposes. The only access needed by customers is to the Kubectl CLI tool, which interacts with the API server. Customers of Azure Kubernetes Service do not have to manage the nodes, and Azure will ensure that all upgrades, as well as planned and unplanned maintenance, are taken care of automatically. Customers can deploy their pods on the cluster and can rest assured that the desired state of the container environment will be maintained by Azure Kubernetes Service.

This solution is not complex compared to the previous two solutions and does not require highly-skilled Kubernetes operators. The solution is highly scalable as Azure Kubernetes Service provides the means by which you can increase the number of nodes on demand. It is also highly available, as Azure Kubernetes Service ensures that, if a node goes down, another node is made available to the cluster.

The disadvantage of this solution is that customers lose some control over the underlying infrastructure. Customers have control over the Azure network, Kubernetes networking stack, and agent nodes; however, they do not have access to master nodes. Master nodes are hidden from customers. An important thing to note about this solution is that it works well with Linux-based images; however, there are additional tasks that would need to be performed in the case of Windows. Also, the master nodes are always Linux-based. These are limitations that will definitely change in future as Microsoft makes Windows completely native to containers.

This is one of the most highly-recommended options and solutions out of all the available options.

Containers on Azure App Service

This is a relatively new feature provided by Azure and is currently in preview at the time of writing. Azure App Service is another PaaS that provides a managed environment for deploying applications. App Service can be automatically scaled up and out based on rules and conditions, and containers can be created on these new nodes. Web Apps for Containers provides all the benefits of Azure App Service with an additional benefit. When provisioning an App Service, customers are bound by the given platform in terms of .NET version and operating system. Customers cannot control the operating system version or the .NET version. However, when deploying containers on top of App Service, the container images can consist of any .NET runtime, framework, and SDK. Moreover, the container image can be based on any compatible version of the base operating system. In short, containers on App Service provide much more flexibility for provisioning an environment. It is also cost-effective compared to IaaS container options; however, there is reduced control of virtual machines. This option, however, provides keyhole access to the underlying services and servers.

This option necessitates that the images are stored on some registry. It could be Azure Container Registry, Docker Hub, or any private registry. It is not possible to upload a Dockerfile targeting Windows images to App Service; however, this option of using Docker Compose and Kubernetes is available for Linux containers.

The complexity of creating and maintaining applications with Web Apps for Containers is considerably lower compared to other options. It's a great solution for demo and development/test environments.

Containers in Azure Container Instances

This, again, is a relatively new feature in Azure, and helps with hosting containers. It helps to quickly create a container and host it without the need for any virtual machines. One of the major complexities of this solution is that multi-container solutions need more attention, as this service treats every container as a standalone solution. It can download images from Azure Container Registry.

Containers in Azure Functions

This resource is specific to running Azure Functions within containers. If customers are writing and deploying functions and want to host in any environment that is different than the out-of-the-box environment provided by Azure, then this is a good solution. Microsoft already provided a few images hosted in Docker Hub related to functions. These images already contain the Azure Functions runtime and framework. It is important to note that images not containing the Azure Functions runtime should not be used for hosting containers with functions.

The benefits and constraints for this solution are similar to those of containers with web apps.

Containers in Service Fabric

Service Fabric is another Azure resource on which containers can be hosted. Service Fabric can be hosted on premises, on Azure virtual machines, and as a managed service by providing a managed platform.

The advantages and disadvantages are similar to those of the IaaS and PaaS solutions mentioned earlier in this chapter.

Summary

This chapter was focused on containers in Azure. Azure provides multiple services that are either native or compatible with containers. These services are relatively new features and newer capabilities are being added to them on an ongoing basis. Some of the important Azure services related to containers are Azure Container Registry, Azure Container Instances, Azure Kubernetes Service, and Azure App Service. There are other services, such as Azure Service Fabric, which supports hosting containers, and function app containers, which support functions being run in containers. Containers can be hosted either as IaaS or PaaS. Both have their advantages and disadvantages, and organizations should evaluate their requirements and choose an appropriate service for their containers. It is important to know which container service to use in different scenarios, as that would be a key factor in the success of an architect in Azure.

11

Azure DevOps

Software development is a complex undertaking comprising multiple processes and tools, and involving people from different departments. They all need to come together and work in a cohesive manner. With so many variables, the risks are high when you are delivering to the end customers. One small omission or misconfiguration, and the application might come crashing down. This chapter is about adopting and implementing practices that reduce this risk considerably and ensure that high-quality software can be delivered to the customer again and again.

Before getting into the details of DevOps, let's list the problems faced by software companies that DevOps addresses:

- Organizations are rigid and don't welcome change
- Rigid and time-consuming processes
- Isolated teams working in silos
- Monolithic design and big bang deployments
- Manual execution
- Lack of innovation

In this chapter, we will cover the following topics:

- DevOps
- DevOps practices
- Azure DevOps
- DevOps preparation
- DevOps for PaaS solutions
- DevOps for virtual machine-based (IaaS) solutions
- DevOps for container-based (IaaS) solutions

- Azure DevOps and Jenkins
- Azure Automation
- Azure tools for DevOps

DevOps

There's currently no industry-wide consensus regarding the definition of DevOps. Organizations have formulated their own definition of DevOps and tried to implement it. They have their own perspective and think they've implemented DevOps if they implement automation and configuration management, and use Agile processes.

DevOps is about the delivery mechanism of software systems. It's about bringing people together, making them collaborate and communicate, working together towards a common goal and vision. It's about taking joint responsibility, accountability, and ownership. It's about implementing processes that foster collaboration and a service mindset. It enables delivery mechanisms that bring agility and flexibility to the organization. Contrary to popular belief, DevOps isn't about tools, technology, and automation. These are enablers that help with collaboration, the implementation of Agile processes, and faster and better delivery to the customer.

There are multiple definitions available on the internet for DevOps, and they aren't wrong. DevOps doesn't provide a framework or methodology. It's a set of principles and practices that, when employed within an organization, engagement, or project, achieve the goal and vision of both DevOps and the organization. These principles and practices don't mandate any specific process, tools and technologies, or environments. DevOps provides guidance that can be implemented through any tool, technology, or process, although some of the technology and processes might be more applicable than others to achieve the vision of DevOps' principles and practices.

Although DevOps practices can be implemented in any organization that provides services and products to customers, going forward in this book, we'll look at DevOps from the perspective of software development and the operations department of any organization.

So, what is DevOps? DevOps is defined as a set of principles and practices bringing all teams, including developers and operations, together from the start of the software system for faster, quicker, and more efficient end-to-end delivery of value to the end customer, again and again in a consistent and predictable manner, reducing time to market, thereby gaining a competitive advantage.

Read the preceding definition of DevOps out loud; if you look at it closely, it doesn't indicate or refer to any specific processes, tools, or technology. It doesn't prescribe any methodology or environment.

The goal of implementing DevOps principles and practices in any organization is to ensure that the demands of stakeholders (including customers) and expectations are met efficiently and effectively.

The customer's demands and expectations are met when the following happens:

- The customer gets the features they want.
- The customer gets the features when they want.
- The customer gets faster updates on features.
- The quality of delivery is high.

When an organization can meet these expectations, customers are happy and remain loyal. This, in turn, increases the market competitiveness of the organization, which results in a bigger brand and market valuation. It has a direct impact on the top and bottom lines of the organization. The organization can invest further in innovation and customer feedback, bringing about continuous changes to its system and services in order to stay relevant.

The implementation of DevOps principles and practices in any organization is guided by its surrounding ecosystem. This ecosystem is made up of the industry and domains the organization belongs to.

DevOps is based on a set of principles and practices. We'll look into the details of these principles and practices later in this chapter. The core principles of DevOps are the following:

- Agility
- Automation
- Collaboration
- Feedback

The core DevOps practices are the following:

- Continuous integration
- Configuration management
- Continuous deployment
- Continuous delivery
- Continuous learning

DevOps is not a new paradigm; however, it's gaining a lot of popularity and traction in recent times. Its adoption is at its highest level, and more and more companies are undertaking this journey. I purposely mentioned DevOps as a journey because there are different levels of maturity within DevOps. While successfully implementing continuous deployment and delivery are considered the highest level of maturity in this journey, adopting source code control and Agile software development are considered the first step in the DevOps journey.

One of the first things DevOps talks about is breaking down the barriers between the development and the operations teams. It brings about close collaboration between multiple teams. It's about breaking the mindset that the developer is responsible for writing the code only and passing it on to operations for deployment once it's tested. It's also about breaking the mindset that operations have no role to play in development activities. Operations should influence the planning of the product and should be aware of the features coming up as releases. They should also continually provide feedback to the developers on the operational issues such that they can be fixed in subsequent releases. They should influence the design of the system to improve the operational working of the system. Similarly, the developers should help the operations team to deploy the system and solve incidents when they arise.

The definition of DevOps talks about faster and more efficient end-to-end delivery of systems to stakeholders. It doesn't talk about how fast or efficient the delivery should be. It should be fast enough for the organization's domain, industry, customer segmentation, and needs. For some organizations, quarterly releases are good enough, while for others it could be weekly. Both are valid from a DevOps point of view, and these organizations can deploy relevant processes and technologies to achieve their target release deadlines. DevOps doesn't mandate any specific time frame for CI/CD. Organizations should identify the best implementation of DevOps principles and practices based on their overall project, engagement, and organizational vision.

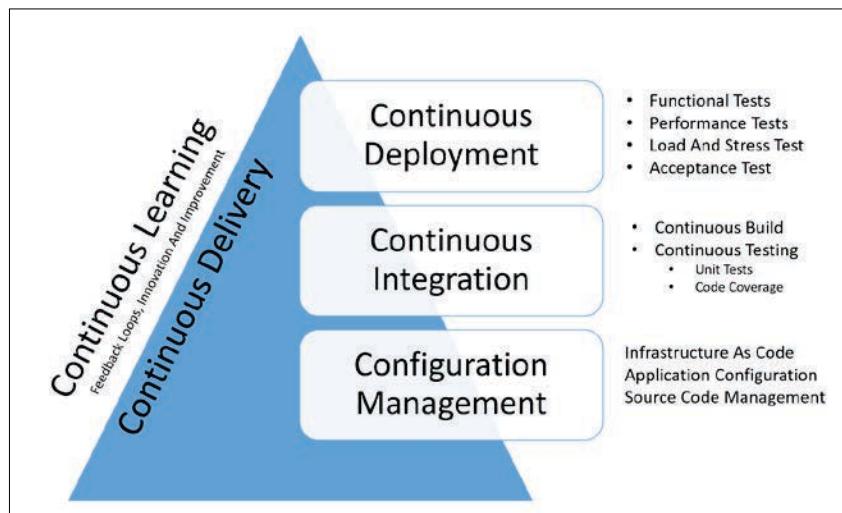
The definition also talks about end-to-end delivery. This means that everything from the planning and delivery of the system through to the services and operations should be part of the DevOps adoption. The processes should be such that it allows for greater flexibility, modularity, and agility in the application the development life cycle. While organizations are free to use the best fit process – Waterfall, Agile, Scrum, and more. Typically, organizations tend to favor Agile processes with iterations-based delivery. This allows for faster delivery in smaller units, which are far more testable and manageable, compared to a large delivery.

DevOps repeatedly talks about end customers in a consistent and predictable manner. This means that organizations should continually deliver to customers with newer and upgraded features using automation. We can't achieve consistency and predictability without the use of automation. Manual work should be nonexistent to ensure a high level of consistency and predictability. The automation should also be end-to-end, to avoid failures. This also indicates that the system design should be modular, allowing faster delivery on systems that are reliable, available, and scalable. Testing plays a big role in consistent and predictable delivery.

The end result of implementing these practices and principles is that the organization is able to meet the expectations and demands of customers. The organization is able to grow faster than the competition, and further increase the quality and capability of their product and services through continuous innovation and improvement.

DevOps practices

DevOps consists of multiple practices, each providing a distinct functionality to the overall process. The following diagram shows the relationship between them. Configuration management, continuous integration, and continuous deployment form the core practices that enable DevOps. When we deliver software services that combine these three services, we achieve continuous delivery. Continuous delivery is a capability and level of maturity of an organization that's dependent on the maturity of configuration management, continuous integration, and continuous deployment. Continuous feedback at all stages forms the feedback loop that helps to provide superior services to customers. It runs across all DevOps practices. Let's deep dive into each of these capabilities and DevOps practices:



Configuration management

Business applications and services need an environment on which they can be deployed. Typically, the environment is an infrastructure composed of multiple servers, computers, network, storage, containers, and many more services working together such that business applications can be deployed on top of them. Business applications are decomposed into multiple services running on multiple servers, either on-premise or on the cloud, and each service has its own configuration along with requirements related to the infrastructure's configuration. In short, both the infrastructure and the application are needed to deliver systems to customers, and both of them have their own configuration. If the configuration drifts, the application might not work as expected, leading to downtime and failure. Moreover, as the ALM process dictates the use of multiple stages and environments, an application would be deployed to multiple environments with different configurations. The application will be deployed to the development environment for developers to see the result of their work. The application will be deployed to multiple test environments with different configurations for functional tests, load and stress tests, performance tests, integration tests, and more; it would also be deployed to the preproduction environment to conduct user-acceptance tests, and finally onto the production environment. It's important that an application can be deployed to multiple environments without undertaking any manual changes to its configuration.

Configuration management provides a set of processes and tools and they help to ensure that each environment and application gets its own configuration.

Configuration management tracks configuration items, and anything that changes from environment to environment should be treated as a configuration item. Configuration management also defines the relationships between the configuration items and how changes in one configuration item will impact the other configuration items.

Configuration management helps in the following places:

- **Infrastructure as Code:** When the process of provisioning infrastructure and its configuration is represented through code, and the same code goes through the application life cycle process, it's known as **Infrastructure as Code (IaC)**. IaC helps to automate the provisioning and configuration of infrastructure. It also represents the entire infrastructure in code that can be stored in a repository and version-controlled. This allows users to employ the previous environment's configurations when needed. It also enables the provisioning of an environment multiple times in a consistent and predictable manner. All environments provisioned in this way are consistent and equal in all ALM stages.

- **Deploying and configuring the application:** The deployment of an application and its configuration is the next step after provisioning the infrastructure. Examples include deploying a webdeploy package on a server, deploying a SQL server schema and data (bacpac) on another server, and changing the SQL connection string on the web server to represent the appropriate SQL server. Configuration management stores values for the application's configuration for each environment on which it is deployed.

The configuration applied should also be monitored. The expected and desired configuration should be consistently maintained. Any drift from this expected and desired configuration would render the application unavailable. Configuration management is also capable of finding the drift and re-configuring the application and environment to its desired state.

With automated configuration management in place, nobody on the team has to deploy and configure the environments and applications on production. The operations team isn't reliant on the development team or long deployment documentation.

Another aspect of configuration management is source code control. Business applications and services comprise code and other artifacts. Multiple team members work on the same files. The source code should always be up to date and should be accessible by only authenticated team members. The code and other artifacts by themselves are configuration items. Source control helps in collaboration and communication within the team, since everybody is aware of what everyone else is doing and conflicts are resolved at an early stage.

Configuration management can be broadly divided into two categories:

- Inside the virtual machine
- Outside the virtual machine

The tools available for configuration management inside the virtual machine are discussed next.

Desired State Configuration

Desired State Configuration (DSC) is a new configuration-management platform from Microsoft, built as an extension to PowerShell. DSC was originally launched as part of **Windows Management Framework (WMF)** 4.0. It's available as part of WMF 4.0 and 5.0 for all Windows Server operating systems before Windows 2008 R2. WMF 5.1 is available out of the box on Windows Server 2016 and Windows 10.

Chef, Puppet, and Ansible

Apart from DSC, there's a host of configuration-management tools, such as Chef, Puppet, and Ansible, supported by Azure. Details about these tools aren't covered in this book.

The tools available for configuration management outside of a virtual machine are mentioned next.

ARM templates

ARM templates are the primary means of provisioning resources in ARM. ARM templates provide a declarative model through which resources, their configuration, scripts, and extensions are specified. ARM templates are based on **JavaScript Object Notation (JSON)** format. It uses the JSON syntax and conventions to declare and configure resources. JSON files are text-based, user friendly, and easily readable. They can be stored in a source code repository and have version control on them. They are also means to represent infrastructure as code that can be used to provision resources in Azure resource groups again and again, predictably, consistently, and uniformly. A template needs a resource group for deployment. It can only be deployed to a resource group and the resource group should exist before executing template deployment. A template isn't capable of creating a resource group.

Templates provide the flexibility to be generic and modular in their design and implementation. Templates give us the ability to accept parameters from users, declare internal variables, help define dependencies between resources, link resources within the same or different resource groups, and execute other templates. They also provide scripting language-type expressions and functions that make them dynamic and customizable at runtime.

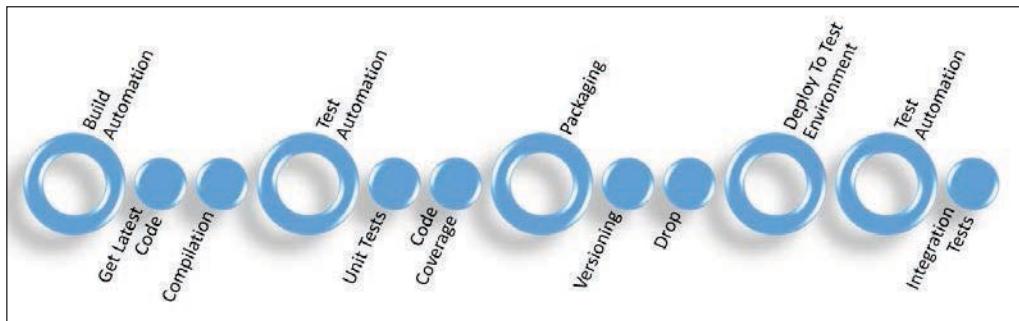
Continuous integration

Multiple developers write code that's eventually stored in a common repository. The code is normally checked in or pushed to the repository when the developer has finished developing their feature. This can happen in a day or might take days or weeks. Some of the developers might be working on the same feature and they might also follow the same practices of pushing/checking-in code in days or weeks. This can create issues with the quality of the code. One of the tenets of DevOps is to fail fast. Developers should check-in/push their code to the repository often and compile the code to check whether they've introduced bugs, and that the code is compatible with the code written by their colleagues. If the developer doesn't follow this practice, the code on their machine will grow too large and will be difficult to integrate with other code. Moreover, if the compile fails, it's difficult and time-consuming to fix the issues that will arise.

Continuous integration solves these kinds of challenges. Continuous integration helps in compiling and validating the code pushed/checked-in by a developer by taking it through a series of validation steps. Continuous integration creates a process flow that consists of multiple steps. Continuous integration is composed of continuous automated builds and continuous automated tests. Normally, the first step is compiling the code. After the successful compilation, each step is responsible for validating the code from a specific perspective. For example, unit tests can be executed on the compiled code, and then code coverage can be executed to check which code paths are executed by unit tests. These could reveal whether comprehensive unit tests are written or whether there's scope to add further unit tests. The end result of continuous integration is deployment packages that can be used by continuous deployment to deploy them to multiple environments.

Developers are encouraged to check in their code multiple times a day, instead of doing so after days or weeks. Continuous integration initiates the execution of the entire pipeline as soon as the code is checked-in or pushed. If compilation succeeds, code tests, and other activities that are part of the pipeline, are executed without error; the code is deployed to a test environment and integration tests are executed on it. Although every system demands its own configuration of continuous integration, a minimal sample continuous integration is shown in the following diagram.

Continuous integration increases developer productivity. They don't have to manually compile their code, run multiple types of tests one after another, and then create packages out of it. It also reduces the risk of getting bugs introduced in the code and the code doesn't get stale. It provides early feedback to the developers about the quality of their code. Overall, the quality of deliverables is high and they are delivered faster by adopting continuous integration practices. A sample continuous integration pipeline is shown here:



Build automation

Build automation consists of multiple tasks executing in sequence. Generally, the first task is responsible for fetching the latest source code from the repository. The source code might comprise multiple projects and files. They are compiled to generate artifacts, such as executables, dynamic link libraries, and assemblies. Successful build automation reflects that there are no compile-time errors in the code.

There could be more steps in build automation, depending on the nature and type of the project.

Test automation

Test automation consists of tasks that are responsible for validating different aspects of code. These tasks are related to testing code from a different perspective and are executed in sequence. Generally, the first step is to run a series of unit tests on the code. Unit testing refers to the process of testing the smallest denomination of a feature by validating its behavior in isolation from other features. It can be automated or manual; however, the preference is toward automated unit testing.

Code coverage is another type of automated testing that can be executed on the code to find out how much of the code is executed when running the unit tests. It's generally represented as a percentage and refers to how much code is testable through unit testing. If the code coverage isn't close to 100%, it's either because the developer hasn't written unit tests for that behavior or the uncovered code isn't required at all.

The successful execution of the test automation, resulting in no significant code failure, should start executing the packaging tasks. There could be more steps in the test automation depending on the nature and type of the project.

Packaging

Packaging refers to the process of generating deployable artifacts, such as MSI, NuGet, webdeploy packages, and database packages; versioning them; and then storing them at a location such that they can be consumed by other pipelines and processes.

Continuous deployment

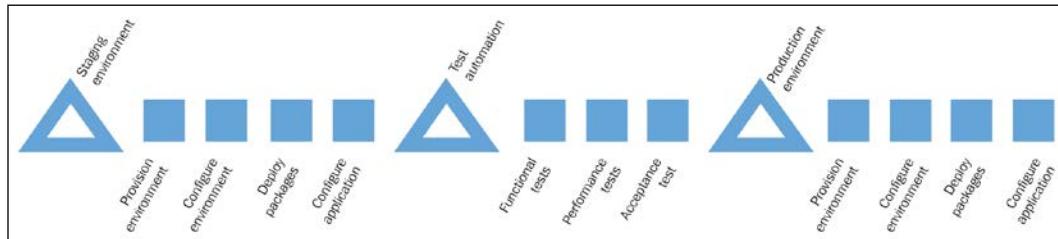
By the time the process reaches continuous deployment, continuous integration has ensured that we have fully-working bits of an application that can now be taken through different continuous-deployment activities. Continuous deployment refers to the capability of deploying business applications and services to preproduction and production environments through automation. For example, continuous deployment could provision and configure the preproduction environment, deploy applications to it, and configure the applications. After conducting multiple validations, such as functional tests and performance tests on the preproduction environment, the production environment is provisioned, configured, and the application is deployed through automation. There are no manual steps in the deployment process.

Every deployment task is automated. Continuous deployment can provision the environment and deploy the application from scratch while it can just deploy the delta changes to existing environment if the environment already exists.

All the environments are provisioned through automation using IaC. This ensures that all environments, whether development, test, preproduction, or production, are the same. Similarly, the application is deployed through automation, ensuring that it's also deployed uniformly across all environments. The configuration across these environments could be different for the application.

Continuous deployment is generally integrated with continuous integration. When continuous integration has done its work, by generating the final deployable packages, continuous deployment kicks in and starts its own pipeline. This pipeline is called the release pipeline. The release pipeline consists of multiple environments, with each environment consisting of tasks responsible for provisioning the environment, configuring the environment, deploying applications, configuring applications, executing operational validation on environments, and testing the application on multiple environments.

Employing continuous deployment provides immense benefits. There is a high level of confidence in the overall deployment process, which helps with faster and risk-free releases on production. The chances of anything going wrong decreases drastically. The team will be less stressed, and rollback to the previous working environment is possible if there are issues with the current release:



Although every system demands its own configuration of the release pipeline, a small sample of continuous deployment is shown in the preceding diagram. It's important to note that, generally, provisioning and configuring multiple environments is part of the release pipeline, and approvals should be sought before moving to the next environment. The approval process might be manual or automated, depending on the maturity of the organization.

Test environment deployment

The release pipeline starts once the drop is available from continuous integration and the first step it should take is to get all the artifacts from the drop. After which, it might create a completely new bare-metal test environment or reuse an existing one. This is again dependent on the type of project and the nature of the testing planned to be executed on this environment. The environment is provisioned and configured. The application artifacts are deployed and configured.

Test automation

After deploying an application, a series of tests can be performed on the environment. One of the tests executed here is a functional test. Functional tests are primarily aimed at validating the feature completeness and functionality of the application. These tests are written from requirements gathered from the customer. Another set of tests that can be executed is related to the scalability and availability of the application. This typically includes load tests, stress tests, and performance tests. It should also include an operational validation of the infrastructure environment.

Staging environment deployment

This is very similar to the test environment deployment, the only difference being that the configuration values for the environment and application would be different.

Acceptance tests

Acceptance tests are generally conducted by application stakeholders, and this can be manual or automated. This step is a validation from the customer's point of view about the correctness and completeness of the application's functionality.

Deployment to production

Once the customer gives their approval, the same steps as that of the test and staging environment deployment are executed, the only difference being that the configuration values for the environment and application are specific to the production environment. A validation is conducted after deployment to ensure that the application is running according to expectations.

Continuous delivery

Continuous delivery and continuous deployment might sound similar to you; however, they aren't the same. While continuous deployment talks about deployment to multiple environments and finally to the production environment through automation, continuous delivery is the ability to generate application packages that are readily deployable in any environment. For generating artifacts that are readily deployable, continuous integration should be used to generate the application artifacts, a new or existing environment should be used to deploy these artifacts, and conduct functional tests, performance tests, and user-acceptance tests through automation. Once these activities are successfully executed without any errors, the application package is considered readily deployable. Continuous delivery includes continuous integration and deployment to an environment for final validations. It helps get feedback more quickly from both the operations and the end user. This feedback can then be used to implement subsequent iterations.

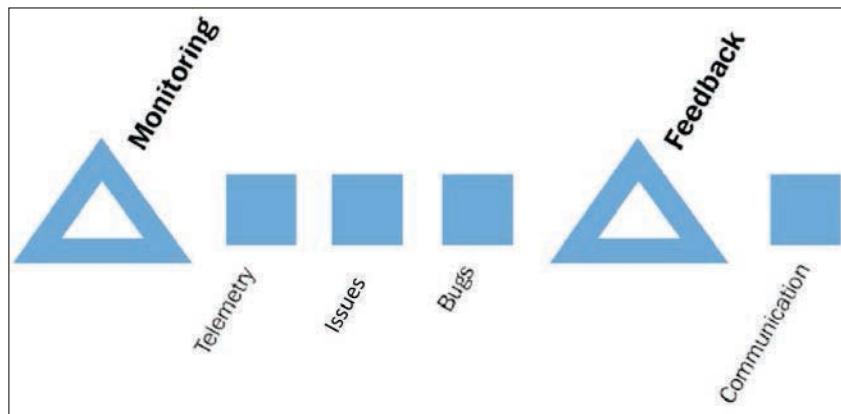
Continuous learning

With all the previously mentioned DevOps practices, it's possible to create great business applications and deploy them automatically to the production environment; however, the benefits of DevOps won't last for long if continuous improvement and feedback principles are not in place. It's of the utmost importance that real-time feedback about the application behavior is passed on as feedback to the development team from both end users and the operations team.

Feedback should be passed to the teams, providing relevant information about what's going well and what isn't.

An application's architecture and design should be built with monitoring, auditing, and telemetry in mind. The operations team should collect telemetry information from the production environment, capture any bugs and issues, and pass it on the development team so that it can be fixed for subsequent releases.

Continuous learning helps to make the application robust and resilient to failure. It helps in making sure that the application is meeting consumer requirements. The following diagram shows the feedback loop that should be implemented between different teams:



Azure DevOps

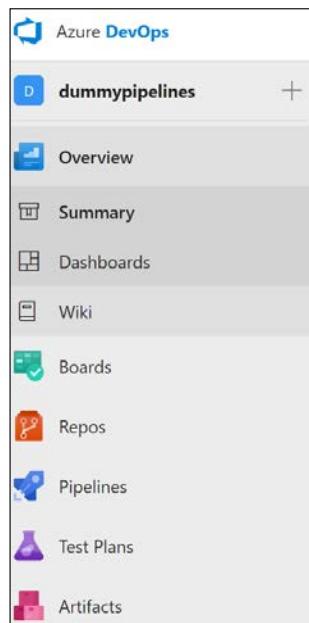
Now, it's time to focus on another revolutionary online service that enables continuous integration, continuous deployment, and continuous delivery seamlessly: Azure DevOps. In fact, it would be more appropriate to call it a suite of services available under a single name. Azure DevOps is a PaaS provided by Microsoft and hosted on the cloud. The same service is available as **Team Foundation Services (TFS)** on-premise. All examples shown in this book use Azure DevOps.

According to Microsoft, Azure DevOps is a cloud-based collaboration platform that helps teams to share code, track work, and ship software. Azure DevOps is a new name; earlier, it was known as **Visual Studio Team Services (VSTS)**. Azure DevOps is an enterprise software-development tool and service that enables organizations to provide automation facilities to their end-to-end application life cycle management process, from planning to deploying applications, and getting real-time feedback from software systems. This increases the maturity and capability of an organization to deliver high-quality software systems to their customers.

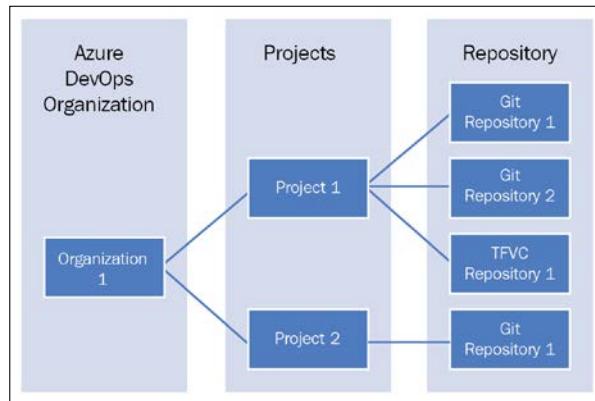
Successful software delivery involves efficiently bringing numerous processes and activities together. These include executing and implementing various Agile processes, increasing collaboration among teams, the seamless and automatic transition of artifacts from one phase of the ALM to another phase, and deployments to multiple environments. It's important to track and report on these activities to measure and improve delivery processes. Azure DevOps makes this simple and easy. It provides a whole suite of services that enables the following:

- Collaboration among every team member by providing a single interface for the entire application life cycle management.
- Collaboration among development teams using source-code-management services.
- Collaboration among test teams using test-management services.
- Automatic validation of code and packaging through continuous integration using build-management services.
- Automatic validation of application functionality, deployment, and configuration of multiple environments through continuous deployment and delivery using release-management services.
- Tracking and work-item management using work-management services.

The following screenshot shows all the services available to a project from the **Azure DevOps** left navigation bar:



An organization in Azure DevOps is a security boundary and logical container that provides all the services that are needed to implement a DevOps strategy. Azure DevOps allows the creation of multiple projects within a single organization. By default, a repository is created with the creation of a project; however, Azure DevOps allows for the creation of additional repositories within a single project. The relationship between the **Azure DevOps Organization**, **Projects**, and **Repository** is shown in the following diagram:



Azure DevOps provides two types of repositories:

- Git
- **Team Foundation Version Control (TFVC)**

It also provides the flexibility to choose between the Git or TFVC source-control repository. There can be a combination of TFS and TFVC repositories available within a single project.

TFVC

TFVC is the traditional and centralized way of implementing version control, where there's a central repository and developers work on it directly in connected mode to check-in their changes. If the central repository is offline or unavailable, developers can't check in their code and have to wait for it to be online and available. Other developers can see only the checked-in code. Developers can group multiple changes into a single change set for checking in code changes that are logically grouped to form a single change. TFVC locks the code files that are undergoing edits. Other developers can read the locked-up file, but they can't edit it. They must wait for the prior edit to complete and release the lock before they can edit. The history of check-ins and changes is maintained on the central repository, while the developers have the working copy of the files but not the history.

TFVC works very well with large teams that are working on the same projects. This enables control over the source code at a central location. It also works best for long duration projects since the history can be managed at a central location. TFVC has no issues working with large and binary files.

Git

Git, on the other hand, is a modern, distributed way of implementing version control, where developers can work on their own local copies of code and history in offline mode. Developers can work offline on their local clone of code. Each developer has a local copy of code and entire history, and they work on their changes with this local repository. They can commit their code to the local repository. They can connect to the central repository for synchronization of their local repository on a per-need basis. This allows every developer to work on any file, since they would be working on their local copy. Branching in Git doesn't create another copy of the original code and is extremely fast to create.

Git works well with both small and large teams. Branching and merging is a breeze with the advanced options that Git has.

Git is the recommended way of using source control because of the rich functionality it provides. We'll use Git as the repository for our sample application in this book.

Preparing for DevOps

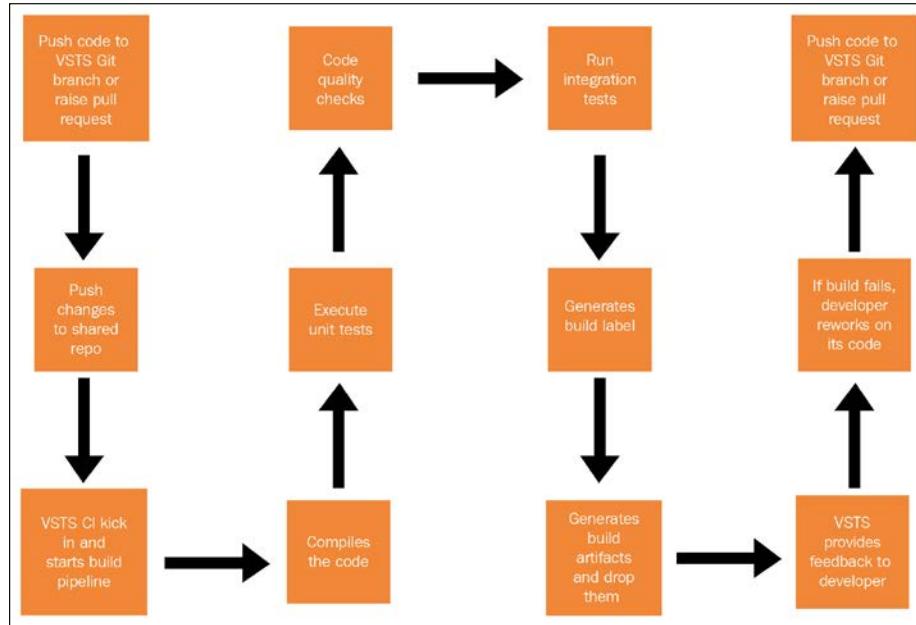
Going forward, our focus will be on process and deployment automation using different patterns in Azure. These include the following:

- DevOps for IaaS solutions
- DevOps for PaaS solutions
- DevOps for container-based solutions

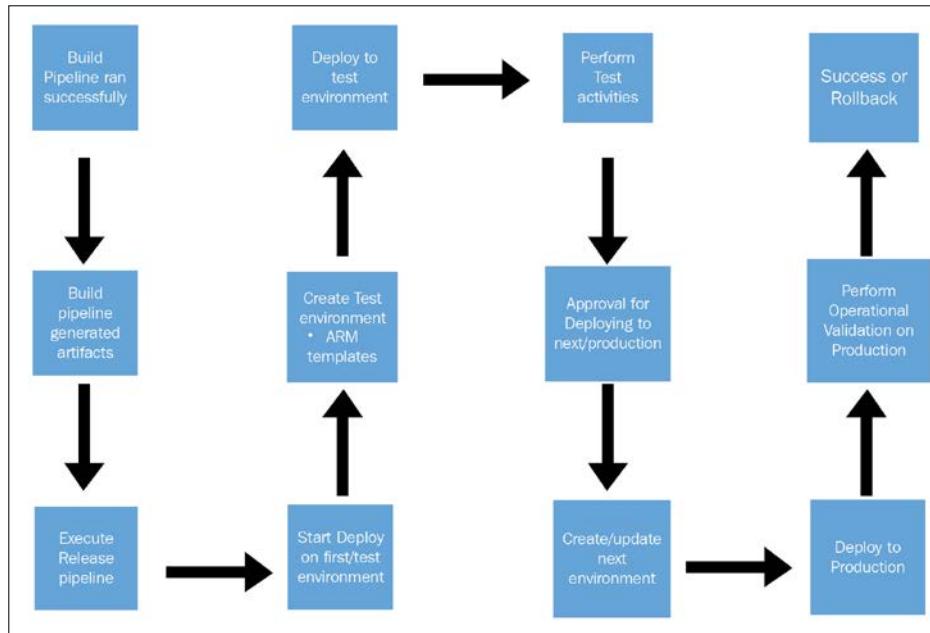
Generally, there are shared services that aren't unique to any one application. Their services are consumed by multiple applications from different environments, such as development, testing, and production. The life cycle of these shared services is different for each application. Therefore, they have different version-control repositories, a different code base, and build and release management. They have their own cycle of plan, design, build, test, and release.

The resources that are part of this group are provisioned using ARM templates, PowerShell, and DSC configurations.

The overall flow for building these common components is shown here:



The release process is shown in the following diagram:



On the DevOps journey, it's important to understand and provision the common components and services before starting any software engagement, product, or service.

Provisioning Azure DevOps organization

A version-control system is needed to collaborate at the code level. Azure DevOps provides both centralized and decentralized versions of control systems. Azure DevOps also provides orchestration services for building and executing build and release pipelines. It's a mature platform that organizes all DevOps-related version control, and builds and releases work-item-related artifacts. After an organization is provisioned in Azure DevOps, an Azure DevOps project should be created to hold all project-related artifacts.

An Azure DevOps organization can be provisioned by visiting <https://dev.azure.com>.

Provisioning the Azure Key Vault

It isn't advisable to store secrets, certificates, credentials, or other sensitive information in code configuration files, databases, or any other general storage system. It's advised to store this important data in a vault that's specifically designed for storing secrets and credentials. The Azure Key Vault provides such a service. The Azure Key Vault is available as a resource and service from Azure.

Provisioning a configuration-management server/service

A configuration-management server/service that provides storage for configurations and applies those configurations to different environments is always a good strategy for automating deployments. DSC on custom virtual machines, DSC from Azure Automation, Chef, Puppet, and Ansible are some options, and can be used on Azure seamlessly for both Windows as well as Linux environments. This book uses DSC as a configuration-management tool for all purposes, and it provides a pull server that holds all configuration documents (MOF files) for the sample application. It also maintains the database of all virtual machines and containers that are configured and registered with the pull server to pull configuration documents from it. The local configuration manager on these target virtual machines and containers periodically checks the availability of new configurations as well as drifts in current configuration and reports it back to the pull server. It also has built-in reporting capabilities that provide information about nodes that are compliant, as well as those that are non-compliant, within a virtual machine. A pull server is a general web application that hosts the DSC pull server endpoint.

Provisioning log analytics

Log analytics is an audit and monitoring service provided by Azure to get real-time information about all changes, drifts, and events occurring within virtual machines and containers. It provides a centralized workspace and dashboard for IT administrators for viewing, searching, and conducting drill-down searches on all changes, drifts, and events that occur on these virtual machines. It also provides agents that are deployed on target virtual machines and containers. Once deployed, these agents start sending all changes, events, and drifts to the centralized workspace.

Azure Storage account

Azure Storage is a service provided by Azure to store files as blobs. All scripts and code for automating the provisioning, deployment, and configuration of the infrastructure and sample application are stored in the Azure DevOps Git repository, and are packaged and deployed in an Azure Storage account. Azure provides PowerShell script-extension resources that can automatically download DSC and PowerShell scripts and execute them on virtual machines during the execution of Azure resource-manager templates. This storage acts as common storage across all deployments for multiple applications.

Source images

Both virtual machine and container images should be built as part of the common services build-and-release pipeline. Tools such as Packer and Docker Build can be used to generate these images.

Monitoring tools

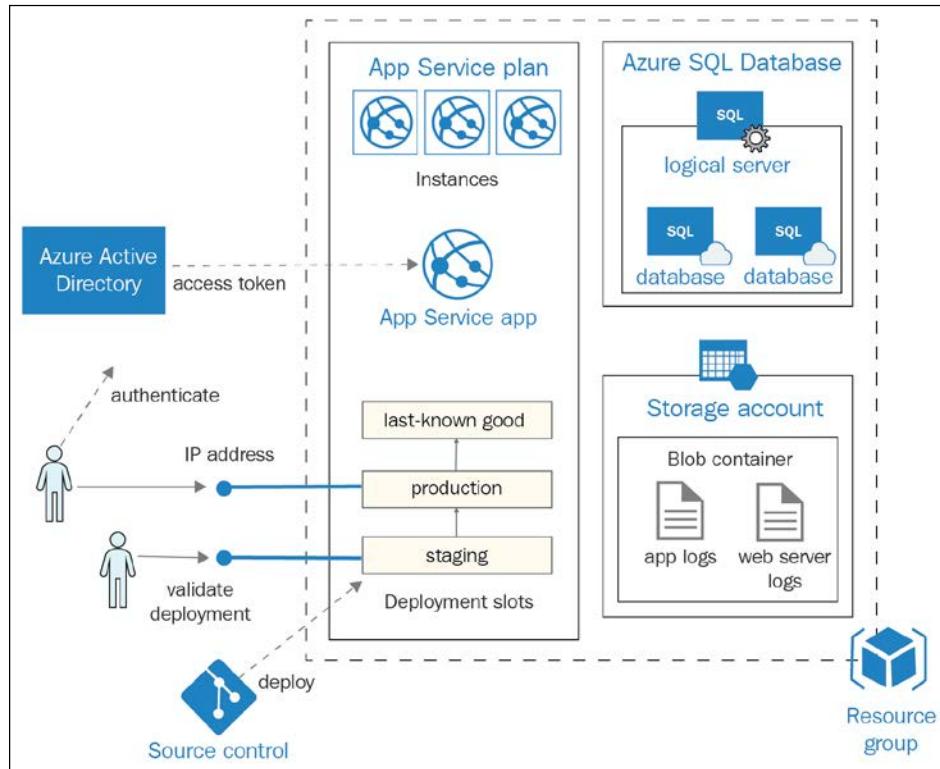
All monitoring tools, such as Azure Monitor, Application Insights, Log Analytics, OMS, and the System Center Operations Manager should be provisioned and configured during the release pipeline of common services.

Management tools

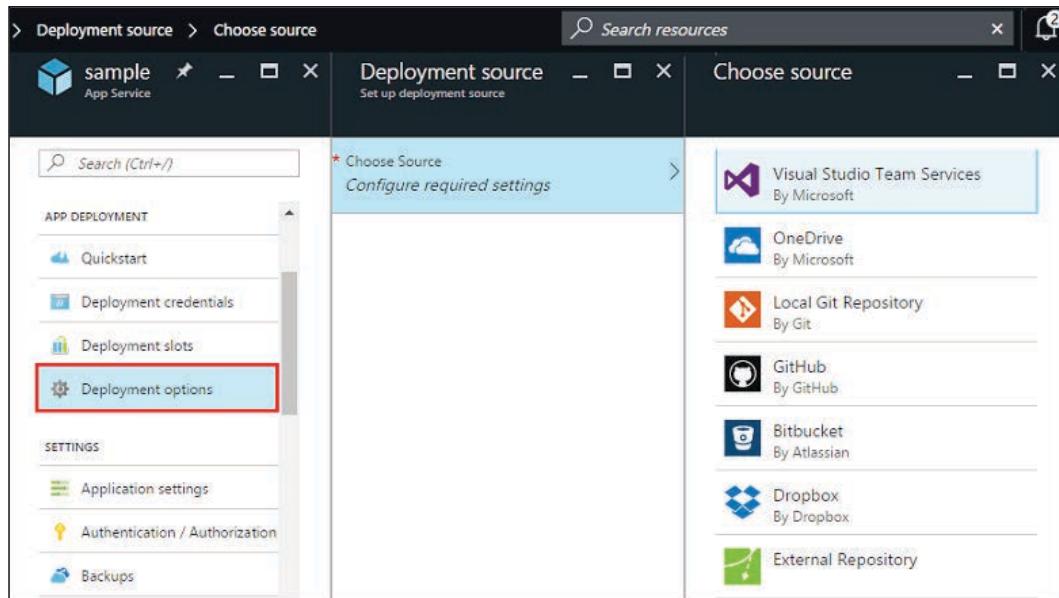
All management tools, such as Kubernetes, DC/OS, Docker Swarm, and ITIL tools, should be provisioned at this stage.

DevOps for PaaS solutions

The typical architecture for Azure PaaS app services is based on the following diagram:



The architecture shows some of the important components, such as Azure SQL, storage accounts, the version control system, that participate in the Azure App Services-based cloud solution architecture. These artifacts should be created using ARM templates. These ARM templates should be part of the overall configuration-management strategy. It can have its own build and release management pipelines, similar to the one shown in the following screenshot:



The template should also configure continuous deployment by configuring **Deployment options**.

Azure App Services

Azure App Services provides managed hosting services for cloud solutions. It's a fully-managed platform that provisions and deploys cloud solutions. Azure App Services takes away the burden of creating and managing infrastructure and provides minimum **service-level agreements (SLA)** for hosting your cloud solutions.

They are open; they let users decide on the language they want to use to build their cloud solutions, and flexible enough to host the cloud solution on either the Windows or Linux operating system. Creating an app service internally provisions virtual machines behind the scenes that are completely managed by Azure, and users don't see them at all. Multiple types of cloud solutions, such as web applications, mobile backend APIs, API endpoints, and containers can be hosted seamlessly on Azure App Services.

Deployment slots

Azure App Services provides deployment slots that make deployment to them seamless and easy. There are multiple slots, and swapping between slots is at DNS level. It means anything in the production slot can be swapped with a staging slot by just swapping the DNS entries. This helps in deploying the custom cloud solution to staging and, after all checks and tests, they can be swapped to production if found satisfactory. However, in case of any issue in production after swapping, the previous good values from the production environment can be reinstated by swapping again.

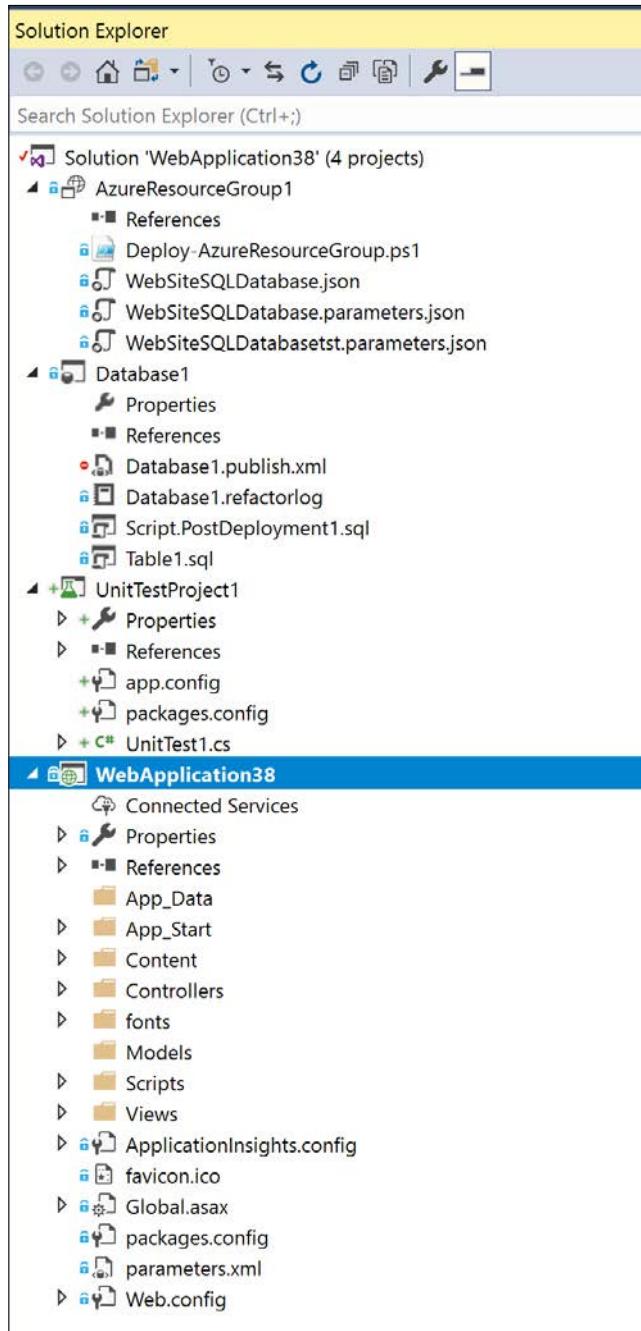
Azure SQL

Azure SQL is a SQL PaaS service provided by Azure to host databases. Azure provides a secure platform to host databases and takes complete ownership to manage the availability, reliability, and scalability of the service. With Azure SQL, there's no need to provision custom virtual machines, deploy a SQL server, and configure it. Instead, the Azure team does this behind the scenes and manages it on our behalf. It also provides a firewall service that enables security; only an IP address allowed by the firewall can connect the server and access the database. The virtual machines provisioned to host web applications have distinct public IP addresses assigned to them and they're added to Azure SQL firewall rules dynamically. Azure SQL Server and its database is created upon executing the ARM template.

The build-and-release pipeline

In this section, a new build pipeline is created that compiles and validates an ASP.NET MVC application, and then generates packages for deployment. After package generation, a release definition ensures that deployment to the first environment happens in an app service and Azure SQL as part of continuous deployment.

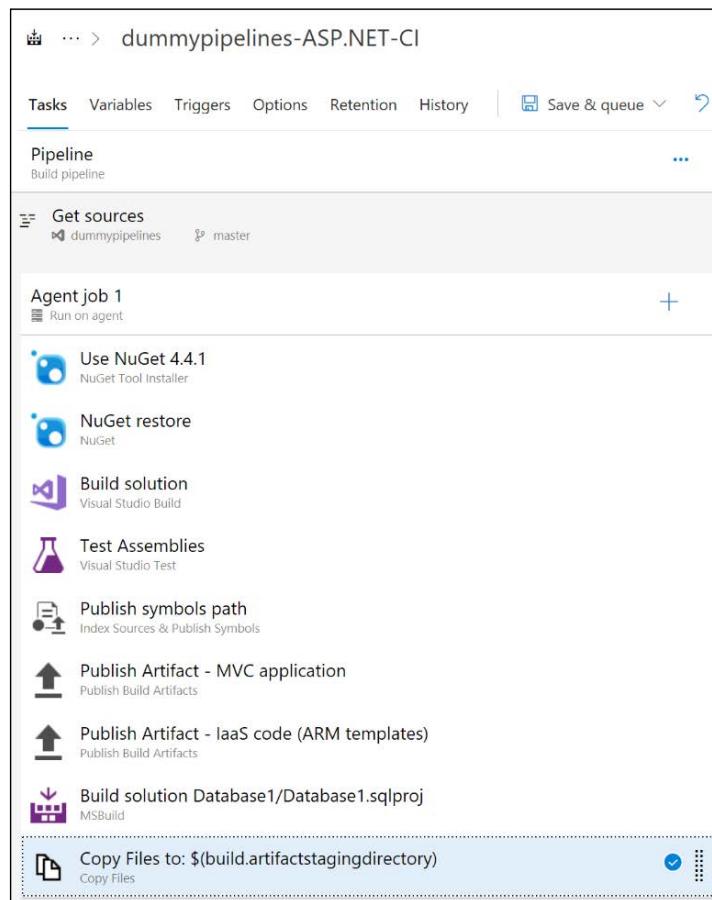
The project structure of the sample application is shown in the following screenshot:



In this project, there's an ASP.NET MVC application – the main application, which consists of application pages. Webdeploy packages will be generated out of this project from build pipelines and they will eventually be on Azure Web Apps.

- **Unit test project:** Code for unit-testing the ASP.NET MVC application. Assemblies from this project will be generated and executed in build execution.
- **SQL Database project:** Code related to the SQL database schema, structure, and master data. DacPac files will be generated out of this project using the build definition.
- **Azure Resource group project:** ARM templates and parameters code to provision the entire Azure environment on which the ASP.NET MVC application and the SQL tables are created.

The build pipeline is shown in the following screenshot:



The configuration of each task is shown in the following table:

Task name	Task configuration
Use NuGet 4.4.1	
NuGet restore	
Build solution	
Test Assemblies	

Task name	Task configuration
Publish symbols path	<p>Index Sources & Publish Symbols</p> <p>Version: 2.*</p> <p>Display name *</p> <p>Publish symbols path</p> <p>Path to symbols folder</p> <p>\$(build.SourcesDirectory)</p> <p>Search patterns *</p> <p>**\bin**.pdb</p> <p><input checked="" type="checkbox"/> Index sources</p> <p><input type="checkbox"/> Publish symbols</p>
Publish Artifact - MVC application	<p>Publish Build Artifacts</p> <p>Version: 1.*</p> <p>Display name *</p> <p>Publish Artifact - MVC application</p> <p>Path to publish</p> <p>\$(build.ArtifactStagingDirectory)</p> <p>Artifact name *</p> <p>drop</p>
Publish Artifact - IaaS code (ARM templates)	<p>Publish Build Artifacts</p> <p>Version: 1.*</p> <p>Display name *</p> <p>Publish Artifact - IaaS code (ARM templates)</p> <p>Path to publish</p> <p>AzureResourceGroup1</p> <p>Artifact name *</p> <p>drop1</p> <p>Artifact publish location *</p> <p>Azure Pipelines/TFS</p>
Build solution Database1/Database1.sqlproj	<p>MSBuild</p> <p>Version: 1.*</p> <p>Display name *</p> <p>Build solution Database1/Database1.sqlproj</p> <p>Project *</p> <p>Database1/Database1.sqlproj</p> <p>MSBuild</p> <p><input checked="" type="radio"/> Version</p> <p>MSBuild Version</p> <p>Latest</p> <p>MSBuild Architecture</p> <p>MSBuild x64</p> <p>Platform</p> <p>Configuration</p> <p>MSBuild Arguments</p> <p>/t:build /p:Config=MemoryStorage=True</p>

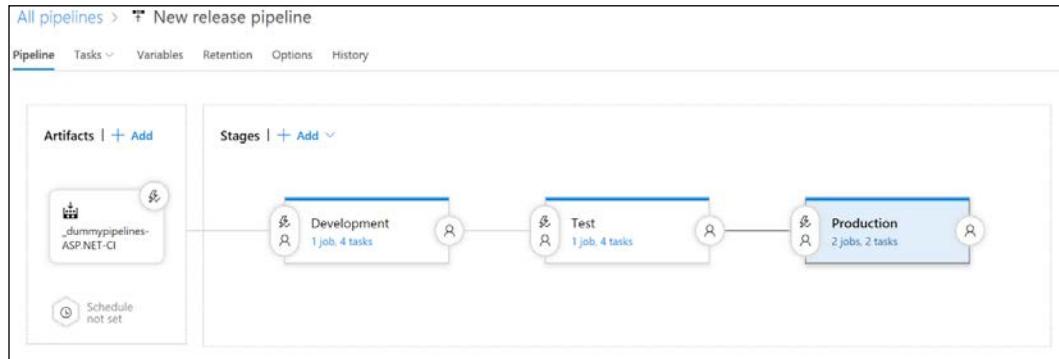
Task name	Task configuration
Copy Files to: \$(build.artifactstagingdirectory)	<p>Copy Files</p> <p>Version: 2.*</p> <p>Display name: Copy Files to: \$(build.artifactstagingdirectory)</p> <p>Source Folder: ...</p> <p>Content: ***.dbspar</p> <p>Target Folder: \$(build.artifactstagingdirectory)</p>

The build pipeline is configured to execute automatically as part of continuous integration, as shown in the following screenshot:

The screenshot shows the 'Triggers' tab of a build pipeline named 'dummypipelines-ASP.NET-CI'. Under the 'Continuous integration' section, the 'dummypipelines' trigger is listed as 'enabled'. In the 'Branch filters' section, there is one entry for the 'master' branch under the 'Include' type. Other sections like 'Scheduled' and 'Build completion' are also visible.

The release definition consists of multiple environments, such as development, testing, SIT, UAT, preproduction, and production. The tasks are pretty similar in each environment, with the addition of tasks specific to that environment. For example, a test environment has additional tasks related to the UI, and functional and integration testing, compared to a development environment.

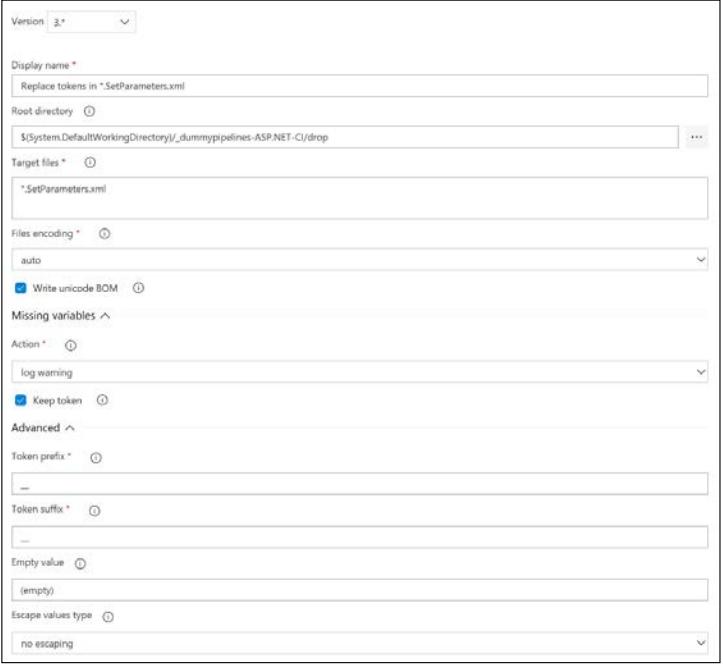
The release definition for this application is shown in the following screenshot:



The release tasks for a single environment are shown in the following screenshot:

The screenshot shows the 'Development' stage of the release pipeline. It has a 'Deployment process' header. Below it, there's a 'Run on agent' section with a 'Run on agent' button and a '+' button. Underneath are four task cards: 'Replace tokens in *.SetParameters.xml' (Replace Tokens), 'Azure Deployment:Create Or Update Resource Group acti...' (Azure Resource Group Deployment), 'Deploy Azure App Service' (Azure App Service Deploy), and 'Azure SQL Publish' (Azure SQL Database Deployment).

The configuration for each of the tasks is listed here:

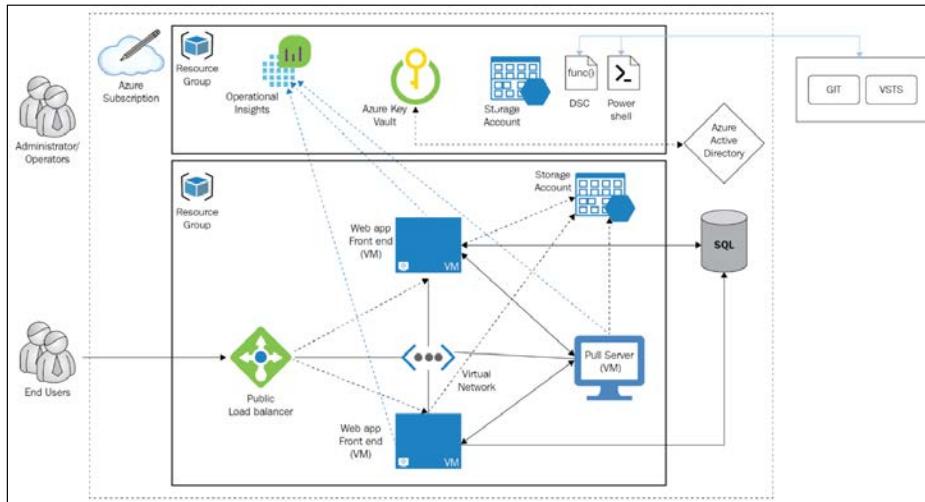
Task name	Task configuration
Replace tokens in *.SetParameters.xml (This is a task installed from MarketPlace.)	 The screenshot shows the configuration interface for the 'Replace tokens in *.SetParameters.xml' task. It includes fields for 'Display name', 'Root directory', 'Target files', 'Files encoding', 'Action', 'Advanced' settings like 'Token prefix' and 'Token suffix', and options for 'Empty value' and 'Escape values type'. A note at the bottom indicates that 'Keep token' is selected.

Task name	Task configuration
Azure Deployment:Create Or Update Resource Group action on devRG	<p>Azure Resource Group Deployment</p> <p>Version: 2.*</p> <p>Display name: Azure Deployment:Create Or Update Resource Group action on devRG</p> <p>Azure Details</p> <p>Azure subscription: myconnection (Selected to subscription "Visual Studio Enterprise")</p> <p>Action: Create or update resource group</p> <p>Resource group: devRG</p> <p>Location: West Europe</p> <p>Template</p> <p>Template location: Linked artifact</p> <p>Template: \${System.DefaultWorkingDirectory}/_dummypipelines-ASP.NET-C/drop1/WebSiteSQLDatabase.json</p> <p>Template parameters</p> <p>-sqlserverName \${SQLServerName} -hostingPlanName \${AppServiceName}</p> <p>Override template parameters</p> <p>Deployment mode: Incremental</p>
Deploy Azure App Service	<p>Azure App Service Deploy</p> <p>Version: 3.*</p> <p>Display name: Deploy Azure App Service</p> <p>Azure subscription: myconnection (Selected to subscription "Visual Studio Enterprise")</p> <p>App type: webApp</p> <p>App Service name: \${AppServiceName}</p> <p><input type="checkbox"/> Deploy to slot</p> <p>Virtual application:</p> <p>Package or folder: \${System.DefaultWorkingDirectory}/_dummypipelines-ASP.NET-C/drop/WebApplication3&.zip</p>

Task name	Task configuration
Azure SQL Publish	

DevOps for virtual machine (IaaS)-based solutions

The typical architecture for an IaaS virtual machine-based solution is shown here:



Azure Virtual Machines

Azure Virtual Machines that host web applications, application servers, databases, and other services are provisioned using ARM templates. Each virtual machine has a single network card with a public IP assigned to it. They're attached to a virtual network and have a private IP address from the same network. The public IP for virtual machines is optional since they're attached to a public load balancer. These virtual machines are based on a Windows 2016 server image. Operational insight agents are installed on virtual machines to monitor the virtual machines. PowerShell scripts are also executed on these virtual machines, downloaded from a storage account available in another resource group to open relevant firewall ports, download appropriate packages, and install local certificates to secure access through PowerShell. The web application is configured to run on the provided port on these virtual machines. The port number for the web application and all its configuration is pulled from the DSC pull server and dynamically assigned.

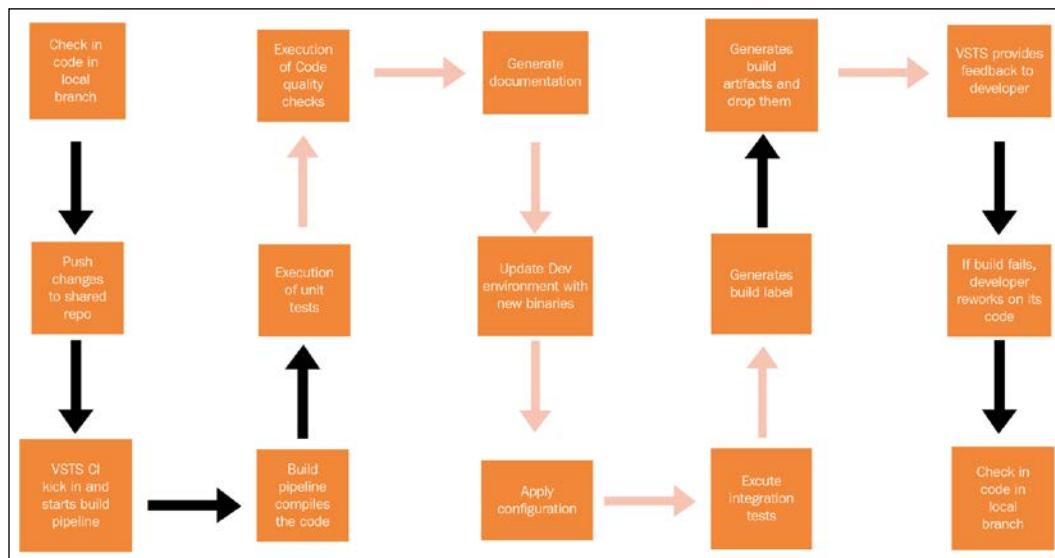
Azure public load balancers

A public load balancer is attached to some of the virtual machines for sending requests to them in a round-robin fashion. This is generally needed for frontend web applications and APIs. A public IP address and DNS name can be assigned to a load balancer such that it can serve internet requests. It accepts HTTP web requests on different port, and routes them to the virtual machines. It also probes certain ports on HTTP protocols with some provided application paths. **Network Address Translation (NAT)** rules can also be applied such that they can be used to log into the virtual machines using remote desktops.

An alternative resource to the Azure public load balancer is the Azure application gateway. Application gateways are layer-7 load balancers and provide features such as SSL termination, session affinity, and URL-based routing.

The build pipeline

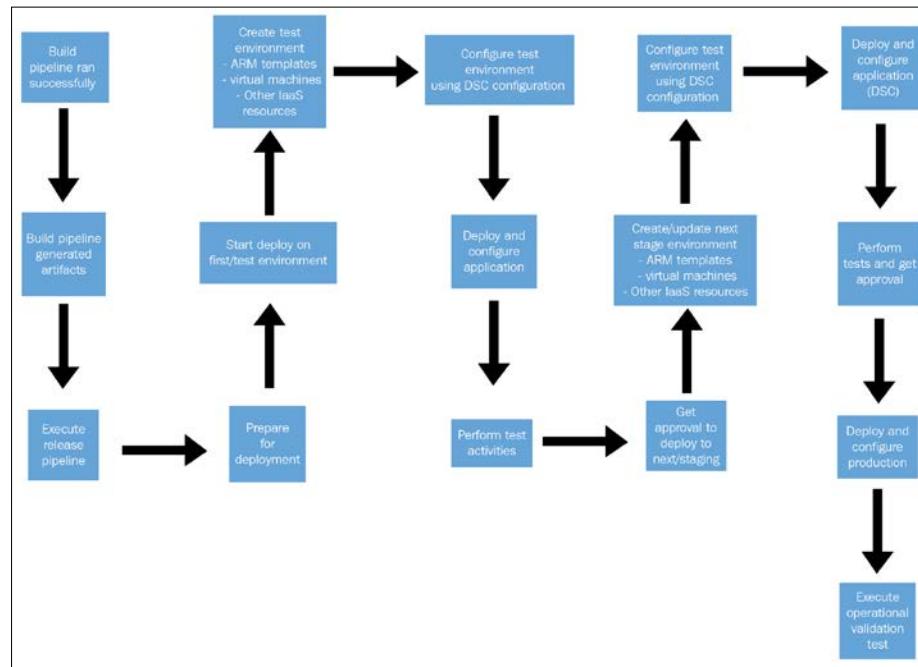
A typical build pipeline for an IaaS virtual machine-based solution is shown next. A release pipeline starts when a developer pushes their code to the repository. The build pipeline starts automatically as part of continuous integration. It compiles and builds the code, executes unit tests on it, checks code quality and generates documentation from code comments. It deploys the new binaries into the development environment (note that development environment is not newly created), changes configuration, executes integration tests, and generates build labels for easy identification. It then drops the generated artifacts into a location accessible by the release pipeline. If there are issues during the execution of any step in this pipeline, this is communicated to the developer as part of the build pipeline feedback so that they can rework and resubmit their changes. The build pipeline should fail or pass based on the severity of issues found, and that varies from organization to organization. A typical build pipeline is shown in the following diagram:



The release pipeline

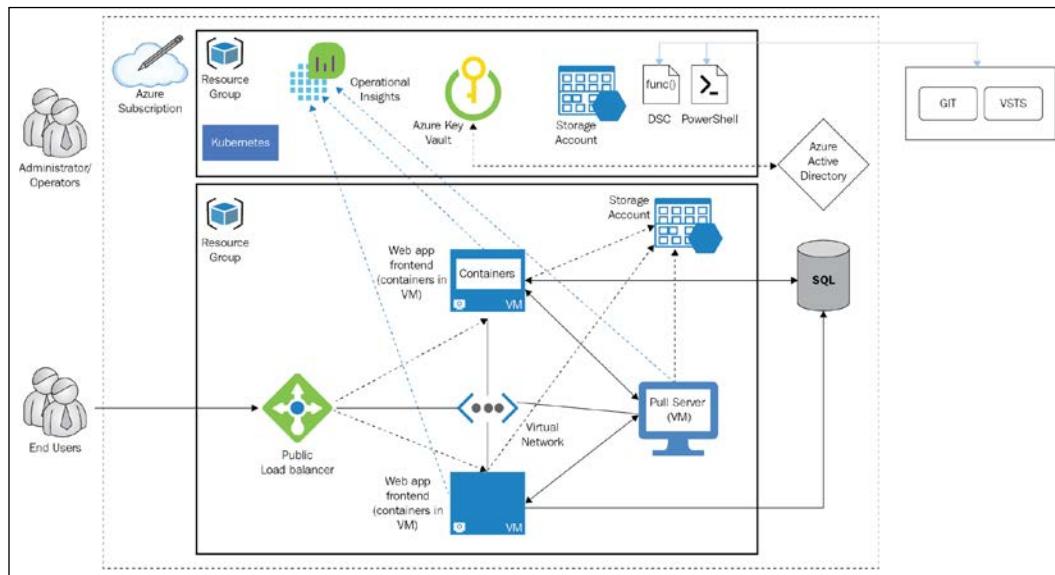
A typical release pipeline for an IaaS virtual machine-based deployment is shown next. A release pipeline starts after the completion of the build pipeline. The first step for the release pipeline is to gather the artifacts generated by the build pipeline. They are generally deployable assemblies, binaries, and configuration documents. The release pipeline executes and creates or updates the first environment, which generally is a test environment. It uses ARM templates to provision all IaaS and PaaS services and resources on Azure, and configures them as well. They also help in executing scripts and DSC configuration after virtual machines are created as post-creation steps. This helps to configure the environment within the virtual machine and the operating system. At this stage, application binaries from the build pipeline are deployed and configured. Different automated tests are performed to check the solution and, if found satisfactory, the pipeline moves deployment to the next environment after obtaining the necessary approvals. The same steps are executed on the next environment, including the production environment. Finally, the operational validation tests are executed on production to ensure that the application is working as expected and there are no deviations.

At this stage, if there are any issues or bugs, they should be rectified and the entire cycle should be repeated; however, if this doesn't happen within a stipulated time frame, the last-known snapshot should be restored on the production environment to minimize downtime. A typical release pipeline is shown in the following diagram:



DevOps for container-based (IaaS) solutions

The typical architecture for IaaS container-based solutions is shown here:



In the architecture shown earlier, container runtimes are deployed on virtual machines and containers are run within them. These containers are managed by container orchestrators such as Kubernetes. Monitoring services are provided by Log Analytics and all secrets and keys are stored in Azure Key Vault. There is also a pull server, which could be on a virtual machine or Azure Automation, providing configuration information to the virtual machines.

Containers

Containers are a virtualization technology; however, they don't virtualize physical servers. Instead, containers are an operating-system-level virtualization. This means that containers share the operating system kernel provided by their host among themselves and with the host. Running multiple containers on a host (physical or virtual) shares the host operating system kernel. There's a single operating system kernel provided by the host and used by all containers running on top of it.

Containers are also completely isolated from their host and other containers, much like a virtual machine. Containers use operating system namespaces, controls groups on Linux, to provide the perception of a new operating system environment, and use specific operating system virtualization techniques on Windows. Each container gets its own copy of the operating system resources.

Docker

Docker provides management features to containers. It comprises two executables:

- The Docker daemon
- The Docker client

The Docker daemon is the workhorse for managing containers. It's a management service that's responsible for managing all activities on the host related to containers. The Docker client interacts with the Docker daemon, and is responsible for capturing input and sending them to the Docker daemon. The Docker daemon provides the runtime; libraries; graph drivers; the engines to create, manage, and monitor containers; and images on the host server. It can also create custom images that are used for building and shipping applications to multiple environments.

Dockerfile

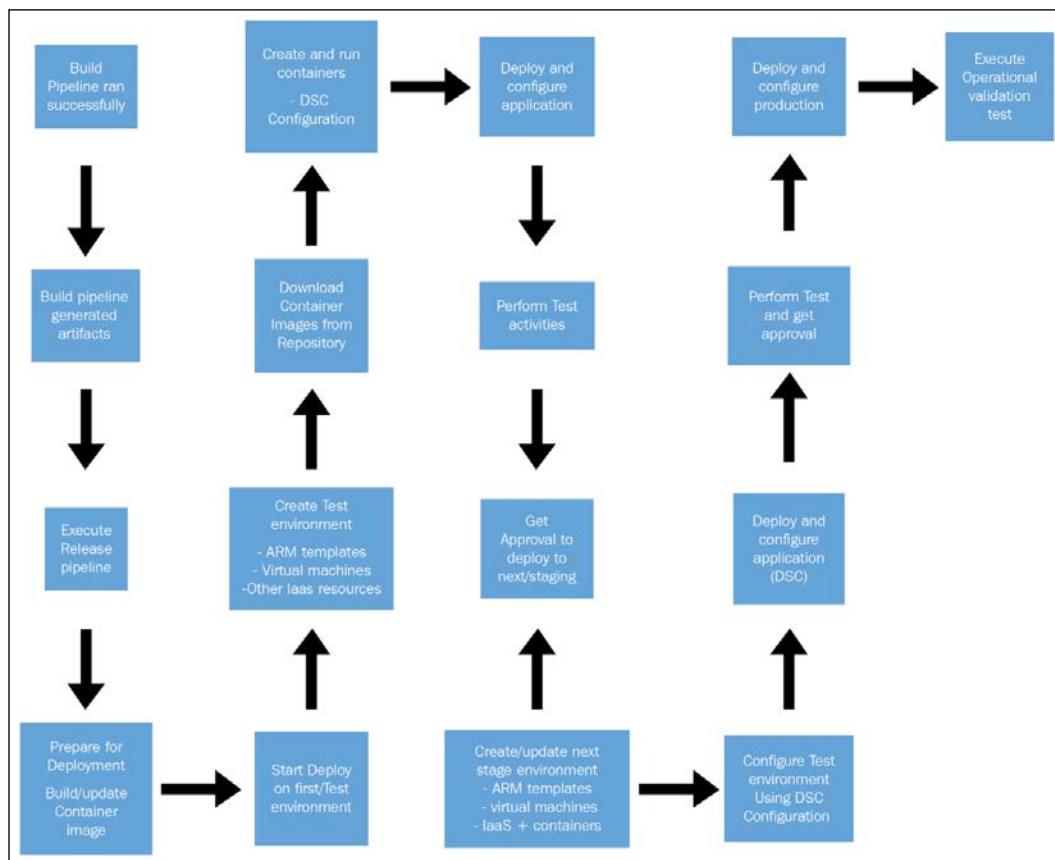
The Dockerfile is the primary building block for creating container images. It's a simple text-based human-readable file without an extension, and is even named Dockerfile. Although there's a mechanism to name it differently, generally it is named Dockerfile. The Dockerfile contains instructions to create a custom image using a base image. These instructions are executed sequentially from top to bottom by the Docker daemon. The instructions refer to the command and its parameters, such as `COPY`, `ADD`, `RUN`, and `ENTRYPOINT`. The Dockerfile enables IaC practices by converting the application deployment and configuration into instructions that can be versioned and stored in a source code repository.

The build pipeline

There's no difference, from the build perspective, between the container and a virtual-machine-based solution. The build step remains the same. Please refer to the *DevOps for virtual machine (IaaS)* based solutions section, for build pipeline details.

The release pipeline

A typical release pipeline for an IaaS container-based deployment is shown next. The only difference between this and the release pipeline is the container-image management and the creation of containers using Dockerfile and Docker Compose. Advanced container-management utilities, such as Docker Swarm, DC/OS, and Kubernetes, can also be deployed and configured as part of release management. However, note that these container management tools should be part of the shared services release pipeline, as discussed earlier. The following diagram shows a typical release pipeline for a container based solution:

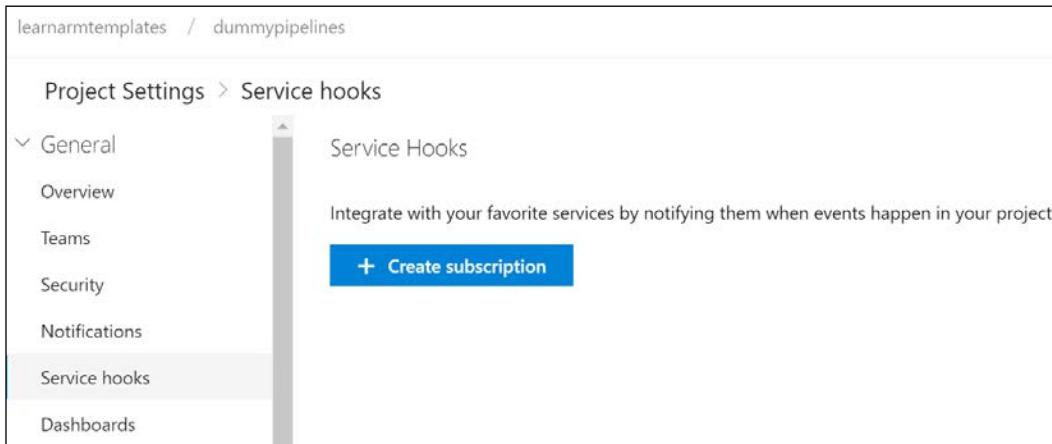


Azure DevOps and Jenkins

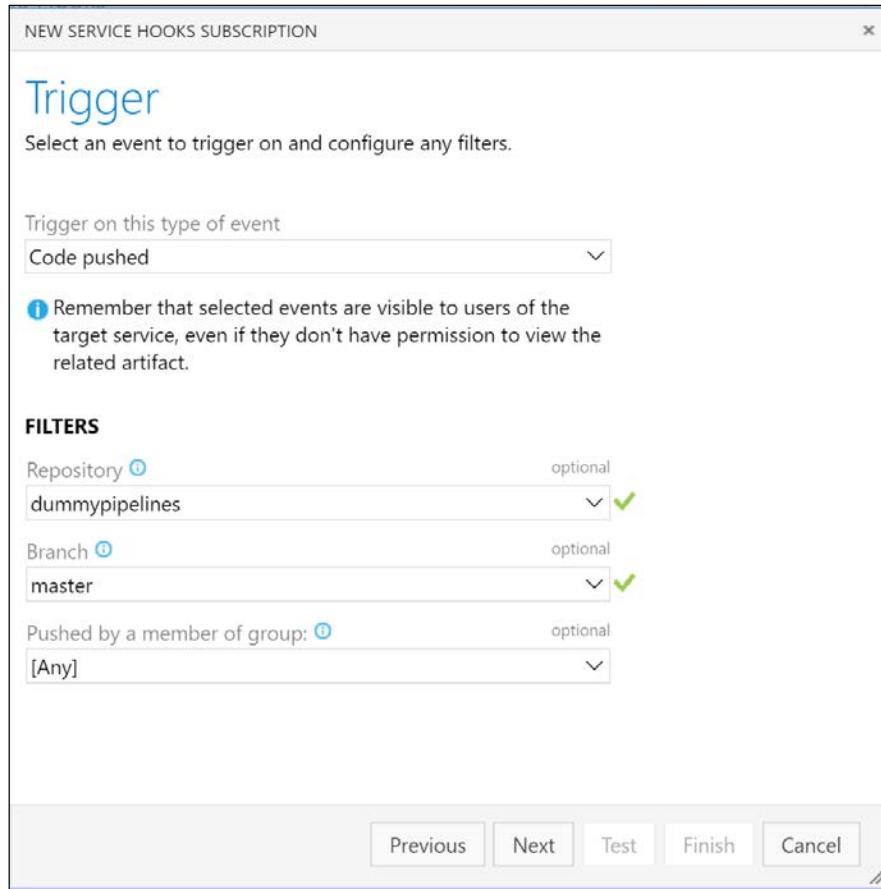
Azure DevOps is an open platform orchestrator that integrates with other orchestrator tools seamlessly. It provides all the necessary infrastructure and features that integrate well with Jenkins, as well. Organizations with well-established CI/CD pipelines built on Jenkins can reuse them with the advanced but simple features of Azure DevOps to orchestrate them.

Jenkins can be used as a repository and can execute CI/CD pipelines in Azure DevOps, while it's also possible to have a repository in Azure DevOps and execute CI/CD pipelines in Jenkins.

The Jenkins configuration can be added in Azure DevOps as service hooks, and whenever any code change is committed to the Azure DevOps repository, it can trigger pipelines in Jenkins. The next screenshot shows the configuration of Jenkins from the Azure DevOps service hook configuration section:



There are multiple triggers that execute the pipelines in Jenkins; one of them is **Code pushed**, as shown in the following screenshot:



It's also possible to deploy to Azure VM and execute Azure DevOps release pipelines, as explained here: <https://docs.microsoft.com/en-us/azure/virtual-machines/linux/tutorial-build-deploy-jenkins>.

Jenkins should already be deployed before using it in any scenario. The deployment process on Linux can be found at <https://docs.microsoft.com/en-us/azure/virtual-machines/linux/tutorial-jenkins-github-docker-cicd>.

Azure Automation

Azure Automation is Microsoft's platform for all automation implementation with regard to cloud, on-premise, and hybrid deployments. Azure Automation is a mature automation platform that provides rich capabilities in terms of the following:

- Defining assets, such as variables, connections, credentials, certificates, and modules
- Implementing runbooks using Python, PowerShell scripts, and PowerShell workflows
- Providing UIs to create runbooks
- Managing the full runbook life cycle, including building, testing, and publishing
- Scheduling runbooks
- The ability to run runbooks anywhere—on cloud or on-premise
- DSC as a configuration-management platform
- Managing and configuring environments—Windows and Linux, applications, and deployment
- The ability to extend Azure Automation by importing custom modules

Azure Automation provides a DSC pull server that helps to create a centralized configuration-management server that consists of configurations for nodes/virtual machines and their constituents.

It implements the hub and spoke pattern wherein nodes can connect to the DSC pull server and download configurations assigned to them, and reconfigure themselves to reflect their desired state. Any changes or deviations within these nodes are auto-corrected by DSC agents the next time they run. This ensures that administrators don't need to actively monitor the environment to find any deviations.

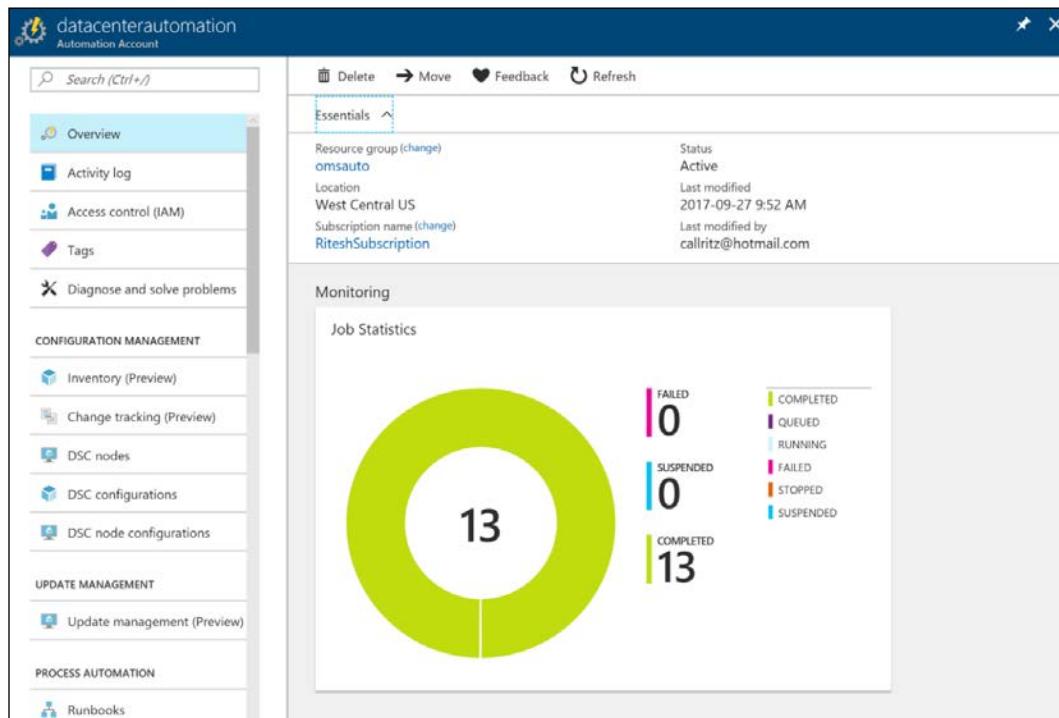
DSC provides a declarative language in which you define the intent and configuration, but not how to run and apply those configurations. These configurations are based on the PowerShell language and ease the process of configuration management.

In this section, we'll look into a simple implementation of using Azure Automation DSC to configure a virtual machine to install and configure the web server (IIS) and create an `index.htm` file that informs users that the website is under maintenance.

Provisioning the Azure Automation account

Create a new Azure Automation account from the Azure portal or PowerShell within an existing or new resource group. Astute readers will find in the following diagram that Azure Automation provides menu items for DSC. It provides the following:

- **DSC nodes:** These list all the virtual machines and containers that are enlisted with the current Azure Automation DSC pull server. These virtual machines and containers are managed using configurations from the current DSC pull server.
- **DSC configurations:** These list all the raw PowerShell configurations imported and uploaded to the DSC pull server. They are in human-readable format and aren't in a compiled state.
- **DSC node configurations:** These list all compiles of DSC configurations available on the pull server to be assigned to nodes—virtual machines and containers. A DSC configuration produces MOF files after compilations and they're eventually used to configure nodes. The next screenshot shows an Azure Automation account after getting provisioned:



Creating DSC configuration

The next step is to write a DSC configuration using any PowerShell editor to reflect the intent of the configuration. For this sample, a single configuration, `ConfigureSiteOnIIS`, is created. It imports the base DSC module, `PSDesiredStateConfiguration`, which consists of resources used within the configuration. It also declares a node web server. When this configuration is uploaded and compiled, it will generate a DSC Configuration named `ConfigureSiteOnIISwebserver`. This configuration can then be applied to nodes.

The configuration consists of a few resources. These resources configure the target node. The resources install a web server, ASP.NET, and framework, and create an `index.htm` file within the `inetpub\wwwroot` directory with content to show that the site is under maintenance. For more information about writing DSC configuration, refer to <https://docs.microsoft.com/en-us/PowerShell/dsc/configurations>.

The next code listing shows the entire configuration described in previous paragraph. This configuration will be uploaded to Azure Automation account:

```
Configuration ConfigureSiteOnIIS {
    Import-DscResource -ModuleName 'PSDesiredStateConfiguration'
    Node WebServer {
        WindowsFeature IIS
        {
            Name = "Web-Server"
            Ensure = "Present"
        }
        WindowsFeature AspDotNet
        {
            Name = "net-framework-45-Core"
            Ensure = "Present"
            DependsOn = "[WindowsFeature]IIS"
        }
        WindowsFeature AspNet45
        {
            Ensure = "Present"
            Name = "Web-Asp-Net45"
            DependsOn = "[WindowsFeature]AspNet"
        }
        File IndexFile
        {
            DestinationPath = "C:\inetpub\wwwroot\index.htm"
            Ensure = "Present"
            Type = "File"
            Force = $true
        }
    }
}
```

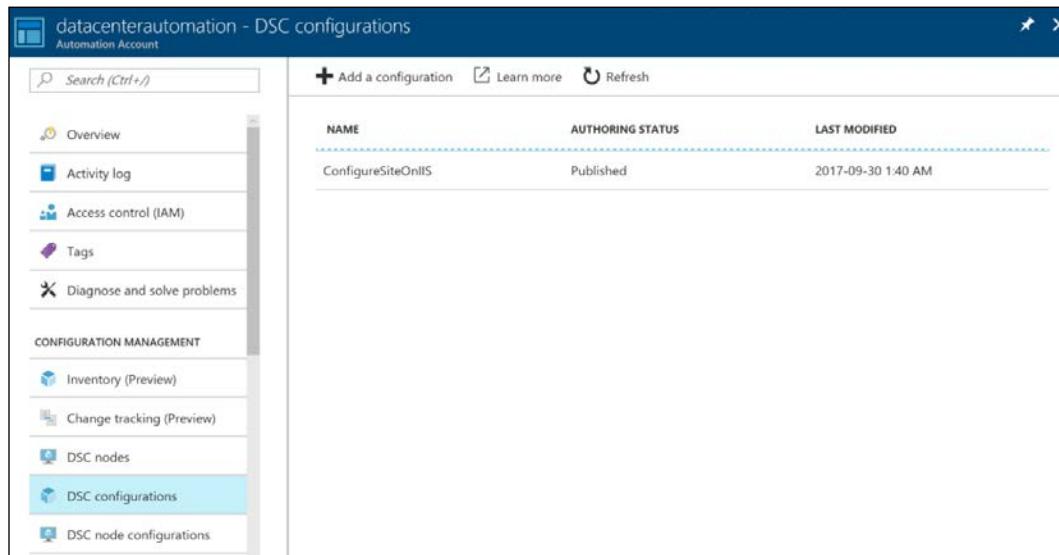
```
        Contents = "<HTML><HEAD><Title> Website under
construction.</Title></HEAD><BODY>
        <h1>If you are seeing this page, it means the website is
under maintenance and DSC Rocks !!!!!</h1></BODY></HTML>"
    }
}
}
```

Importing the DSC configuration

The DSC configuration still isn't known to Azure Automation. It's available on some local machines. It should be uploaded to Azure Automation DSC Configurations. Azure Automation provides the Import-AzureRmAutomationDscConfiguration cmdlet to import the configuration to Azure Automation:

```
Import-AzureRmAutomationDscConfiguration -SourcePath "C:\DSC\AA\
DSCfiles\ConfigSiteOnIIS.ps1" -ResourceGroupName "omsauto"
-AutomationAccountName "datacenterautomation" -Published -Verbose
```

The DSC configuration on Azure after applying the configuration to the node should appear as follows:



The screenshot shows the Azure DevOps Automation Account interface. The left sidebar has a navigation menu with items like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Configuration Management (Inventory (Preview), Change tracking (Preview), DSC nodes, DSC configurations, DSC node configurations), and a search bar. The main area is titled 'datacenterautomation - DSC configurations' and shows a table with one row of data. The table columns are NAME, AUTHORIZING STATUS, and LAST MODIFIED. The single row contains 'ConfigureSiteOnIIS', 'Published', and '2017-09-30 1:40 AM'. The 'DSC configurations' item in the sidebar is highlighted.

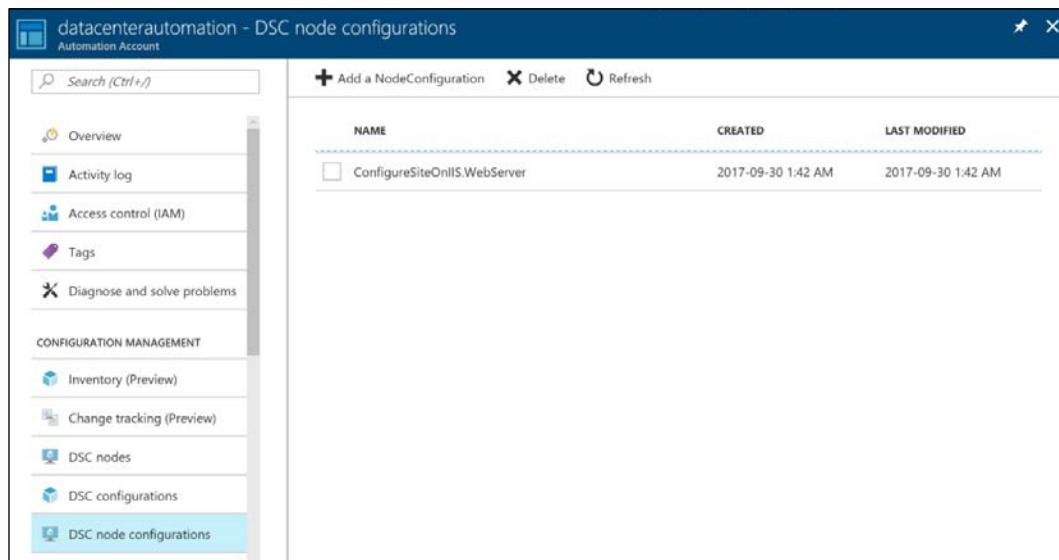
NAME	AUTHORIZING STATUS	LAST MODIFIED
ConfigureSiteOnIIS	Published	2017-09-30 1:40 AM

Compiling the DSC configuration

After the DSC configuration is available in Azure Automation, it can be asked to compile. Azure Automation provides another cmdlet for this. Use the `Start-AzureRmAutomationDscCompilationJob` cmdlet to compile the imported configuration. The configuration name should match the name of the uploaded configuration. Compilation creates an MOF file named after the configuration and node name together, which in this case is the `ConfigureSiteOnIIS` web server. The execution of command is shown here:

```
Start-AzureRmAutomationDscCompilationJob -ConfigurationName  
ConfigureSiteOnIIS -ResourceGroupName "omsauto" -AutomationAccountName  
"datacenterautomation" -Verbose
```

The DSC node's configuration on Azure after applying the configuration should appear as follows:



NAME	CREATED	LAST MODIFIED
ConfigureSiteOnIIS.WebServer	2017-09-30 1:42 AM	2017-09-30 1:42 AM

Assigning configurations to nodes

The compiled DSC configurations can be applied to nodes. Use Register-AzureRmAutomationDscNode to assign the configuration to a node. The NodeConfigurationName parameter identifies the configuration name that should be applied to the node. This is a powerful cmdlet that can also configure the DSC agent, that is localconfigurationmanager, on nodes before they can download configurations and apply them. There are multiple localconfigurationmanager parameters that can be configured—details are available at <https://docs.microsoft.com/en-us/PowerShell/dsc/metaconfig>.

```
Register-AzureRmAutomationDscNode -ResourceGroupName "omsauto" -AutomationAccountName "datacenterautomation" -AzureVMName testtwo -ConfigurationMode ApplyAndAutocorrect -ActionAfterReboot ContinueConfiguration -AllowModuleOverwrite $true -AzureVMResourceGroup testone -AzureVMLocation "West Central US" -NodeConfigurationName "ConfigureSiteOnIIS.WebServer" -Verbose
```

After applying the configuration, the DSC nodes on Azure should appear as follows:

NAME	STATUS	NODE CONFIGURATION	LAST SEEN	VM DSC EXT
testtwo	✓ Compliant	ConfigureSiteOnIIS.WebS...	2017-09-30 2:34 AM	Yes

Browsing the server

If appropriate, network security groups and firewalls are opened and enabled for port 80, and a public IP is assigned to the virtual machine; the default website can be browsed using the IP address. Otherwise, log into the virtual machine that's used to apply the DSC configuration and navigate to <http://localhost>.

It should show the following page:



This is the power of configuration management: without writing any significant code, authoring a configuration once can be applied multiple times to the same and multiple servers, and you can be assured that they will be running in the desired state without any manual intervention.

Azure for DevOps

As mentioned before, Azure is a rich and mature platform that provides the following:

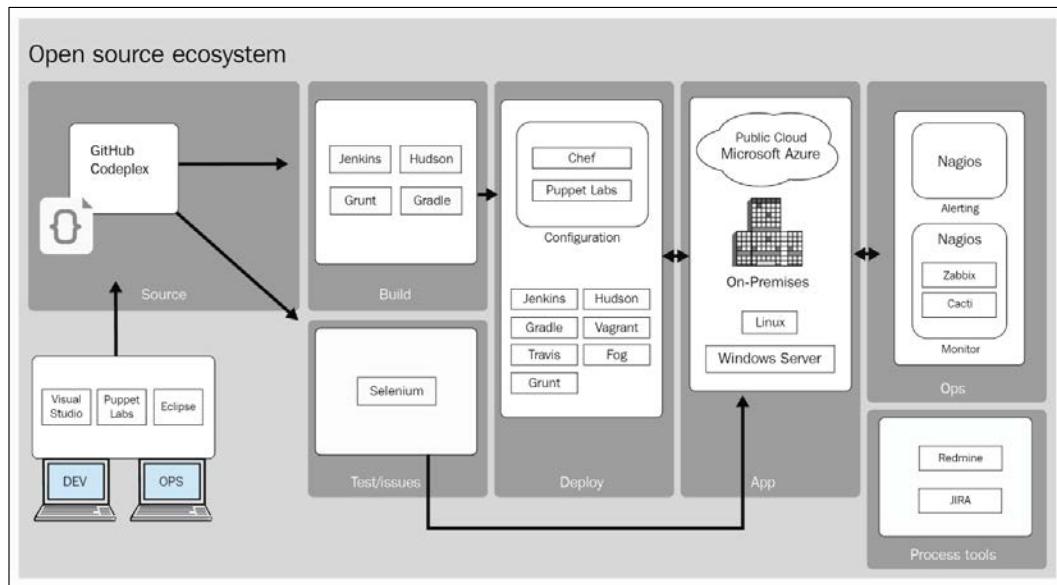
- Multiple choices of languages
- Multiple choices of operating systems
- Multiple choices of tools and utilities
- Multiple patterns for deploying solutions (such as virtual machines, app services, containers, and microservices)

With so many options and choices, Azure offers the following:

- **Open cloud:** It is open to open source, Microsoft, and non-Microsoft products, tools, and services.
- **Flexible cloud:** It is easy enough for both business users and developers to use it with their existing skills and knowledge.
- **Unified management:** It provides seamless monitoring and management features.

All the features mentioned here are important for the successful implementation of DevOps. The next diagram shows the open source tools and utilities that can be used for different phases in managing the application life cycle and DevOps in general. This is just a small representation of all the tools and utilities – there are many more options available, as follows:

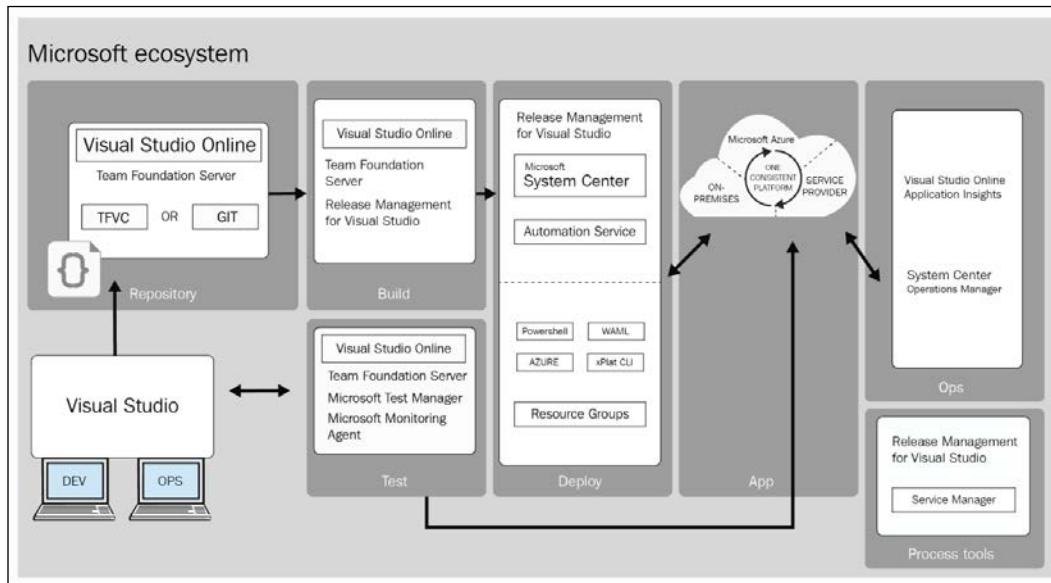
- The Jenkins, Hudson, Grunt, and Gradle tools for constructing the build pipeline
- Selenium for testing
- Chef, Puppet, Jenkins, Hudson, Grunt, and Gradle for deployment or configuration management
- Nagios for alerting and monitoring
- Jira and Redmine for managing processes



The following diagram shows the Microsoft tools and utilities that can be used for different phases in managing the application life cycle and DevOps in general. Again, this is just a small representation of all the tools and utilities – there are many more options available, such as the following:

- Azure DevOps build orchestration for constructing a build pipeline
- Microsoft Test Manager and Pester for testing
- DSC, PowerShell, and ARM templates for deployment or configuration management

- Log Analytics, Application Insights, and **System Center Operations Manager (SCOM)** for alerting and monitoring
- Azure DevOps and System Center Service Manager for managing processes:



Summary

DevOps is gaining a lot of traction and momentum in the industry. Most organizations have realized its benefits and are looking to implement DevOps. This is happening while most of them are moving to the cloud. Azure, as a cloud model, provides rich and mature DevOps services, making it easy for organizations to implement DevOps. In this chapter, we discussed DevOps along with its core practices, such as configuration management, continuous integration, continuous delivery, and deployment. We also discussed different cloud solutions based on PaaS, a virtual machine IaaS, and a container IaaS, along with their respective Azure resources, the build and release pipelines. Configuration management was the core part of this chapter, and we discussed DSC services from Azure Automation and using pull servers to configure virtual machines automatically. Finally, we covered Azure's openness and flexibility regarding the choice of languages, tools, and operating systems.

12

Azure OLTP Solutions Using Azure SQL Sharding, Pools, and Hybrid

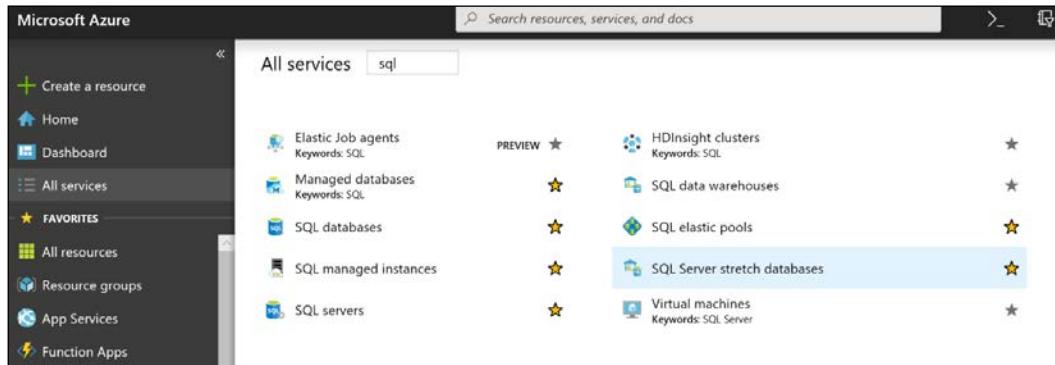
Azure provides both **Infrastructure as a Service (IaaS)** and **Platform as a Service (PaaS)** services. Both these types of services provide organizations with different levels and controls over storage, compute, and networks. Storage is the resource used when working with the storing and transmission of data. Azure provides lots of options for storing data, such as Azure storage blobs, tables, Cosmos DB, Azure SQL, Azure Data Lake, and more. While some of them are meant for big data storage, analytics, and presentation, there are others that are meant for applications that process transactions. Azure SQL is the primary resource in Azure that works with transaction data.

This chapter will focus on various aspects of using transaction data stores, such as Azure SQL and other opensource databases, typically used in **Online Transaction Processing (OLTP)** systems, and will cover the following topics:

- OLTP applications
- Relational databases
- Deployment models
- Azure SQL Database
- Elastic pools
- Managed instances
- SQL database pricing

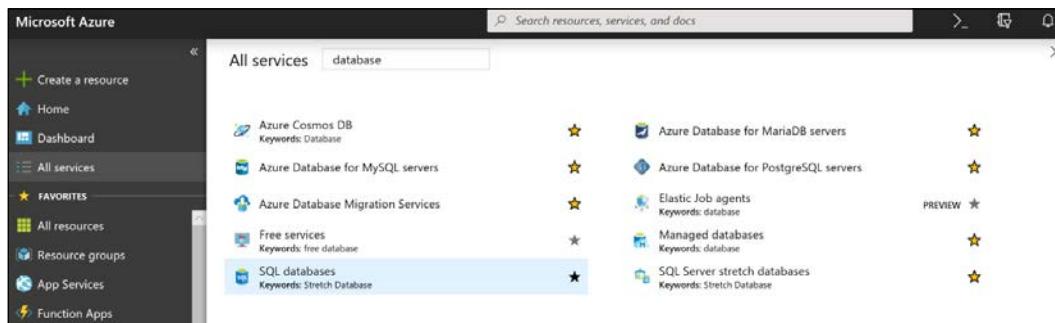
Azure cloud services

A search for `sql` in the Azure portal provides multiple results. I have marked some of them to show the resources that can be used directly for OLTP applications:



The previous screenshot shows the varied features and options available for creating SQL Server-based databases on Azure.

Again, a quick search for `database` on the Azure portal provides multiple resources, and the marked ones can be used for OLTP applications:



The previous screenshot shows resources provided by Azure that can host data in a variety of databases, including the following:

- MySQL databases
- MariaDB databases
- PostgreSQL databases
- Cosmos DB

OLTP applications

OLTP applications are applications that help in the processing and management of transactions. These applications perform data capture, data processing, retrieval, modification, and storage. However, it does not stop here. OLTP applications treat these data tasks as transactions. Transactions have a few important properties and OLTP applications adhere to these properties. These properties are grouped under the acronym **ACID**. Let's discuss these properties:

- **Atomicity:** This property states that a transaction must consist of statements and either all statements should complete successfully or no statement should be executed. If multiple statements are grouped together, these statements form the transaction. Atomicity means each transaction is treated as the lowest single unit of execution that either completes successfully or fails.
- **Consistency:** This property focuses on the state of data in a database. It dictates that any change in state should be complete and based on the rules and constraints of the database, and that partial updates should not be allowed.
- **Isolation:** This property states that there can be multiple concurrent transactions executed on the system and each transaction should be treated in isolation. One transaction should not know about or interfere with any other transaction. If the transactions were to be executed in sequence, by the end, the state of data should be the same as before.
- **Durability:** This property states that the data should be persisted and available, even after failure, once it is committed to the database. A committed transaction becomes a fact.

Relational databases

OLTP applications have generally been relying on relational databases for their transaction management and processing. Relational databases typically come in a tabular format consisting of rows and columns. The data model is converted into multiple tables where each table is connected to another table (based on rules) using relationships. This process is also known as normalization.

As mentioned before, Azure provides multiple relational databases, such as SQL Server, MySQL, and PostgreSQL.

Deployment models

There are two deployment models for deploying databases on Azure:

- Databases on Azure virtual machines (IaaS)
- Databases hosted as managed services (PaaS)

Databases on Azure virtual machines

Azure provides multiple SKUs for virtual machines. There are high-compute, high-throughput (IOPS) machines that are also available along with general-usage virtual machines. Instead of hosting a SQL Server, MySQL or any other database on on-premises servers, it is possible to deploy these databases on these virtual machines. The deployment and configuration of these databases is no different than that of on-premises deployments. The only difference is that the database is hosted on the cloud instead of on on-premises servers. Administrators must perform the same activities and steps that they normally would for on-premises deployment. Although this option is great when customers want full control over their deployment, there are models that can be more cost effective, scalable, and highly-available compared to this option.

The steps to deploy any database on Azure virtual machines are as follows:

1. Create a virtual machine with a size that caters to the performance requirements of the application.
2. Deploy the database on top of it.
3. Configure the virtual machine and database configuration.

This option does not provide any out-of-the-box high availability unless multiple servers are provisioned, and SQL Server is configured with an **AlwaysOn** configuration. It also does not provide any features for automatic scaling unless custom automation supports it.

Disaster recovery is also the responsibility of the customer, and they should deploy servers on multiple regions connected using global peering or network VPN gateways. It is possible for these virtual machines to be connected to an on-premises data center through site-to-site VPNs or ExpressRoute without having any exposure to the outside world.

These databases are also known as **unmanaged databases**.

Databases hosted as managed services

Databases hosted with Azure, other than virtual machines, are managed by Azure and are known as **managed services**. Managed services means that Azure provides management services for the databases. These managed services include the hosting of the database, ensuring that the host is highly available, ensuring that the data is replicated internally for availability during disaster recovery, ensuring scalability within the constraint of a chosen SKU, monitoring the hosts and databases and generating alerts for notifications or executing actions, providing log and auditing services for troubleshooting, and taking care of performance management and security alerts.

In short, there are a lot of services that customers get out of the box when using managed services from Azure, and they do not need to perform active management on these databases. In this chapter, we will look at Azure SQL in depth and provide information on other databases, such as Cosmos DB, MySQL, and Postgre.

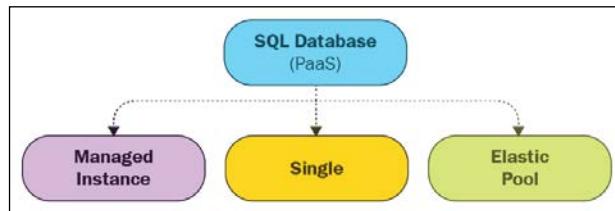
Azure SQL Database

Azure SQL server provides a relational database hosted as a PaaS. Customers can provision this service, bring their own database schema and data, and connect their applications to it. It provides all the features of SQL Server that you get when deploying on a virtual machine. These services do not provide a user interface to create tables and its schema, nor do they provide any querying capabilities directly. You should be using SQL Server Management Studio and the SQL CLI tools to connect to these services and directly work with them.

Azure SQL Database comes with three distinct deployment models:

- **Single Instance:** In this deployment model, a single database is deployed on a logical server. This involves the creation of two resources on Azure:
 - SQL logical server
 - SQL database
- **Elastic pool:** In this deployment mode, multiple databases are deployed on a logical server. Again, this involves the creation of two resources on Azure:
 - SQL logical server
 - SQL elastic database pool – this consists of all the databases

- **Managed Instance:** This is a relatively new deployment model from the Azure SQL team. This deployment reflects a collection of databases on a logical server providing complete control over the resources in terms of system databases. Generally, system databases are not visible in other deployment models, but they are available in the model. This model comes very close to the deployment of SQL Server on premises:



Application features

Azure SQL Database provides multiple application-specific features that cater to the different requirements of OLTP systems:

- **Columnar store:** This feature allows the storage of data to be in a columnar format rather than in a row format.
- **In-memory OLTP:** Generally, the data is stored in backend files in SQL and data is pulled from them whenever it is needed by the application. Contrary to this, in-memory OLTP puts all data in memory and there is no latency in reading the storage for data. Storing in-memory OLTP data on SSD provides the best possible performance for Azure SQL.
- All features of on-premises SQL Server.

Single Instance

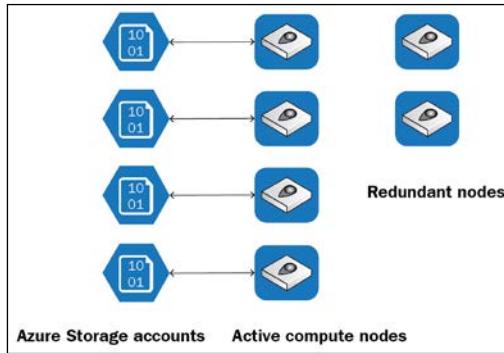
Single-Instance databases are hosted as a single database on a single logical server. These databases do not have access to the complete features provided by SQL Server.

High availability

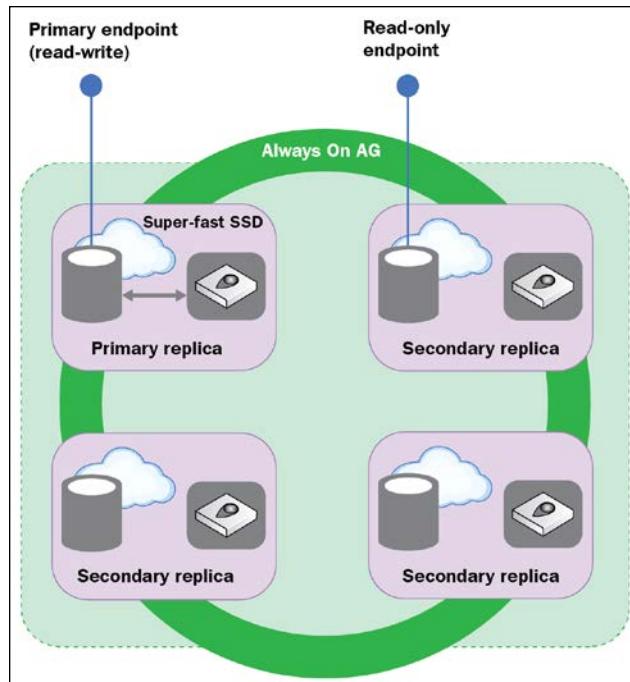
Azure SQL, by default, is 99.99% highly available. It has two different architectures for maintaining high availability based on the SKUs. For the Basic, Standard, and General SKUs, the entire architecture is broken down into the following two layers.

There is redundancy built in for both of these layers to provide high availability:

- Compute layer
- Storage layer:



For the Premium and business-critical SKUs, both compute and storage are on the same layer. High availability is achieved by replication of compute and storage deployed in a four-node cluster, using technology similar to SQL Server AlwaysOn Availability Groups:



Backups

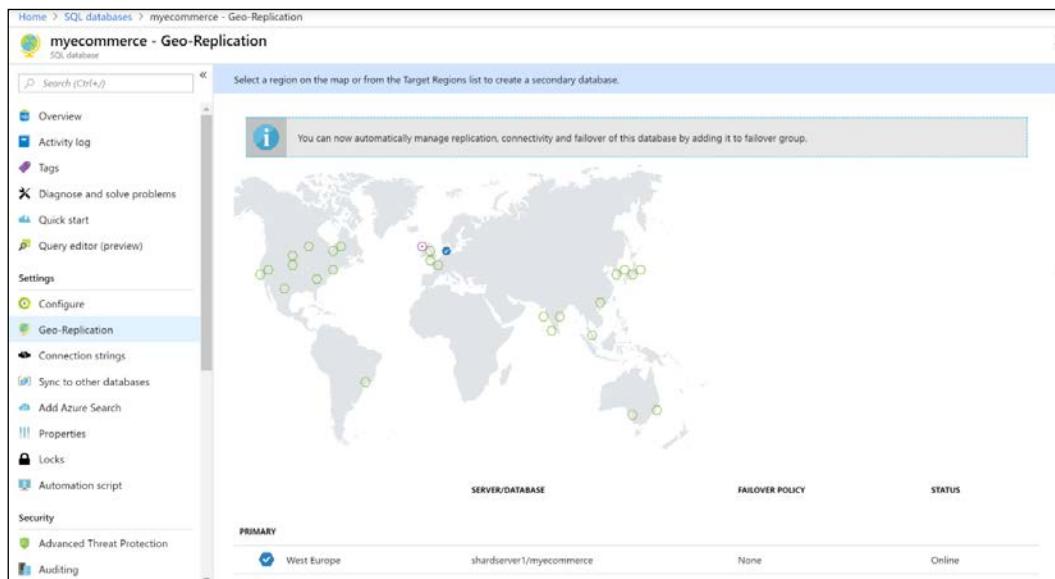
Azure SQL also provides features to automatically back up databases and store them on storage accounts. This feature is important especially in cases where a database becomes corrupt or a user accidentally deletes a table. This feature is available at the server level, as shown in the following screenshot:

The screenshot shows the Azure portal interface for managing backups. On the left, there's a sidebar with various options like Overview, Activity log, Access control (IAM), Tags, and Manage Backups (which is selected). The main area shows a table of databases with their backup configurations. One row for 'myecommerce' is highlighted, showing 'PITR BACKUPS' set to '35 days'. To the right, a modal window titled 'Configure policies' is open, specifically for 'Point In Time Restore Configuration'. It allows setting PITR backup retention in days (set to 35) and configuring long-term retention for weekly, monthly, and yearly backups. Buttons for 'Apply' and 'Cancel' are at the bottom of the modal.

Architects should prepare a backup strategy so that backups can be used in times of need. They should not be very old, nor should they be configured for very frequent backups. Based on business needs, a weekly backup or even a daily backup should be configured, or even more frequently than that, if required. These backups can be used for restoration purposes.

Geo-replication

Azure SQL also provides the benefit of being able to replicate the database to a different region, also known as a secondary region. The database at the secondary region can be read by applications. It allows for readable secondary databases. This is a great business continuity solution as the readable database is available at any point in time. With geo-replication, it is possible to have up to four secondaries of a database on different regions or the same region. These are readable databases. With geo-replication, it is also possible to failover to a secondary database in the case of a disaster. Geo-replication is configured at the database level, as shown in the following screenshot:



If you scroll down on this screen, the regions that can act as secondaries are listed, as shown in the following screenshot:

The screenshot shows the Azure portal interface for managing a SQL database named 'myecommerce'. The left sidebar contains navigation links for Overview, Activity log, Tags, Diagnose and solve problems, Quick start, Query editor (preview), Settings (with options like Configure, Geo-Replication, Connection strings, Sync to other databases, Add Azure Search, Properties, Locks, Automation script), Security (Advanced Threat Protection, Auditing), and Help & support.

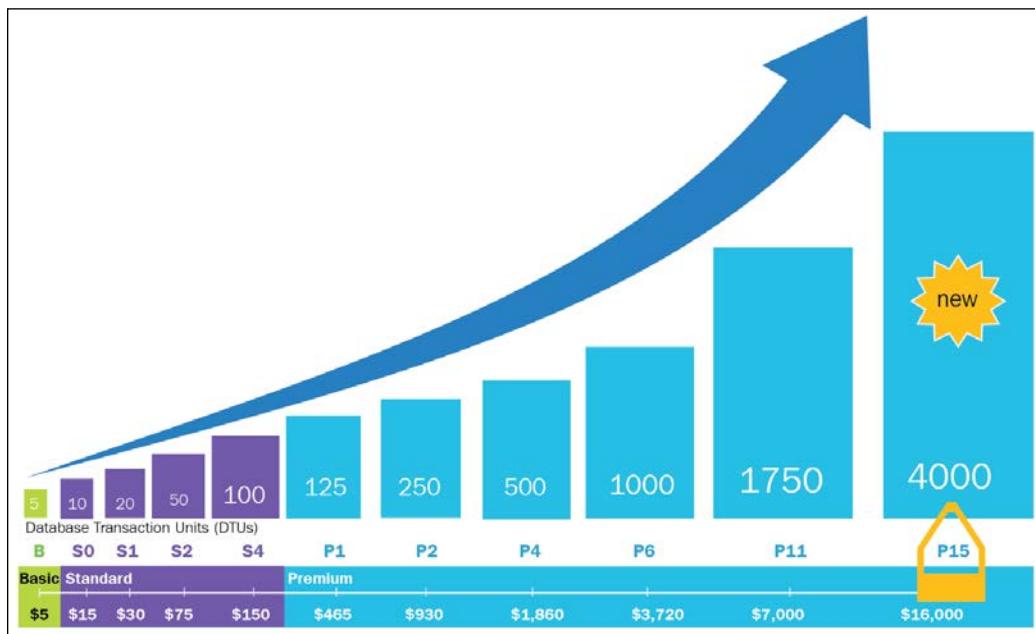
The main content area is titled 'myecommerce - Geo-Replication' and displays the following information:

- Primary:** West Europe, shardserver1/myecommerce, None, Online.
- Secondaries:** Geo-Replication is not configured.
- Target Regions:** A list of available regions including West US, West US 2, Central US, West Central US, South Central US, North Central US, Canada Central, East US, and Canada East.

A search bar at the top is labeled 'Search (Ctrl+ /)'.

Scalability

Azure SQL provides vertical scalability by adding more resources (such as compute, memory, and IOPS). This can be done by increasing the number of DTUs provisioned for it. The single-instance SQL provides features for manual scaling only:



Security

Security is an important factor for any database solution and service. Azure SQL provides enterprise-grade security for Azure SQL, and this section will list some of the important security features in Azure SQL.

Firewall

Azure SQL, by default, does not provide access to any requests. Source IP addresses should be explicitly whitelisted for access to SQL Server. There is an option to allow all Azure-based services access to the SQL database as well. This option includes virtual machines hosted on Azure.

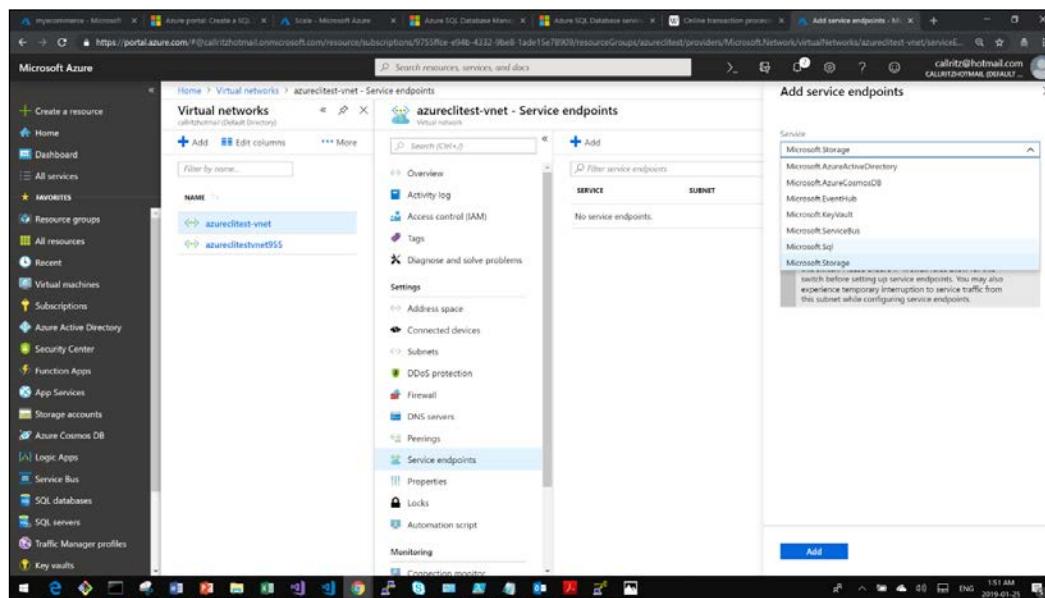
The firewall can be configured at the server level instead of the database level. The **Allow access to Azure services** option allows all services, including virtual machines, to access the database hosted on the logical server:

RULE NAME	START IP	END IP
ClientIPAddress_2019-1-25_...	223.230.50.55	223.230.50.55
ClientIPAddress_2019-1-16_...	182.66.149.67	182.66.149.67

Azure SQL Server on dedicated networks

Although access to SQL Server is generally available through the internet, it is possible that access to SQL Server can be limited to requests arising out of virtual networks. This is a relatively new feature in Azure. This helps in accessing data within SQL Server from application on another server of the virtual network without the request going through the internet.

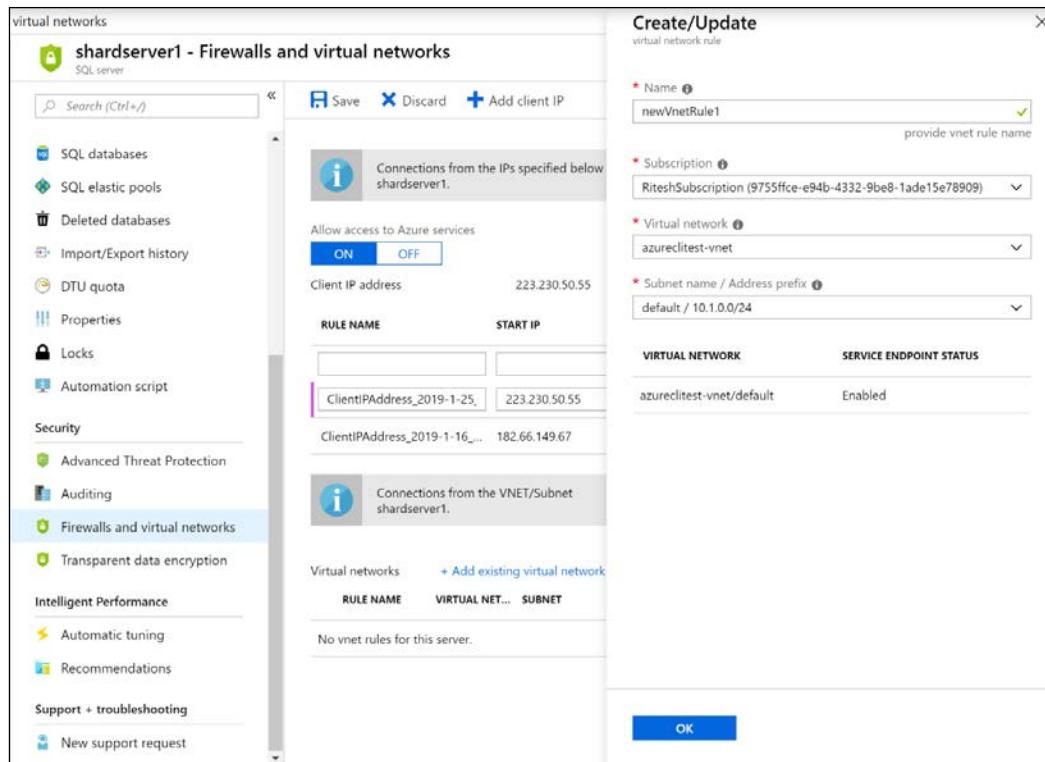
For this, a service endpoint of the `Microsoft.Sql` type should be added within the virtual network, and the virtual network should be in the same region as that of Azure SQL:



An appropriate subnet within the virtual networks should be chosen:

The screenshot shows the 'Service endpoints' blade for the 'azureclitest-vnet' virtual network. The left sidebar lists various settings like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Address space, Connected devices, Subnets, DDoS protection, Firewall, DNS servers, Peerings, and Service endpoints (which is selected). The main area shows a table with columns 'SERVICE' and 'SUBNET'. A note at the bottom states: 'rules using Azure public IP addresses will stop working with this switch. Please ensure IP firewall rules allow for this switch before setting up service endpoints. You may also experience temporary interruption to service traffic from this subnet while configuring service endpoints.' An 'Add' button is located at the bottom right.

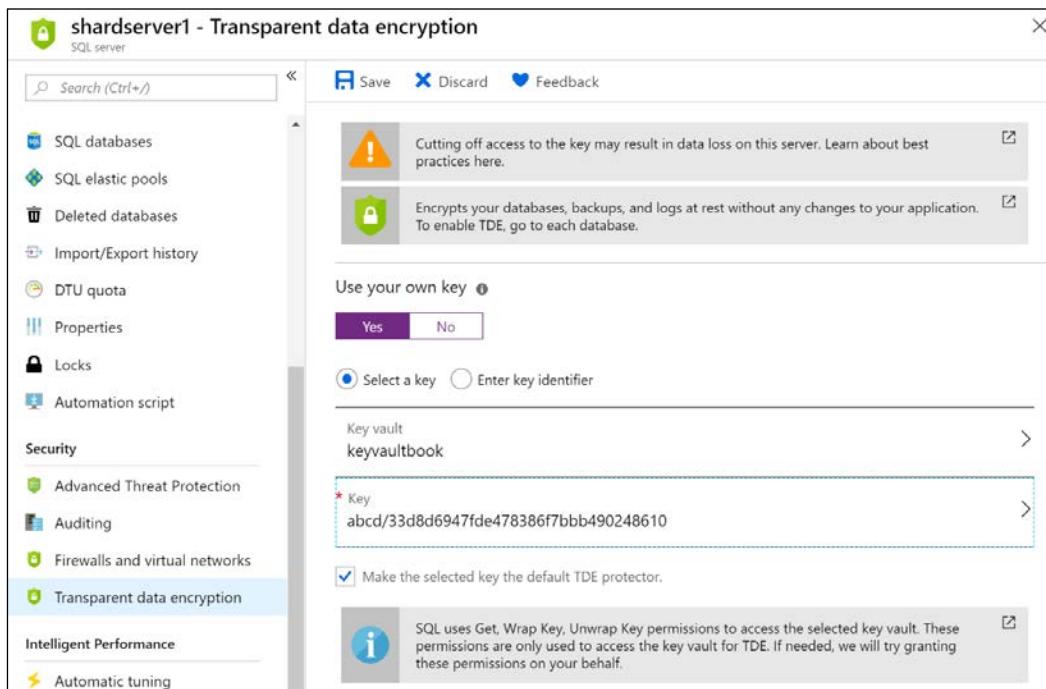
Finally, from the Azure SQL Server configuration blade, an existing virtual network should be added that has a **Microsoft.Sql** service endpoint enabled:



Encrypted databases at rest

The databases should be in an encrypted form when at rest. **Rest** here means the storage location of the database. Although you might not have access to SQL Server and its database, it is preferable to encrypt the database storage.

Databases on the filesystem can be encrypted using keys. These keys must be stored in Azure Key Vault and the vault must be available in the same region as that of Azure SQL Server. The filesystem can be encrypted by using the **Transparent data encryption** menu item of the SQL Server configuration blade and by selecting **Yes** for **Use your own key**. The key is an RSA 2048 key and must exist within the vault. SQL Server will decrypt the data at the page level when it wants to read it and send it to the caller, then it will encrypt it after writing to the database. No changes to the applications are required, and it is completely transparent to them:



Dynamic data masking

SQL Server also provides a feature that masks individual columns that contain sensitive data so that no one, apart from privileged users, can view the actual data by querying it in SQL Server Management Studio. The data will remain masked and will only be unmasked when an authorized application or user queries the table. Architects should ensure that sensitive data, such as credit card details, social security numbers, phone numbers, email addresses, and other financial details, should be masked.

Azure Active Directory integration

Another important security feature of Azure SQL is that it can be integrated with **Azure Active Directory (AD)** for authentication purposes. Without integrating with Azure AD, the only authentication mechanism available to SQL Server is via username and password authentication; that is, SQL authentication. It is not possible to use integrated Windows authentication. The connection string for SQL authentication consists of both the username and password in plaintext, which is not secure. Integrating with Azure AD, enabled authenticating application with Windows authentication, service principal name, or token-based. It is a good practice to used Azure SQL integrated with Azure AD.

There are other security features, such as advanced threat protection, auditing of the environment, and monitoring, that should be enabled on any enterprise-level Azure SQL deployments:

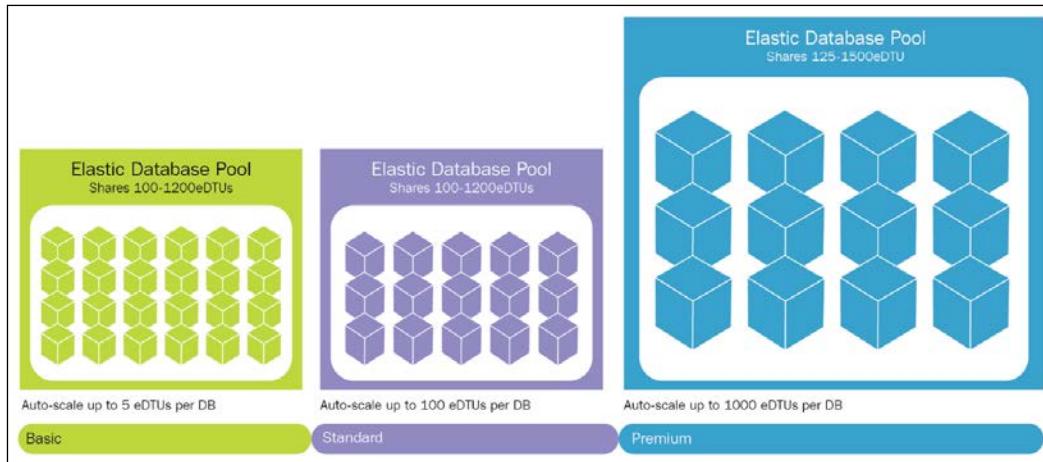
SCHEMA	TABLE	COLUMN	
dbo	Customers	LastName	<button>Add mask</button>
dbo	Salary	SalaryID	<button>Add mask</button>

Elastic pools

An elastic pool is a logical container that can host multiple databases in a single logical server. The SKUs available for elastic pools are as follows:

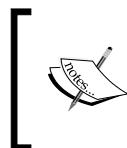
- Basic
- Standard
- Premium

The following screenshot shows the maximum amount of DTUs that can be provisioned for each SKU:



All the features discussed for Azure SQL Single Instances are available to elastic pools as well; however, horizontal scalability is an additional feature it provides with the help of sharding. Sharding refers to the vertical or horizontal partitioning of data and the storage of it in separate databases. It is also possible to have auto-scaling of individual databases in an elastic pool by consuming more DTUs than are actually allocated to that database.

Elastic pools also provide another advantage in terms of cost. We will see in a later section that Azure SQL is priced using the concept of DTUs, and DTUs are provisioned as soon as the SQL Server service is provisioned. DTUs are charged for irrespective of whether those DTUs are consumed. If there are multiple databases, then it is possible to put these databases into elastic pools and they can share the DTUs among them.

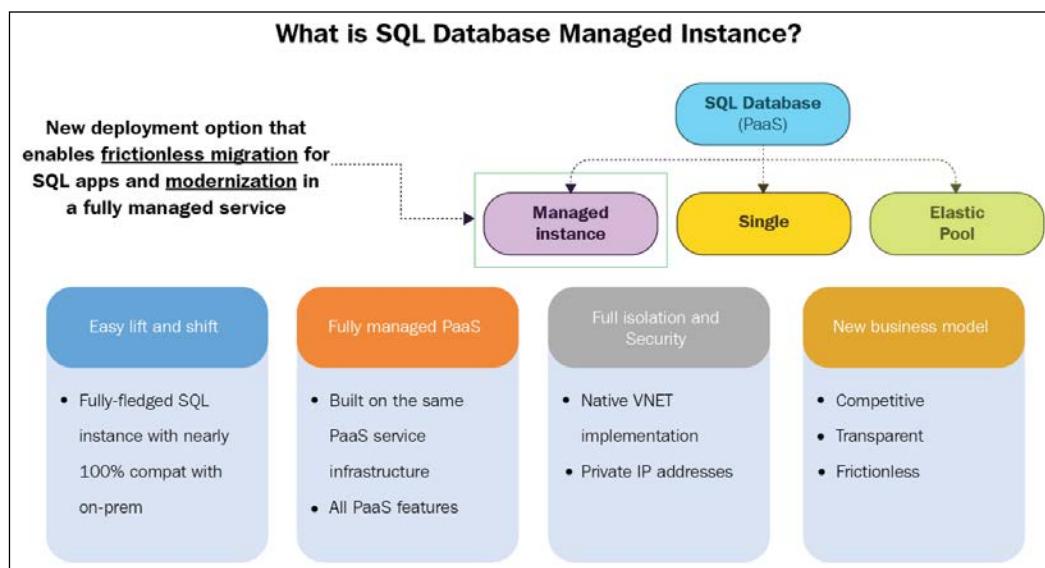


All information for implementing sharding with Azure SQL elastic pools has been provided at <https://docs.microsoft.com/en-us/azure/sql-database/sql-database-elastic-scale-introduction>.

Managed Instance

Managed Instance is a unique service that provides a managed SQL server similar to what's available on on-premises servers. Users have access to master, model, and other system databases. The use of Managed Instance is suitable when there are multiple databases and customers migrating their instances to Azure. Managed Instance consists of multiple databases.

Azure SQL Database provides a new deployment model known as Azure SQL Database Managed Instance that provides almost 100% compatibility with the SQL Server Enterprise Edition Database Engine. This model provides a native VNet implementation addressing the usual security issues and is a highly recommended business model for on-premises SQL Server customers. Managed Instance allows existing SQL Server customers to lift and shift their on-premises applications to the cloud with minimal application and database changes, while preserving all PaaS capabilities at the same time. These PaaS capabilities drastically reduce the management overhead and total cost of ownership, as shown in the following screenshot:



The key features of Managed Instance are shown in the following screenshot:

The key features of Managed Instance are shown in the following table:	
Feature	Description
SQL Server version / build	SQL Server Database Engine (latest stable)
Managed automated backups	Yes
Built-in instance and database monitoring and metrics	Yes
Automatic software patching	Yes
The latest Database Engine features	Yes
Number of data files (ROWS) per the database	Multiple
Number of log files (LOG) per database	1
VNet - Azure Resource Manager deployment	Yes
VNet - Classic deployment model	No
Portal support	Yes
Built-in Integration Service (SSIS)	No - SSIS is a part of Azure Data Factory PaaS
Built-in Analysis Service (SSAS)	No - SSAS is separate PaaS
Built-in Reporting Service (SSRS)	No - use Power BI or SSRS IaaS

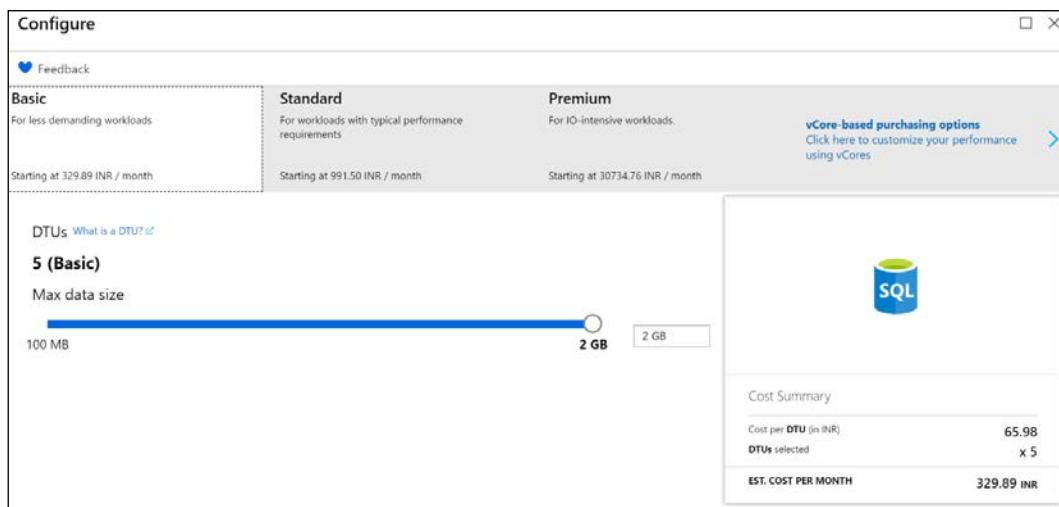
SQL database pricing

Azure SQL previously had just one pricing model – a model based on **Database Throughput Units (DTUs)** and a more recent alternative pricing model based on vCPUs have also been launched.

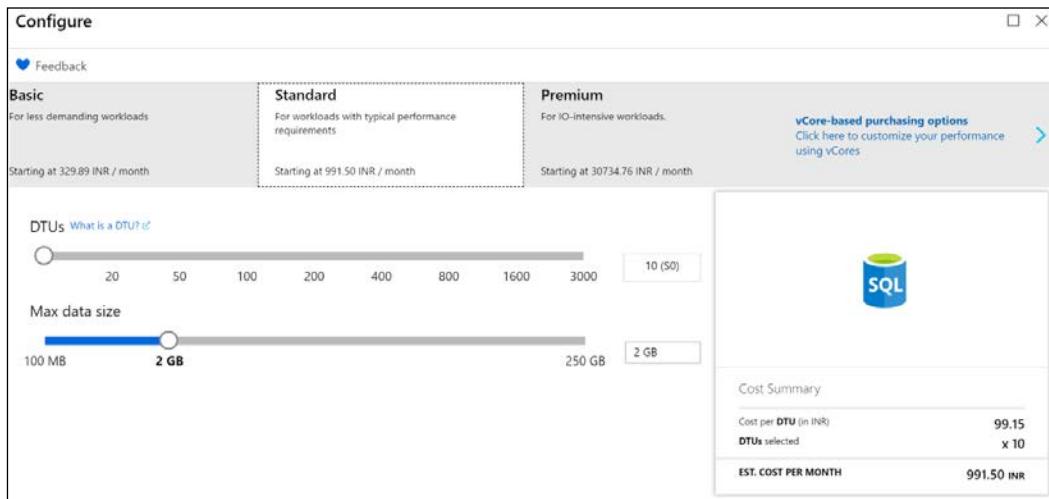
DTU-based pricing

The DTU is the smallest unit of performance measure for Azure SQL. Each DTU corresponds to a certain amount of resources. These resources include storage, CPU cycles, IOPS, and network bandwidth. For example, a single DTU might provide three IOPS, a few CPU cycles, and an IO latency of 5 ms for read operations and 10 ms for write operations.

Azure SQL provides multiple SKUs for creating databases, and each of these SKUs has defined constraints for the maximum amount of DTUs. For example, the Basic SKU provides just 5 DTUs with a maximum 2 GB of data, as shown in the following screenshot:



On the other hand, the standard SKU provides anything between 10 DTUs and 300 DTUs with a maximum of 250 GB of data. As you can see here, each DTU costs around 999 rupees or around \$1.40:



A comparison between these SKUs in terms of performance and resources is provided by Microsoft, as shown in the following screenshot:

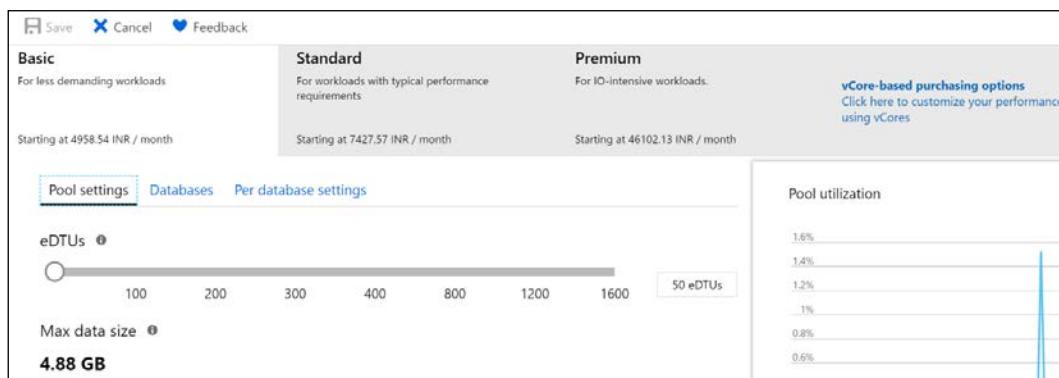
	Basic	Standard	Premium
Target workload	Development and production	Development and production	Development and production
Uptime SLA	99.99%	99.99%	99.99%
Backup retention	7 days	35 days	35 days
CPU	Low	Low, Medium, High	Medium, High
IO throughput (approximate)	2.5 IOPS per DTU	2.5 IOPS per DTU	48 IOPS per DTU
IO latency (approximate)	5 ms (read), 10 ms (write)	5 ms (read), 10 ms (write)	2 ms (read/write)
Columnstore indexing	N/A	S3 and above	Supported
In-memory OLTP	N/A	N/A	Supported

Once you provision a certain number of DTUs, the backend resources (CPU, IOPS, and memory) are allocated and are charged for whether they are consumed or not. If more DTUs are procured than are actually needed, it leads to wastage, while there would be performance bottlenecks if not enough DTUs are provisioned.

Azure provides elastic pools for this purpose as well. As we know, there are multiple databases in an elastic pool and DTUs are assigned to elastic pools instead of individual databases. It is possible for all databases within the pool to share the DTUs. This means that if a database has low utilization and is consuming 5 DTUs, there will be another database consuming 25 DTUs in order to compensate.

It is important to note that, collectively, DTU consumption cannot exceed the amount of DTUs provisioned for the elastic pool. Moreover, there is a minimum amount of DTUs that should be assigned to each database within the elastic pool, and this minimum DTU count is pre-allocated for the database.

An elastic pool comes with its own SKUs:



Also, there is a limitation on the maximum amount of databases that can be created within a single elastic pool.

vCPU-based pricing

This is the new pricing model for Azure SQL. In this pricing model, instead of identifying the amount of DTUs required for an application, it provides options to procure the number of **virtual CPUs (vCPUs)** allocated for the server. A vCPU is a logical CPU with attached hardware such as storage, memory, and CPU cores.

In this model, there are three SKUs: **General Purpose**, **Hyperscale**, and **Business Critical**, with a varied number of vCPUs and resources available. This pricing is available for all SQL deployment models:

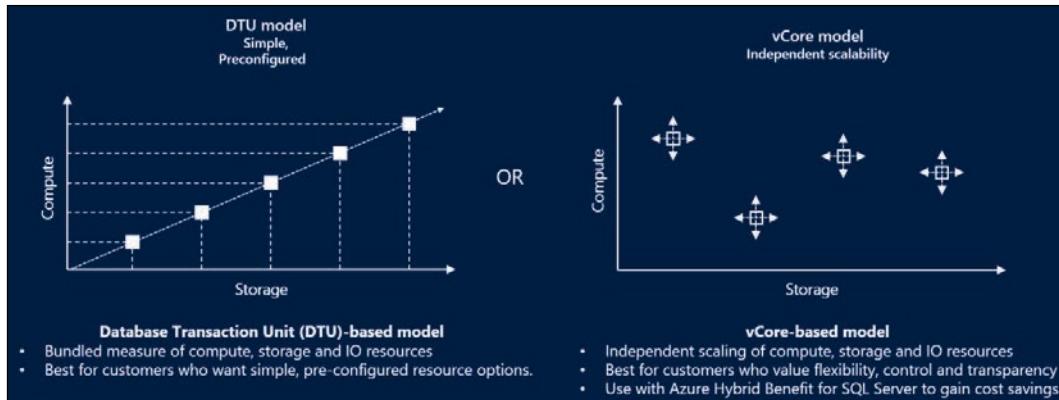
The screenshot shows the 'Configure' interface for Azure SQL. It displays three SKU options: General Purpose, Hyperscale, and Business Critical. The Hyperscale option is selected. Under Compute Generation, 'Gen5' is chosen with 2 vCores and 32 GB of memory. The 'Cost Summary' section shows the estimated cost per month.

SKU	Compute Generation	vCores	Max storage	Allocated Log Storage	Cost per Month (INR)
General Purpose	Gen4	up to 24 vCores	up to 168 GB memory		
Hyperscale	Gen5	2 vCores	32 GB	9.6 GB	8233.91
Business Critical					9.05
		x 2	x 41.6		16844.11 INR

How to choose the appropriate pricing model

Architects should be able to choose an appropriate pricing model for Azure SQL. DTUs are a great mechanism for pricing where there is a usage pattern applicable and available for the database. Since resource availability in the DTU scheme of things is linear, as shown in the next diagram, it is quite possible for the usage to be more memory intensive than CPU intensive. In such cases, it is possible to choose different levels of CPU, memory, and storage for a database. In DTUs, resources come packaged, and it is not possible to configure these resources at a granular level. With a vCPU model, it is possible to choose different levels of memory and CPU for different databases. If the usage pattern for an application is known, using the vCPU pricing model could be a better option compared to the DTU model. In fact, the vCPU model also provides the benefit of hybrid licenses if an organization already has on-premises SQL Server licenses. There is a discount of up to 30% provided to these SQL Server instances.

In the following screenshot, you can see from the left-hand graph that as the amount of DTUs increases, resource availability also grows linearly; however, with vCPU pricing (in the right-hand graph), it is possible to choose independent configurations for each database:



Summary

Azure SQL is one the flagship services of Azure. Millions of customers are using this service today and it provides all the enterprise capabilities that are needed for a mission-critical database management system. There are multiple deployment types for Azure SQL, such as Single Instance, Managed Instance, and elastic pools. Architects should do a complete assessment of their requirements and choose the appropriate deployment model. After they choose a deployment model, they should choose a pricing strategy between DTUs and vCPUs. They should also configure all the security, availability, disaster recovery, monitoring, performance, and scalability requirements in Azure SQL with regard to data.

13

Azure Big Data Solutions with Azure Data Lake Storage and Data Factory

Big data has been gaining significant traction over the last few years. Specialized tools, software, and storage are required to handle it. Interestingly, these tools, platforms, and storage options were not available as services a few years back. However, with new cloud technology, Azure provides numerous tools, platforms, and resources to create big data solutions easily.

The following topics will be covered in this chapter:

- Data integration
- Extract-Transform-Load (ETL)
- Data Factory
- Data Lake Storage
- Migrating data from Azure Storage to Data Lake Storage

Data integration

We are well aware of how integration patterns are used for applications; applications that are composed of multiple services are integrated together using a variety of patterns. However, there is another paradigm that is a key requirement for many organizations, which is known as data integration. This surge in data integration has primarily happened during the last decade, when the generation and availability of data has been incredibly high. The velocity, variety, and volume of data being generated has increased drastically, and there is data almost everywhere.

Every organization has many different types of applications, and they all generate data in their own proprietary format. Often, data is also purchased from the marketplace. Even during mergers and amalgamations of organizations, data needs to be migrated and combined.

Data integration refers to the process of bringing data from multiple sources and generating a new output that has more meaning and usability.

There is a definite need for data integration for the following scenarios:

- For migrating data from a source or group of sources to a target destination. This is needed to make data available in different formats to different stakeholders and consumers.
- With the rapid availability of data, organizations want to derive insights from them. They want to create solutions that provide insights, data from multiple sources should be merged, cleaned, augmented, and store in a data warehouse.
- For generating real-time dashboards and reports.
- For creating analytics solutions.

Application integration has a runtime behavior when users are consuming the application; for example, in the case of credit card validation and integration. On the other hand, data integration happens as a backend exercise and is not directly linked to user activity.

ETL

A very popular process known as ETL helps in building a target data source for house data that is consumable by applications. Generally, the data is in a raw format, and to make it consumable, the data should go through the following three distinct phases:

- **Extract:** During this phase, data is extracted from multiple places. For instance, there could be multiple sources and they all need to be connected together in order to retrieve the data. Extract phases typically use data connectors consisting of connection information related to the target data source. They might also have temporary storage to bring the data from the data source and store it for faster retrieval. This phase is responsible for the ingestion of data.
- **Transform:** The data that is available after the extract phase might not be directly consumable by applications. This could be for a variety of reasons; for example, the data might have irregularities, there might be missing data, or there might be erroneous data. Or, there might even be data that is not needed at all.

Alternatively, the format of the data might not be conducive for consumption by the target applications. In all of these cases, transformation has to be applied to the data in such a way that it can be efficiently consumed by applications.

- **Load:** After transformation, data should be loaded to the target data source in a format and schema that enables faster, easier, and performance-centric availability for applications. Again, this, typically, consists of data connectors for destination data sources and loading data into them.

A primer on Data Factory

Data Factory is a fully managed, highly available, highly scalable, and easy-to-use tool for creating integration solutions and implementing ETL phases. Data Factory helps you to create new pipelines in a drag and drop fashion using a user interface, without writing any code; however, it still provides features to allow you to write code in your preferred language.

There are a few important concepts to learn about before using the Data Factory service, which we will be exploring in more detail in the following sections:

- **Activities:** Activities are individual tasks that enable the execution and processing of logic within a Data Factory pipeline. There are multiple types of activities. There are activities related to data movement, data transformation, and control activities. Each activity has a policy through which it can decide the retry mechanism and retry interval.
- **Pipelines:** Pipelines in Data Factory are composed of groups of activities and are responsible for bringing activities together. Pipelines are the workflows and orchestrators that enable the execution of the ETL phases. Pipelines allow the weaving together of activities and allow the declaration of dependencies between them. By using dependencies, it is possible to execute some tasks in parallel and other tasks in sequence.
- **Datasets:** Datasets are the sources and destinations of data. These could be Azure Storage accounts, Data Lake Storage, or a host of other sources.
- **Linked services:** These are services that contain the connection and connectivity information for datasets and are utilized by individual tasks for connecting to them.

- **Integration runtime:** The main engine that is responsible for the execution of Data Factory is called the integration runtime. The integration runtime is available on the following three configurations:
 - **Azure:** In this configuration, the Data Factory executes on the compute resources that are provided by Azure.
 - **Self-hosted:** The Data Factory in this configuration executes when you bring your own compute resources. This could be through on-premises or cloud-based virtual machine servers.
 - **Azure SQL Server Integration Services (SSIS):** This configuration allows for the execution of traditional SSIS packages written using SQL Server.
- **Versions:** Data Factory comes with two different versions. It is important to understand that all new developments will happen on V2, and that V1 will stay as it is, or fade out at some point. V2 is preferred because of the following reasons:
 - It provides the capability to execute SQL Server integration packages.
 - It has enhanced functionalities compared to V1.
 - It comes with enhanced monitoring, which is missing in V1.

A primer on Data Lake Storage

Azure Data Lake Storage provides storage for big data solutions. It is especially designed for storing the large amounts of data that are typically needed in big data solutions. It is an Azure-provided managed service and is, therefore, completely managed by Azure. Customers need only bring their data and store it in a Data Lake.

There are two versions of Azure Data Lake Storage: version 1 (Gen1) and the current version, version 2 (Gen2). Gen2 has all the functionality of Gen1, but with the difference that it is built on top of Azure Blob Storage.

As Azure Blob Storage is highly available, can be replicated multiple times, is disaster ready, and is low in cost, these benefits are transferred to Gen2 Data Lake. Data Lake can store any kind of data, including relational, non-relational, filesystem-based, and hierarchical data.

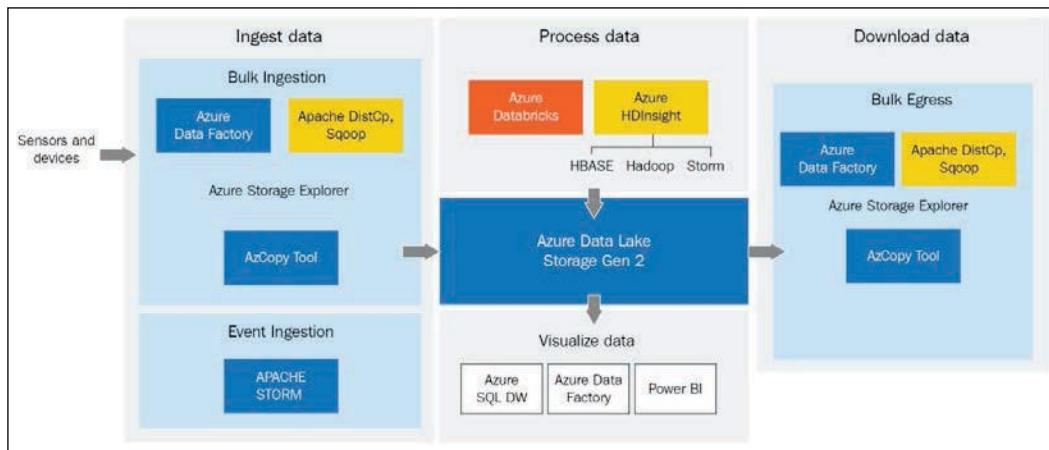
Creating a Data Lake Gen2 instance is as simple as creating a new Storage account. The only change that needs to be done is to enable the hierarchical namespace from the Advanced tab of your Storage account.

Understanding big data processing

There are four distinct phases in big data processing, as follows:

- The ingestion of big data
- The processing of big data
- Storing the resultant data in a data store
- The presentation of insights and data

These four stages are summarized in the following diagram; we will be looking into each of these phases in the next sections:



Ingesting data

The data ingestion phase brings data into the Data Lake. This data can be streamed in real time using tools such as Azure HDInsight Storm.

On-premises data can be also migrated to the Data Lake.

Processing data

The data in the Data Lake can be processed and transformed using tools such as Azure Databricks and HDInsight, and any output that is typically demanded by applications can be generated.

Storing data

In this stage, the transformed data is moved to data stores such as data warehouses, Cosmos DB, and Azure SQL such that it can be consumed by applications directly.

Presenting data

The data insights and wisdom that are generated can be used for visualization, real-time dashboards, reports, and notifications.

Migrating data from Azure Storage to Data Lake Gen2 Storage

In this section, we will be migrating data from Azure Blob Storage to another Azure container of the same Azure Blob Storage instance, and we will also migrate data to an Azure Gen2 Data Lake instance using an Azure Data Factory pipeline. The following sections detail the steps needed for creating such an end-to-end solution.

Preparing the source storage account

Before we can create Azure Data Factory pipelines and use them for migration, we need to create a new Storage account, consisting of a number of containers, and upload the data files. In the real world, these files and the storage connection would already be prepared. The first step for creating a new Azure Storage account is to create a new resource group or choose an existing resource group within an Azure Subscription.

Provisioning a new resource group

Every resource in Azure is associated with a resource group. Before we provision an Azure Storage account, we should be creating a resource group which will host the Storage account. The steps for creating a resource group are mentioned here. It is to be noted that a new resource group can be created while provisioning an Azure Storage account or an existing resource group can be used:

1. Navigate to the Azure portal, log in, and click on **+ Create a resource**; then, search for **Resource group**.
2. Select **Resource group** from the search results and create a new resource group. Provide a name and choose an appropriate location. Note that all the resources should be hosted in the same resource group and location so that it is easy to delete them.

Provisioning a Storage account

In this section, we will go through the steps of creating a new Azure Storage Account. This Storage Account will be the source data source from which data will be migrated.

1. Click on **+ Create a resource** and search for **Storage Account**. Select **Storage Account** from the search results and then create a new storage account.
2. Provide a name and location, and then select a subscription based on the resource group that was created earlier.
3. Select **StorageV2 (general purpose v2)** for **Account kind**, **Standard** for **Performance**, and **Locally-redundant storage (LRS)** for **Replication**, as demonstrated in the following screenshot:

Create storage account

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

PROJECT DETAILS

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

* Subscription: RiteshSubscription

* Resource group: BigDataSolutions

[Create new](#)

INSTANCE DETAILS

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

* Storage account name: adfsamplesourcedata

* Location: East US

Performance: Standard (selected) Premium

Account kind: StorageV2 (general purpose v2)

Replication: Locally-redundant storage (LRS)

Access tier (default): Cool (selected) Hot

4. Now create a couple of containers within the storage account. **rawdata** contains the files that will be extracted by the Data Factory pipeline and will act as the source dataset, while **finaldata** will contain files that the Data Factory pipelines will write data to and will act as the destination dataset:

The screenshot shows the Azure Storage Blobs interface for the 'adfsamplesourcedata' storage account. On the left, there's a sidebar with links: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, and Events. The main area has a search bar at the top labeled 'Search (Ctrl+ /)'. Below it are buttons for '+ Container', 'Refresh', and 'Delete'. A message says 'Storage account: adfsamplesourcedata'. There's another search bar labeled 'Search containers by prefix'. A table lists containers with columns 'NAME' and 'Actions'. The 'finaldata' container is highlighted with a dashed blue border, and its name is also in a highlighted box. The 'rawdata' container is listed below it.

5. Upload a data file (this file is available with the source code) to the rawdata container, as follows:

The screenshot shows the Azure Storage Blobs upload interface for the 'rawdata' container. At the top, there are buttons for 'Upload', 'Refresh', 'Delete', and 'Acquire lease'. Below that, it says 'Location: rawdata'. There's a search bar labeled 'Search blobs by prefix (case-sensitive)'. A table lists blobs with a 'NAME' column. The file 'products.csv' is shown in the list, with its icon and name visible.

Creating a new Data Lake Gen2 service

As we already know, the Data Lake Gen3 service is built on top of the Azure Storage account. Because of this, we will be creating a new Storage account in the same way that we did earlier – with the only difference being the selection of **Enabled** for **Hierarchical namespace** in the **Advanced** tab of the new Azure Storage account. This will create the new Data Lake Gen2 service:

Create storage account

Basics **Advanced** Tags Review + create

SECURITY

Secure transfer required i Disabled Enabled

VIRTUAL NETWORKS

Allow access from All networks Selected network
i All networks will be able to access this storage account. [Learn more](#)

DATA LAKE STORAGE GEN2 (PREVIEW)

Hierarchical namespace i Disabled Enabled

Provision Azure Data Factory Pipeline

Now that we have already provisioned both the resource group and Azure Storage Account, it's time to create a new Data Factory Pipeline:

1. Create a new Data Factory pipeline by selecting V2 and by providing a name and location along with a resource group and subscription selection, as shown in the following screenshot:

New data factory □ X

* Name i
 ✓

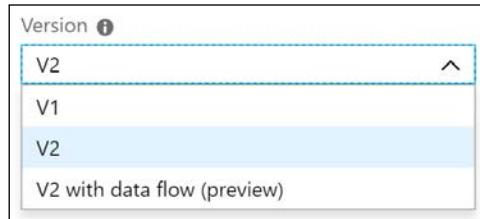
* Subscription
 ▾

* Resource Group i
 Create new Use existing
 ✓

Version i
 ▾

* Location i
 ▾

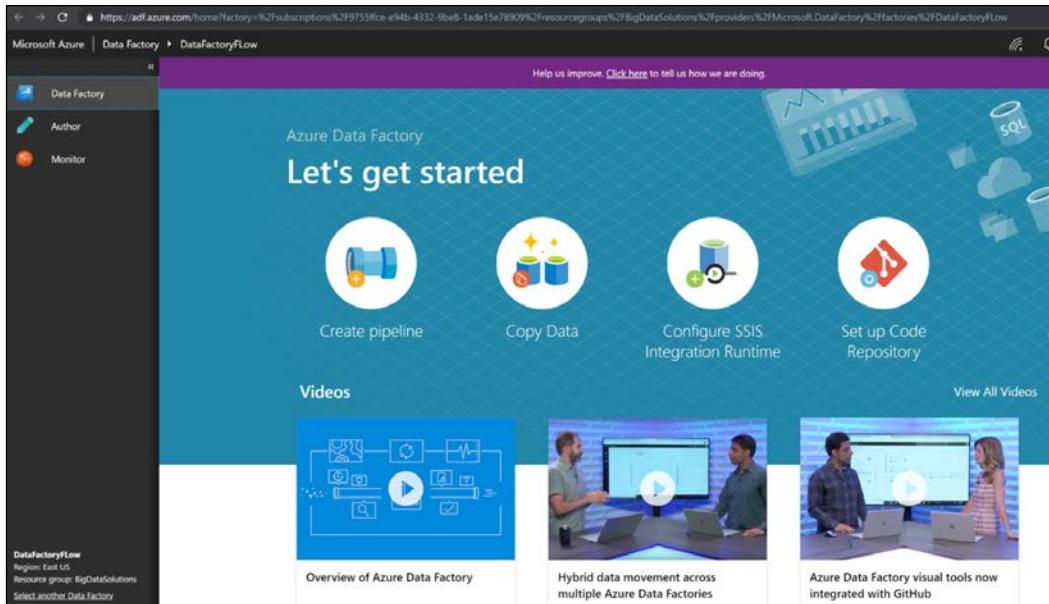
2. Data Factory has three different versions, as shown in the following screenshot. We've already discussed **V1** and **V2**. **V2 with data flow** is in preview, so you will need to get your subscription whitelisted before using it:



3. Once the Data Factory resource is created, click on the **Author & Monitor** link from the central page:

A screenshot of the Azure portal showing the properties of a Data Factory resource. The resource group is "BigDataSolutions", status is "Succeeded", location is "East US", subscription is "RiteshSubscription", and subscription ID is "9755ffce-e94b-4332-9be8-1ade15e78909". Below the properties, there are two links: "Documentation" and "Author & Monitor". The "Author & Monitor" link is highlighted with a red box.

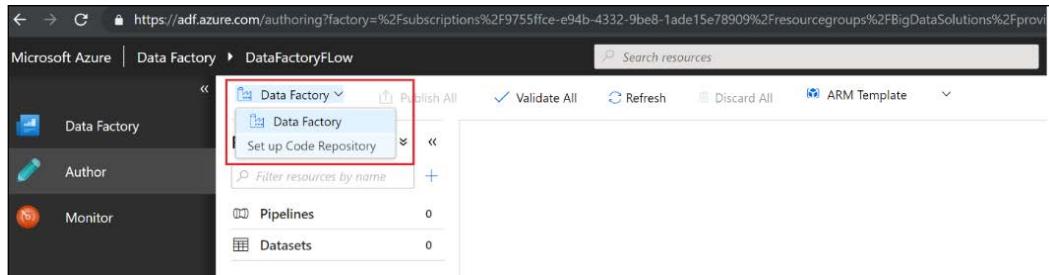
4. This will open another window, consisting of the Data Factory designer for the pipelines:



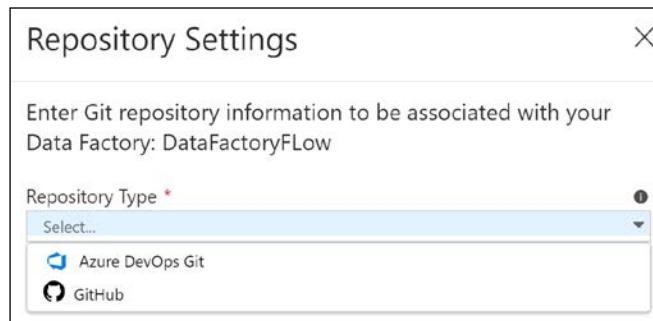
Repository settings

Before creating any Data Factory artifacts, such as datasets and pipelines, it is a good idea to set up the code repository for hosting files related to Data Factory:

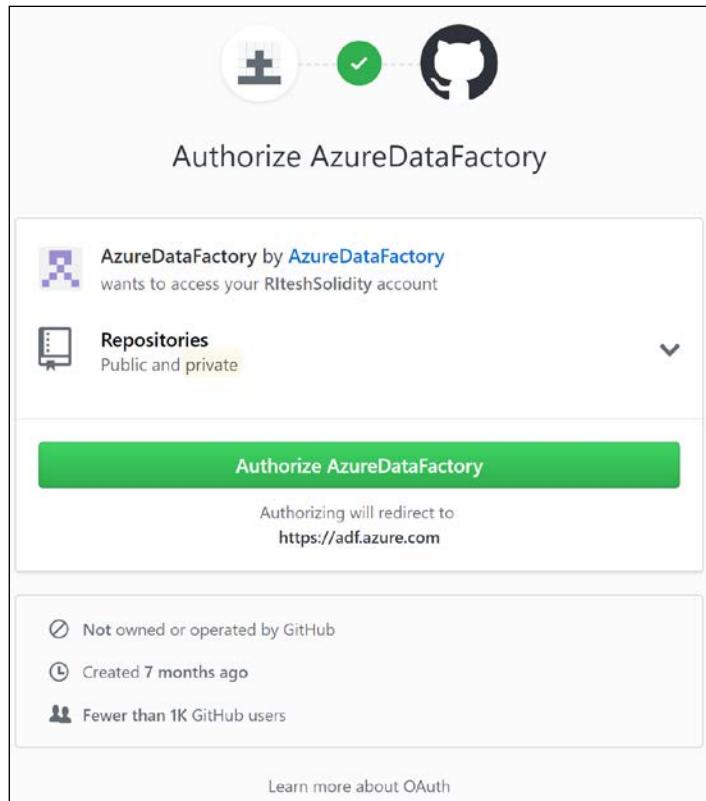
1. Click on the **Data Factory** dropdown from the top menu:



2. From the resultant blade, select any one of the repositories that you want to store Data Factory code files in. In this case, let's select **GitHub**, as demonstrated in the following screenshot:



3. It will then ask for authorization to the GitHub account.
4. Log into GitHub with your credentials and provide permissions to the Azure Data Factory service:



5. Create or reuse an existing repository from GitHub. In our case, we are creating a new repository named ADF. If you are creating a new repository, ensure that it is already initialized, otherwise, the Data Factory repository setting will complain:

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: RiteshSolidity / Repository name *: ADF

Great repository names are short and memorable. Need inspiration? How about [verbose-winner](#).

Description (optional): Repository for ADF demo

 **Public**
Anyone can see this repository. You choose who can commit.

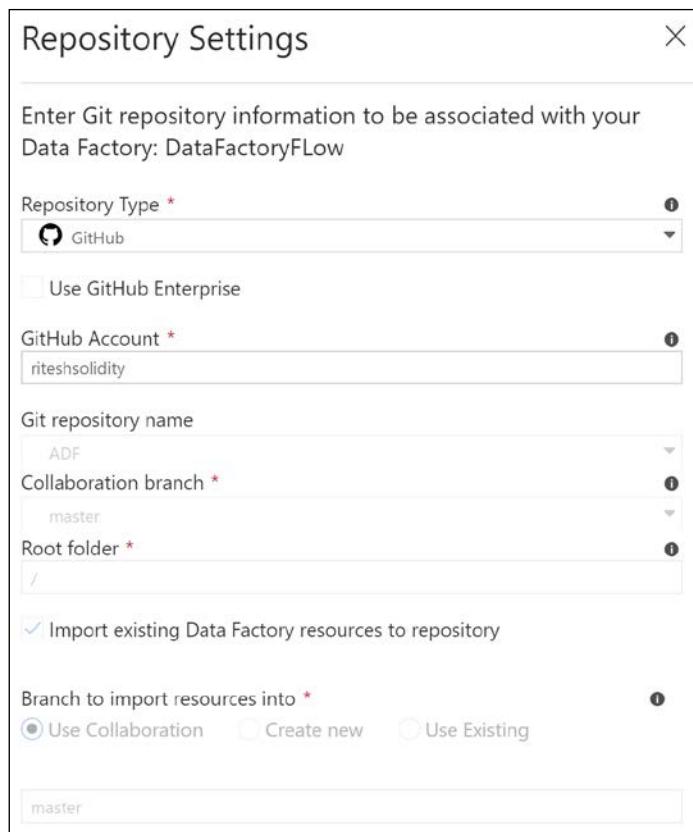
 **Private**
You choose who can see and commit to this repository.

Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

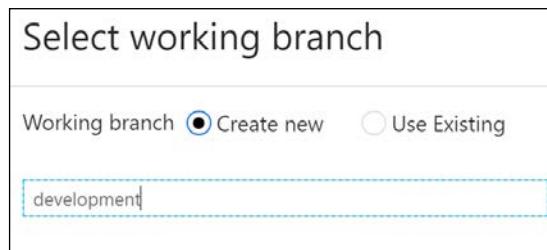
Add .gitignore: None ▾ | Add a license: None ▾ | ⓘ

Create repository

- Now, we can move back to the Data Factory pipeline window and ensure that the repository settings are appropriate, including **Git repository name**, **Collaboration branch**, **Root folder**, and **Branch**, which will act as the main branch:



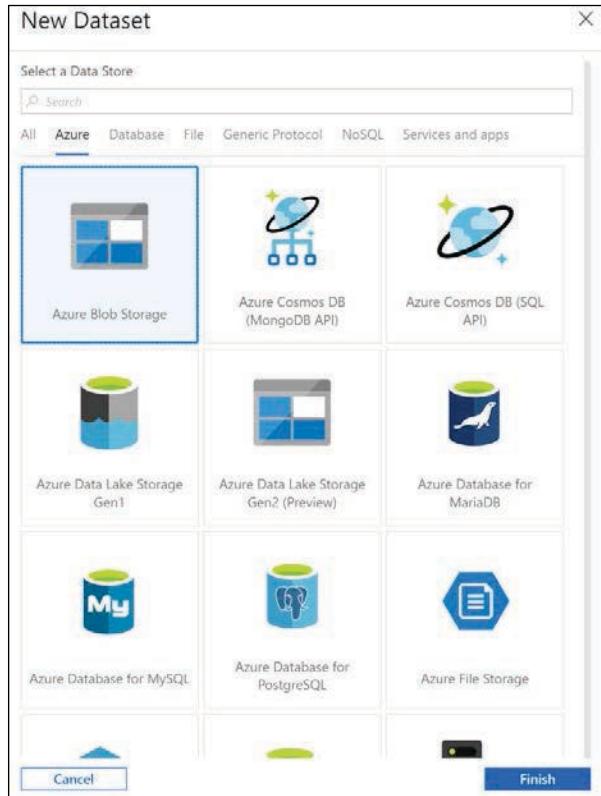
- On the next screen, create a new branch on which developers will be working. In our case, a new development branch is created, as demonstrated in the following screenshot:



Creating the first dataset

Now we can go back to the Data Factory pipeline. First, create a new dataset that will act as the source dataset. It will be the first storage account that we create and upload the sample products.csv file to:

1. Click on **+ Dataset** from the top menu and select **Azure Blob Storage**:



2. Then, on the resultant lower pane in the **General** tab, provide a name for the dataset:

General	Connection 1	Schema	Parameters
Name *	InputBlobStorageData		
Description			
Annotations	+ New		

3. From the **Connection** tab, create a new linked service according to the configuration of the following screenshot:

New Linked Service X

Name *

Description

Connect via integration runtime * i

Authentication method ▼

Account selection method From Azure subscription Enter manually i

Azure subscription ▼

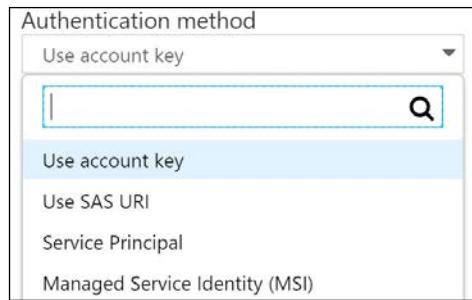
Storage account name * ▼

Additional connection properties
+ New

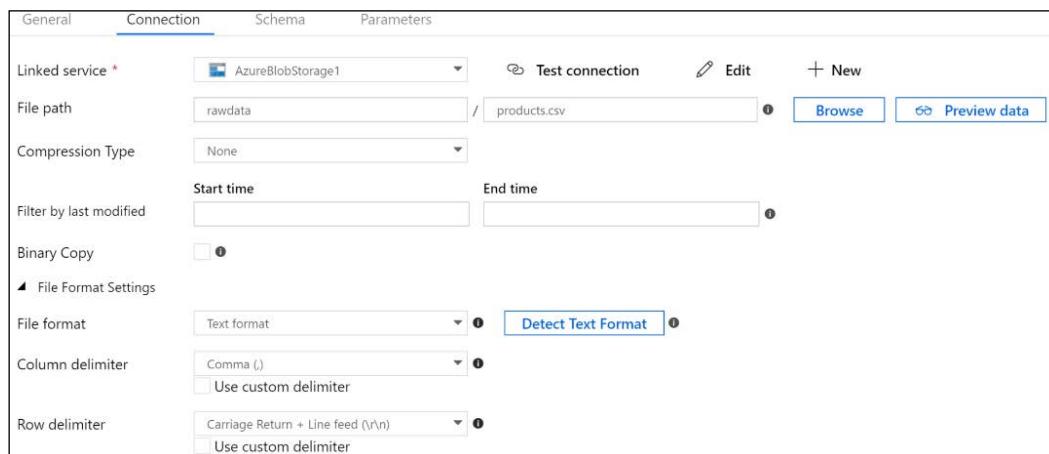
Annotations
+ New

► Advanced i

4. Linked services provide multiple authentication methods, and we are using the **shared access signature (SAS) uniform resource locator (URI)** method. It is also possible to use an account key, service principal, and managed identity as authentication methods:



5. The **Connection** tab after configuration should look similar to the following screenshot. Notice that the path includes the name of the container and the name of the file:



6. At this point, if you click on the **Preview data** button, it shows preview data from the products.csv file. On the Schema tab, add two columns, and name them `ProductID` and `ProductPrice`. The schema helps in providing an identifier to the columns and also mapping the source columns in the source dataset to the target columns in the target dataset when the names are not the same:

The screenshot shows the Azure Data Factory dataset configuration interface. On the left, there's a navigation bar with 'Save' and a blob storage icon. Below it, the dataset is named 'InputBlobStorageData'. The 'Schema' tab is selected. Under 'Import Schema', there are two columns defined: 'ProductID' (String type) and 'ProductPrice' (Decimal type). The 'Preview data' button is highlighted. To the right, the 'Data Preview' section shows a table of data with 10 rows. The columns are 'ProductID' and 'ProductPrice'. The data values are as follows:

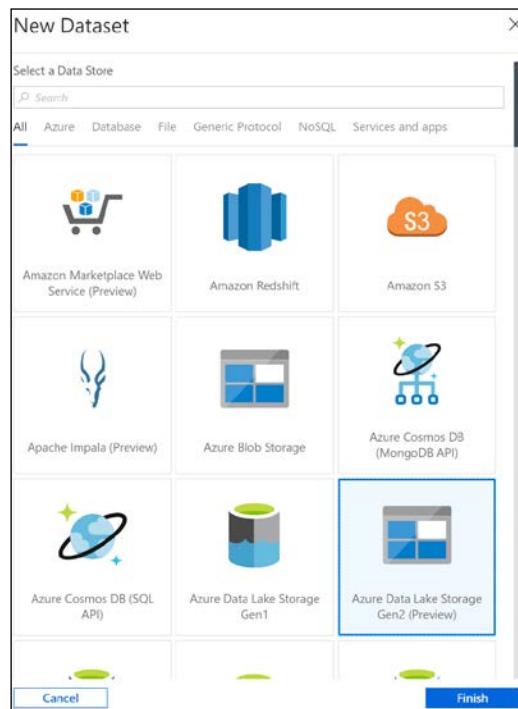
ProductID	ProductPrice
90269	9.989999771118164
101285	9.989999771118164
102676	9.989999771118164
84745	9.989999771118164
119920	9.989999771118164
123586	9.989999771118164
156029	15.989999771118164
130727	13.989999771118164
130727	9.989999771118164

Creating the second dataset

Create a new dataset and linked service for the destination Blob Storage account in the same way that you did before. Note that the Storage account is the same as the source but the container is different. Ensure that the incoming data have schema information associated with them as well, as shown in the following screenshot:

Creating a third dataset

Create a new dataset for the Data Lake Gen2 storage instance as the target dataset. To do this, select the new dataset and then select **Azure Data Lake Storage Gen2 (Preview)**, as demonstrated in the following screenshot:



Give the new dataset a name and create a new linked service in the **Connection** tab. Choose **Use account key** as the authentication method and the rest of the configuration will be auto-filled after selecting the Storage account name. Then, test the connection by clicking on the **Test connection** button. Keep the default configuration for the rest of the tabs, as shown in the following screenshot:

New Linked Service (Azure Data Lake Storage Gen2 ... X)

Name *
AzureDataLakeStorage1

Description

Connect via integration runtime *
AutoResolveIntegrationRuntime

Authentication method
Use account key

Account selection method
 From Azure subscription Enter manually

Azure subscription
Select all

Storage account name *
datalakegen2productstore

Sign up to the public preview of Azure Data Lake Storage Gen2.

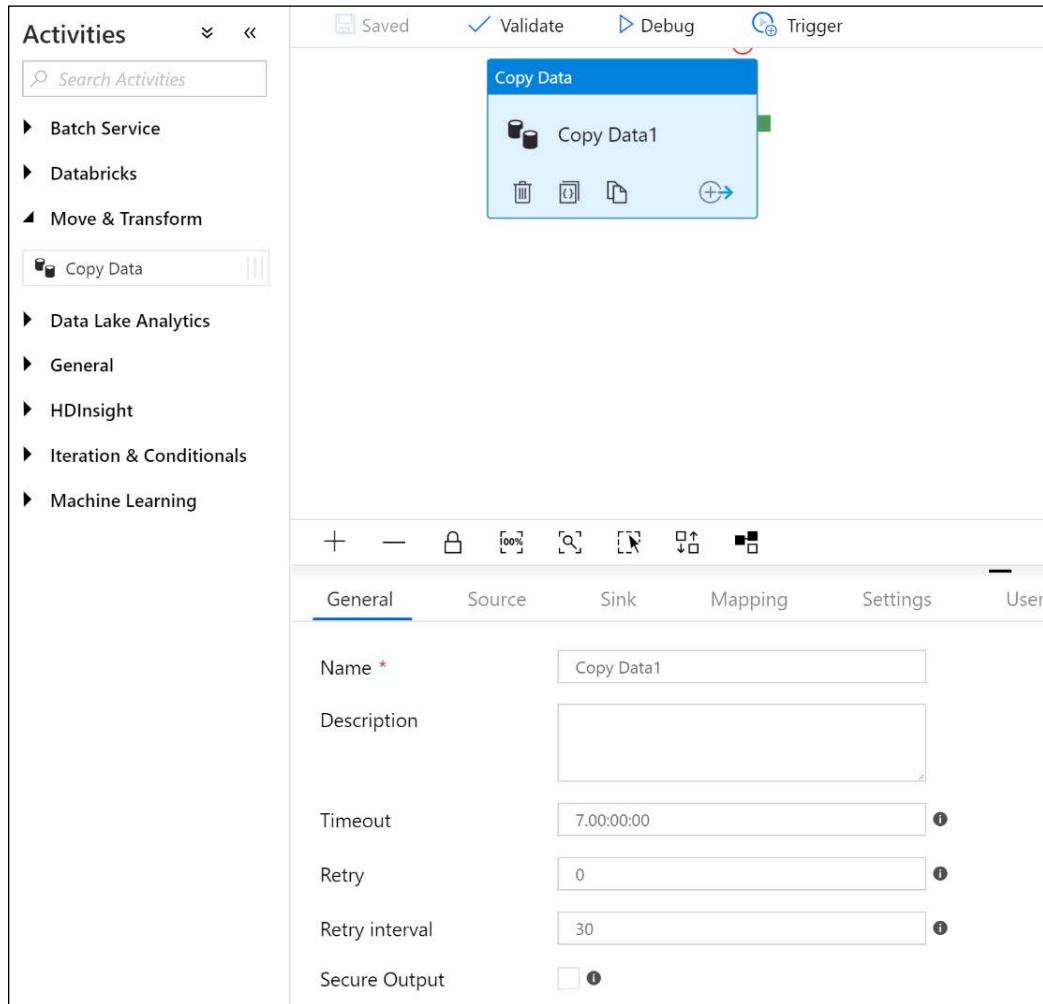
Annotations
+ New
► Advanced ⓘ

Cancel Test connection Finish

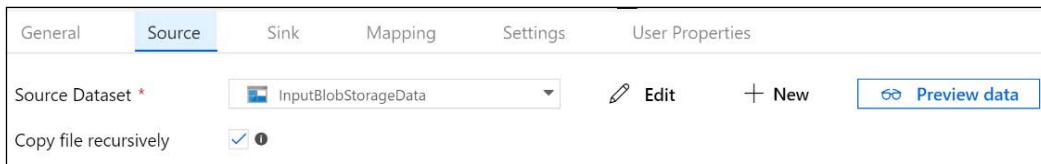
Creating a pipeline

After all the datasets are created, we can create a pipeline that will consume those datasets. The steps for creating a pipeline are mentioned next:

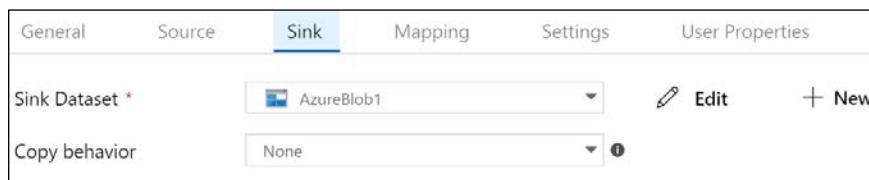
1. Click on the **+ Pipeline** menu from the top menu to create a new pipeline. Then, drag and drop the **Copy Data** activity from the **Move & Transform** menu, as demonstrated in the following screenshot:



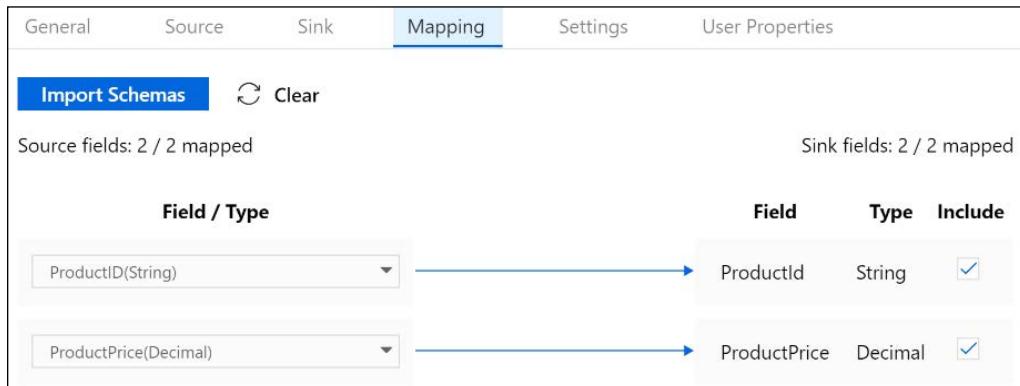
2. The resultant **General** tab can be left as it is, but the **Source** tab should be configured to use the source dataset that we configured earlier:



3. The **Sink** tab is used for configuring the destination data store and dataset, and it should be configured to use the target dataset that we configured earlier:



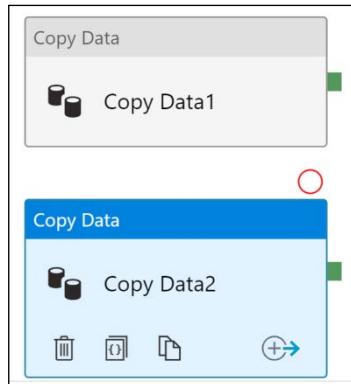
4. In the **Mapping** tab, map the columns from the source to the destination dataset columns, as shown in the following screenshot:



Add one more copy data activity

Within Pipeline, we can add multiple activities, each responsible for a particular transformation task. The task mentioned in this section is responsible for copying data from Azure Storage Account to Azure Data Lake Storage:

1. Add another **Copy Data** activity from the left activity menu to migrate data to Data Lake Storage; both of the copy tasks will execute in parallel:

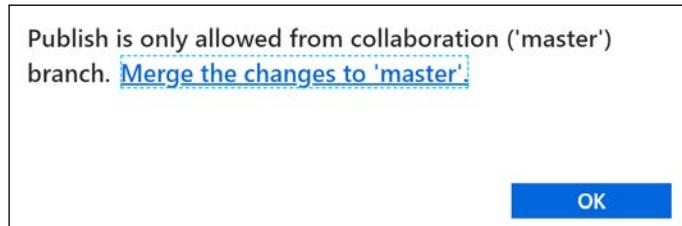


2. The configuration for the source is the Azure Blob Storage account that contains the `products.csv` file.
3. The sink configuration will target the Data Lake Gen2 storage account.
4. The rest of the configuration can be left in their default settings for the second copy data activity.

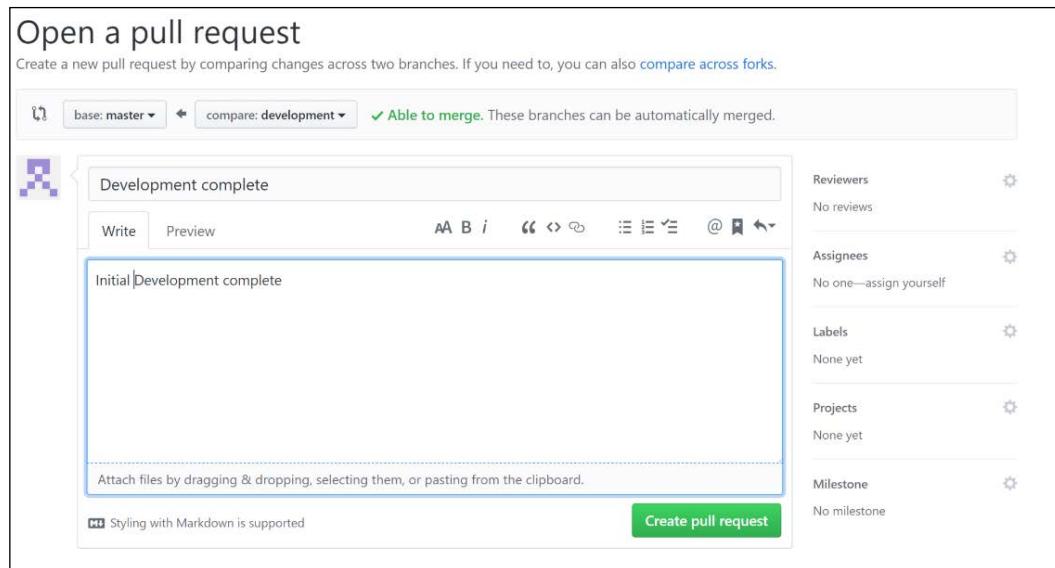
Publishing

After creating and configuring pipeline, it is important that they are published such that it can be consumed and scheduled for production use. The steps to publish a pipeline are mentioned here:

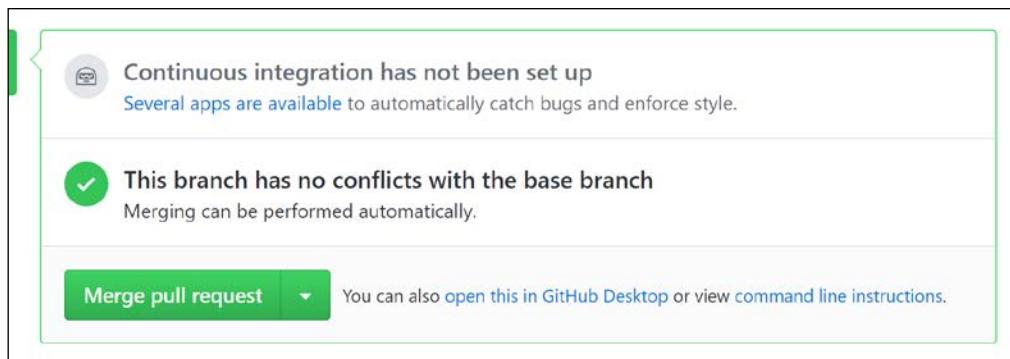
1. We can run the already configured pipeline in debug mode. After debugging, we can publish the pipeline. However, before we can publish the pipeline, we need to merge the code from the development branch to the master branch and, optionally, delete the development branch:



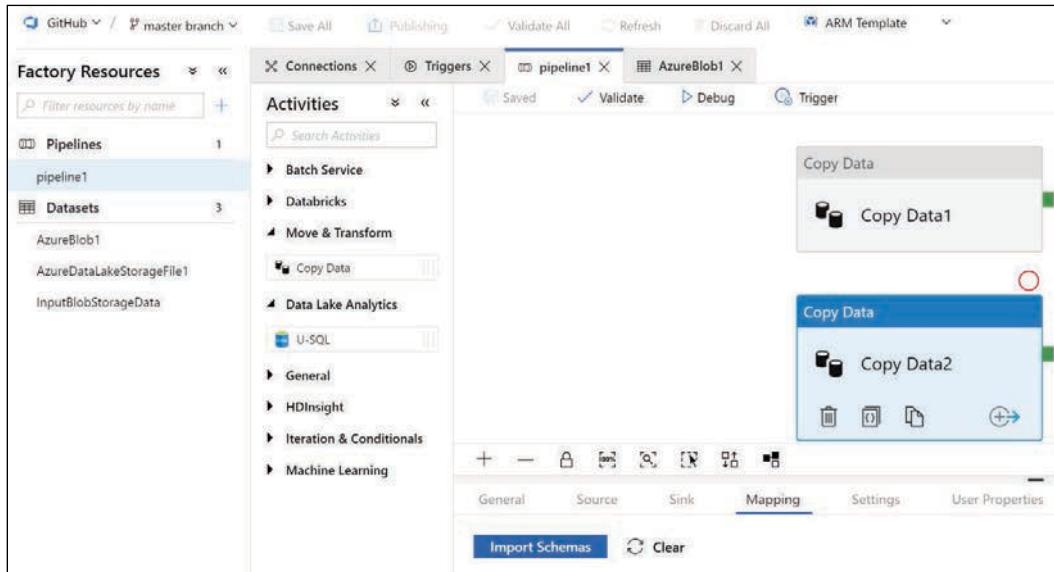
2. Go to the GitHub account and raise a new pull request for merging the code from the development branch to the master branch. This is because the entire code for the pipelines, datasets, linked services, and **Azure Resource Manager (ARM)** templates is in the development branch:



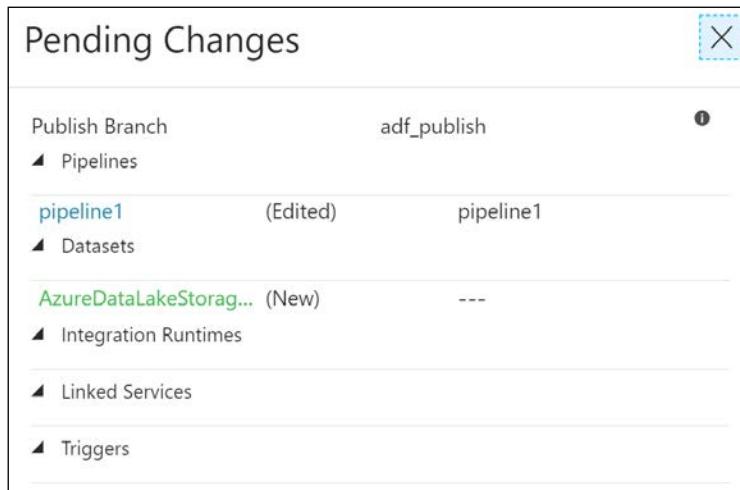
3. After the pull request is raised, accept the pull request and merge the code from the development branch to the master branch:



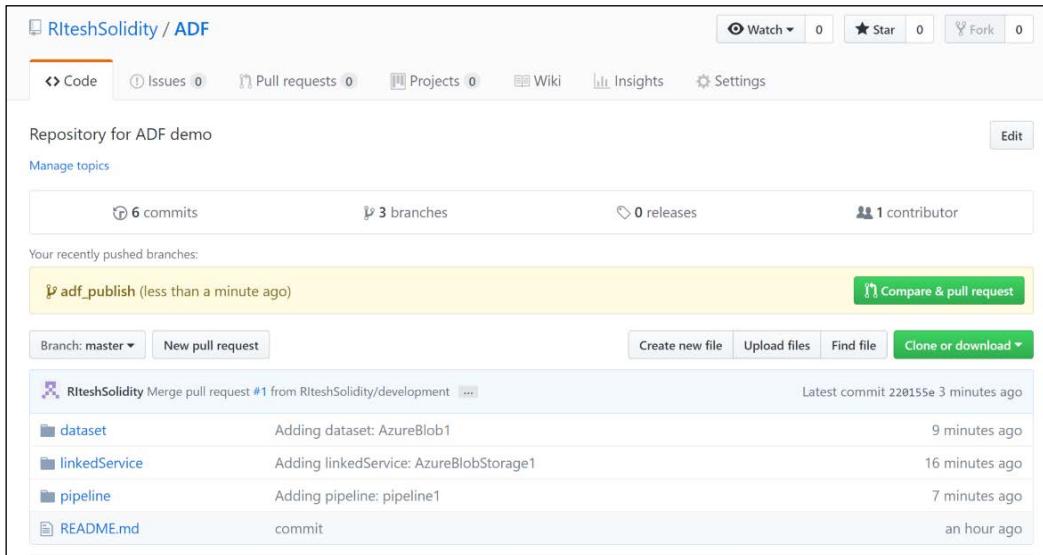
4. Back in the Data Factory designer, select **master branch** from the top menu and publish the code:



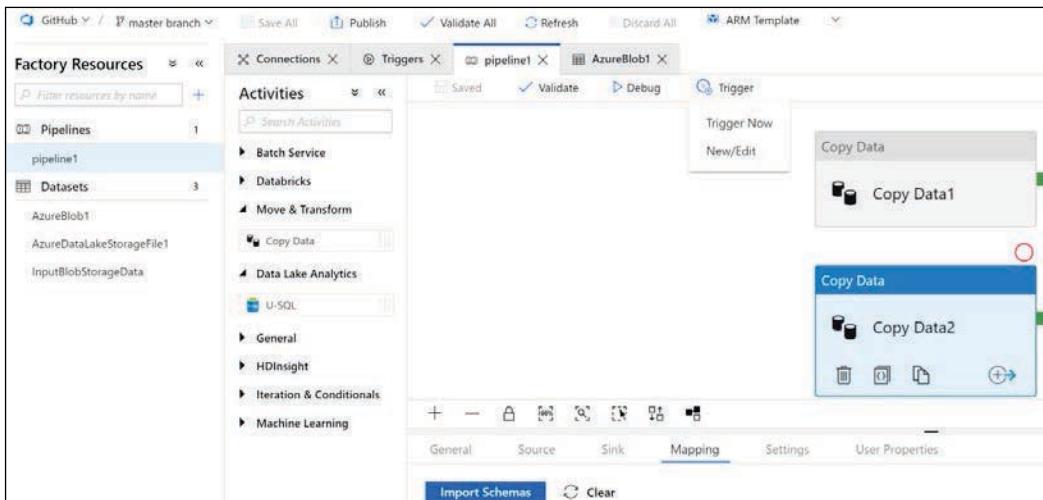
5. Click on the resultant blade, as demonstrated in the following screenshot:



6. The GitHub repository should look like the following screenshot:



7. Trigger the pipeline by clicking on the **Trigger Now** button from the top menu once the publishing is complete:



Triggering a data factory will push the code to the repository, generate the ARM templates, and execute the pipeline.

The final result

If we navigate to the Storage account containing both the source and destination containers, we should be able to see our new `processedproducts.csv` file, as follows:

NAME	MODIFIED	ACCESS TIER	BLOB TYPE	SIZE	LEASE STATE
processedproducts.csv	1/27/2019, 5:47:22 AM	Hot (Inferred)	Block blob	9.12 MB	Available

We cannot see the files for Data Lake Storage; therefore, we have to download the Storage Explorer tool to view the files in the Data Lake Storage account:

Name	Last Modified	Content Type	Size
information.csv	1/27/2019, 2:20:37 AM		9.1 MB

The Storage Explorer tool should be no earlier than version 1.6.2 and can be downloaded from <https://aka.ms/portalfx/downloadstorageexplorer>.

Summary

This was another chapter on handling big data. This chapter dealt with the Azure Data Factory service, which is responsible for providing ETL services on Azure. Since it is a **Platform as a Service (PaaS)**, it provides unlimited scalability, high availability, and easy-to-configure pipelines. Its integration with Azure DevOps and GitHub is also seamless. We also explored the features and benefits of using Azure Data Lake Gen2 storage for storing any kind of big data. It is a cost-effective, highly scalable, hierarchical data store for handling big data, and is compatible with Azure HDInsight, Databricks, and the Hadoop ecosystem.

The next chapter will focus on creating solutions that are based on events. Azure provides us with resources that help in generating, as well as consuming, events. The next chapter will go deep into resources including Event Hubs and Stream Analytics.

14

Azure Stream Analytics and Event Hubs

Events are everywhere! Any activity or task that changes the current state generates an event. Events were not that popular a few years ago; there was no cloud, and not much traction for the **Internet of Things (IoT)** and devices or sensors. During this time, organizations used hosted environments from **internet service providers (ISPs)** that just had monitoring systems on top of them. These monitoring systems raised events that were few and far between.

However, with the advent of the cloud, things are changing rapidly. With increased deployments on the cloud, especially of **Platform as a Service (PaaS)** services, organizations no longer need much control over the hardware and the platform, and now every time there is a change in an environment, an event is raised. With the emergence of cloud events, IoT has gained a lot of prominence and events have started taking center stage.

Another phenomenon during this time has been the rapid burst of growth in the availability of data. The velocity, variety, and volume of data has spiked, and so has the need for solutions for storing and processing data. Multiple solutions and platforms have emerged, such as Hadoop, data lakes for storage, data lakes for analytics, and machine learning services for analytics.

Apart from storage and analytics, there is also a need for services that are capable of ingesting millions upon millions of events and messages from various sources. There is also a need for services that can work on temporal data, rather than working on the entire snapshot of data.

In this chapter, we will go through a couple of pertinent services in Azure, as follows:

- Azure Event Hubs
- Azure Stream Analytics

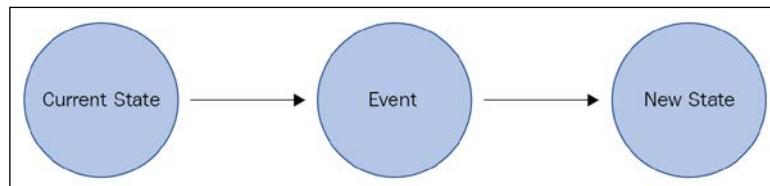
Introducing events

Events are important constructs in both Azure and Azure Application architecture. Events are everywhere within the software ecosystem. Generally, any action taken results in an event that can be trapped and further action can be taken. To take this discussion forward, it is important to first understand the basics of events.

Events

Events help to change the current state to the target state. Events are different to messages. Messages are related to business functionality, such as sending order details to another system. In comparison, events are different; for instance, a virtual machine being stopped is an event.

An event, along with the current state, helps in transitioning from the current state to the target state, and this transition is demonstrated in the following diagram:



Event streaming

If there are only a few events, then there is not much of a need for event streaming, as event streaming provides real value with big data processes, where data comes in high volumes and in many different formats.

Event streaming refers to services that can accept data as and when it arises, rather than accepting it periodically. For example, event streams should be capable of accepting temperature information from devices as and when they send it, rather than making the data wait in a queue or a staging environment.

Event streaming also has the capability of querying data while in transit. This is temporal data that is stored for a while, and the queries occur on the moving data; therefore, the data is not stationary. This capability is not available in other data platforms, which can only query stored data and not temporal data that has just been ingested.

Event streaming services should be able to scale easily to accept millions or even billions of events. They should be highly available such that sources can send events and data to them at any time. Real-time data ingestion and being able to work on that data, rather than data stored in a different location, is the key to event streaming.

But when we already have so many data platforms with advanced query execution capabilities, why do we need event streaming? Let me provide you with some scenarios in which working on incoming data is quite important. These scenarios cannot be effectively and efficiently solved by existing data platforms:

- **Credit card fraud detection:** This should happen as and when a fraudulent transaction is happening.
- **Telemetry information from sensors:** In the case of IoT devices sending vital information about their environments, the user should be notified as and when an anomaly is detected.
- **Live dashboards:** Event streaming is needed to create dashboards that show live information.
- **Data center environment telemetry:** This will let the user know about any intrusions, security breaches, failures of components, and more.

There are many possibilities for applying event streaming within an enterprise, and its importance cannot be stressed enough.

Event Hubs

Event Hubs is the service in Azure that provides capability related to the ingestion and storage of events that are needed for streaming.

It can ingest data from a variety of sources; these sources could be IoT sensors or any applications using the Event Hubs **Software Development Kit (SDK)** to send data. It supports multiple protocols for ingesting and receiving data. These protocols are industry standard, and they include the following:

- **HTTP:** This is a stateless option and does not require an active session.
- **Advanced Messaging Queuing Protocol (AMQP):** This requires an active session (that is, an established connection using sockets) and works with **Transport Layer Security (TLS)** and **Secure Socket Layer (SSL)**.
- **Apache Kafka:** This is a distributed streaming platform similar to Stream Analytics.

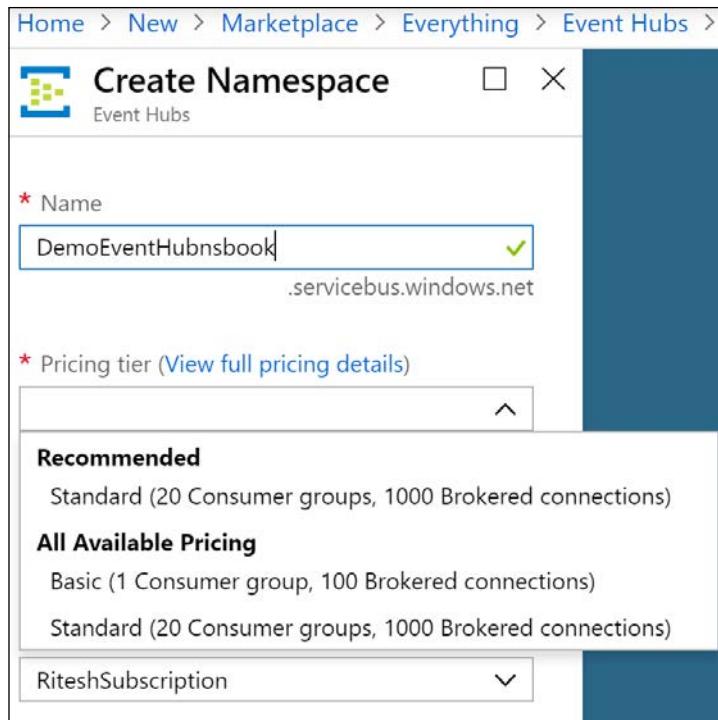
Event Hubs is an event ingestion service. It does not have the capability to query and output query results to another location. That is the responsibility of Stream Analytics, which is covered in the next section.

Event Hubs, being a PaaS on Azure, is highly distributed, highly available, and highly scalable.

Event Hubs comes with the following two SKUs or pricing tiers:

- **Basic:** This comes with one consumer group and can retain messages for 1 day. It can have a maximum of 100 brokered connections.
- **Standard:** This comes with a maximum of 20 consumer groups and can retain messages for 1 day with additional storage for 7 days. It can have a maximum of 1,000 brokered connections. It is also possible to define policies in this SKU.

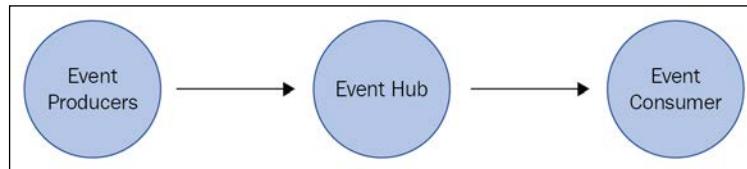
The following screenshot shows the form for creating a new Event Hubs namespace. It provides an option to choose an appropriate pricing tier, along with other important details:



It is important to note that the SKU cannot be changed after provisioning an Event Hubs namespace. Due consideration and planning should be taken before selecting an SKU. The planning process should include planning the number of consumer groups required and the number of applications interested in reading events from the event hub.

The architecture of Event Hubs

There are three main components of the Event Hubs architecture: the **Event Producers**, the **Event Hub**, and the **Event Consumer**, as shown in the following diagram:



Event Producers generate events and send them to the Event Hub. The **Event Hub** stores the ingested events and provides that data to the **Event Consumer**. The **Event Consumer** is whatever is interested in those events, and they connect to the **Event Hub** to fetch the data.

Event hubs cannot be created without an Event Hubs namespace. The Event Hubs namespace acts as a container and can host multiple event hubs. Each Event Hubs namespace provides a unique REST-based endpoint that is consumed by clients to send data to Event Hubs. This namespace is the same namespace that is needed for Service Bus artifacts, such as topics and queues.

The connection string of an Event Hubs namespace is composed of its URL, policy name, and the key. A sample connection string is shown in the following code block:

```
Endpoint=sb://demoeventhubsbook.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=M/4eeBsr7DALXcvw6ziFqlSDNbFX6E49Jfti8CRkbA=
```

This connection string can be found in the **Shared Access Signature (SAS)** menu item of the namespace. There can be multiple policies defined for a namespace, with each having different levels of access to the namespace. The three levels of access are as follows:

- **Manage**: This can manage the event hub from an administrative perspective. It also has rights for sending and listening to events.
- **Send**: This can write events to Event Hubs.
- **Listen**: This can read events from Event Hubs.

By default, the **RootManageSharedAccessKey** policy is created when creating an event hub, as shown in the following screenshot. Policies help in creating granular access control on Event Hubs. The key associated with each policy is used by consumers to determine their identity; additional policies can also be created:

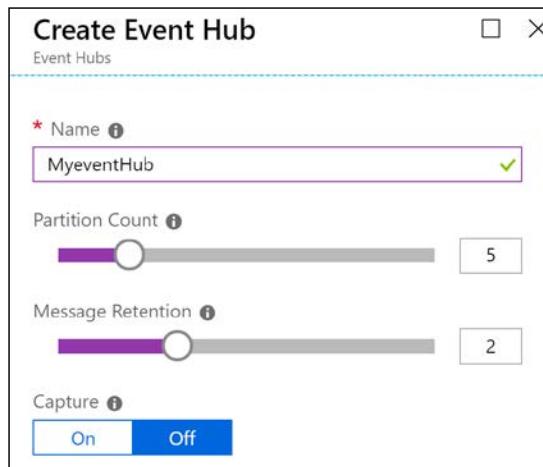
The screenshot shows the Azure portal interface for managing a Shared access policy. On the left, the navigation pane includes links for Overview, Access control (IAM), Tags, Diagnose and solve problems, Events, Settings, Shared access policies (which is the current page), Geo-Recovery, Firewalls and virtual networks, and Properties. The main content area displays the 'DemoEventHubsbook - Shared access policies' screen. It lists a single policy: 'RootManageSharedAccessKey' with 'Manage, Send, Listen' claims. To the right, a detailed view of the 'RootManageSharedAccessKey' policy shows the following fields:

- Manage**: Checked checkbox.
- Send**: Checked checkbox.
- Listen**: Checked checkbox.
- Primary key**: Text input field containing the value 'Mr/4eeBsr7DALXcvw6ziFqlSDNbFX6E49Jfti8CRkbA='.
- Secondary key**: Text input field containing the value 'tSar0mrREoyK7D/92R27d3l5bu4QuX00sRzzVqxG+GQ='.
- Connection string-primary key**: Text input field containing the value 'Endpoint=sb://demoeventhubsbook.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=M/4eeBsr7DALXcvw6ziFqlSDNbFX6E49Jfti8CRkbA='.
- Connection string-secondary key**: Text input field containing the value 'Endpoint=sb://demoeventhubsbook.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=tSar0mrREoyK7D/92R27d3l5bu4QuX00sRzzVqxG+GQ='.

Event hubs can be created from the Event Hubs namespace service by clicking on **Event Hubs** in the left-hand menu and clicking on **+ Event Hub** in the resultant screen:

The screenshot shows the Azure portal interface for the 'DemoEventHubsbook - Event Hubs' namespace. The top navigation bar includes 'Home > DemoEventHubsbook - Overview > DemoEventHubsbook - Event Hubs'. The main title is 'DemoEventHubsbook - Event Hubs' with the subtitle 'Event Hubs Namespace'. On the left, a sidebar lists various management options: Overview, Access control (IAM), Tags, Diagnose and solve problems, Events, Settings (Shared access policies, Geo-Recovery, Firewalls and virtual networks, Properties, Locks, Automation script), and Entities (Event Hubs). The 'Event Hubs' item in the Entities section is highlighted with a red box. The main content area features a search bar ('Search (Ctrl+ /)'), a '+ Event Hub' button (also highlighted with a red box), and a 'Refresh' button. Below these are sections for 'NAME' and 'no Event Hubs yet.'

Next, provide information about the **Partition Count** and **Message Retention** values, along with the name with which to create the event hub. Then, select Off for Capture, as demonstrated in the following screenshot:



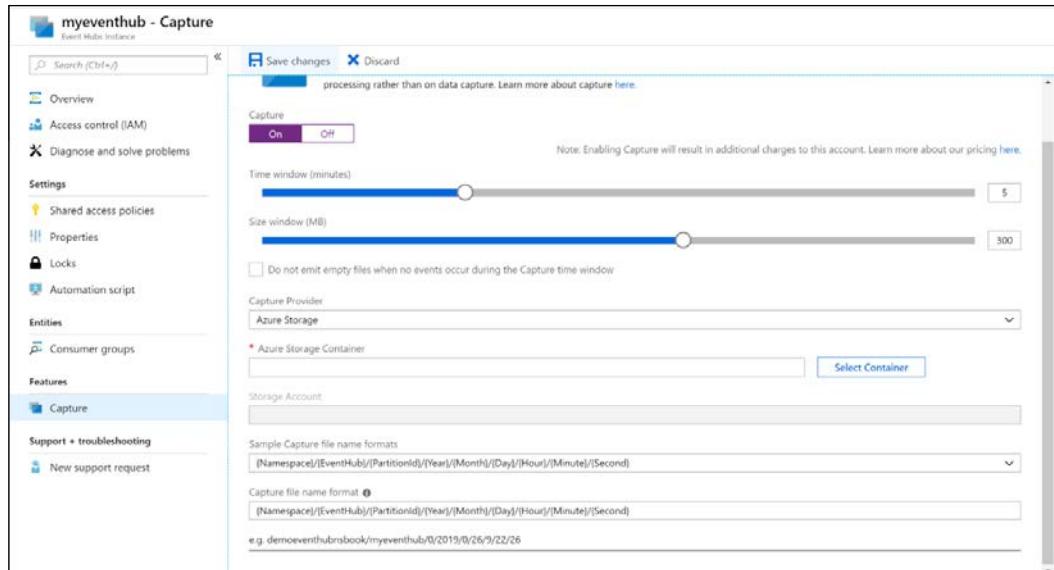
Separate policies can be assigned to each event hub by adding a new policy at the event hub level.

After creating the policy, the connection string is available from the **Secure Access Signature** left-menu item on the Azure portal.

Since a namespace can consist of multiple event hubs, the connection string for an individual event hub will be similar to the following code block. The difference here is in the key value and the addition of EntityPath with the name of the event hub:

```
Endpoint=sb://azurertwittereventdata.servicebus.windows.net/;SharedAccessKey=EventhubPolicy;SharedAccessKey=rxEu5K4Y2qsi5wEeOKuOvRnhtgW8xW35UBex4VlIKqg=;EntityPath=myeventhub
```

We had to keep the **Capture** option set to **Off** while creating the event hub, and it can be switched back on after creating the event hub. It helps to save events to Azure Blob Storage or an Azure Data Lake Storage account automatically. The configuration for the size and time interval is shown in the following screenshot:

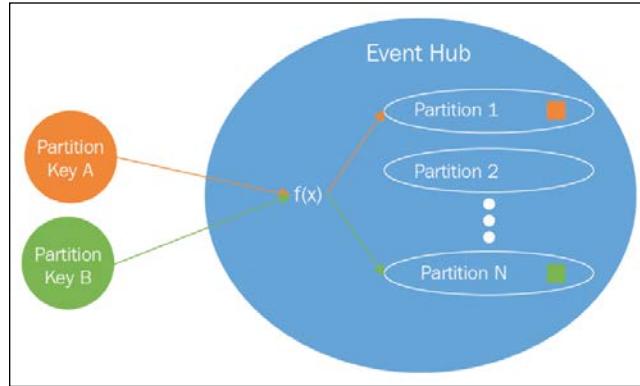


We did not cover the concepts of partitions and message retention options when creating this event hub.

Partition is an important concept related to the scalability of any data store. Events are retained within event hubs for a specific period of time. If all the events are stored within the same data store, then it becomes extremely difficult to scale that data store. Every event producer will connect to the same data store and send their events to it. Compare this with a data store that can partition the same data into multiple smaller data stores, each uniquely identified with a value. The smaller data store is called a **partition**, and the value that defines the partition is known as the **partition key**. This partition key is part of the event data.

Now the event producers can connect to the event hub, and based on the value of the partition key, the event hub will store the data in an appropriate partition. This will allow the event hub to ingest multiple events at the same time in parallel.

Deciding on the number of partitions is a crucial aspect for the scalability of an event hub. The following diagram shows that ingested data is stored in the appropriate partition internally by Event Hubs using the partition key:



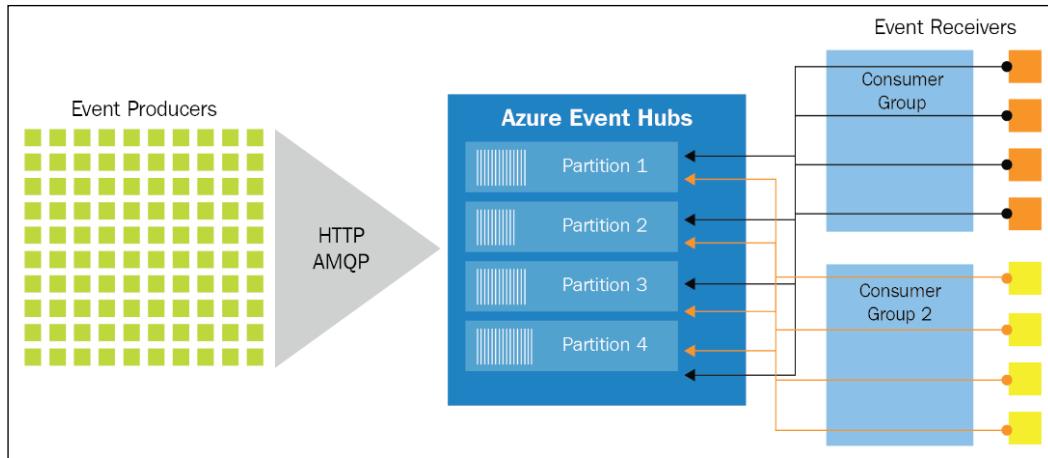
It is important to understand that the same partition might have multiple keys. The user decides how many partitions are required, and the event hub internally decides the best way to allocate the partition keys between them. Each partition stores data in an orderly way using a timestamp, and newer events are appended toward the end of the partition.

It is important to note that it is not possible to change the number of partitions once the event hub is created.

It is also important to remember that partitions also help in bringing parallelism and concurrency for applications reading the events. For example, if there are 10 partitions, 10 parallel readers can read the events without any degradation in performance.

Consumer groups

Consumers are applications that read events from an event hub. Consumer groups are created for consumers to connect to it in order to read the events. There can be multiple consumer groups for an event hub, and each consumer group has access to all the partitions within an event hub. Each consumer group forms a query on the events in events hubs. Applications can use consumer groups and each application will get a different view of the event hub events. A default `$default` consumer group is created when creating an event hub. It is a good practice for one consumer to be associated with one consumer group for optimal performance. However, it is possible to have five readers on each partition in a consumer group:



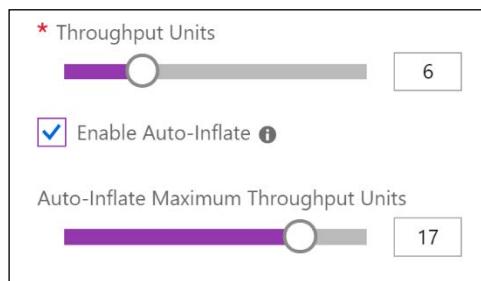
Throughput

Partitions help in scalability, while throughput helps in capacity per second. So, what is capacity in terms of Event Hubs? It is the amount of data that can be handled per second.

In Event Hubs, a single throughput unit allows the following:

- 1 MB of ingestion data per second or 1,000 events per second (whichever happens first)
- 2 MB of egress data per second or 4,096 events per second (whichever happens first)

Auto-inflate helps in increasing the throughput automatically if the number of incoming/outgoing events or the incoming/outgoing total size crosses a threshold. Instead of throttling, the throughput will scale up and down. The configuration of throughput at the time of the creation of the namespace is shown in the following screenshot. Again, careful thought should go into deciding the throughput units:

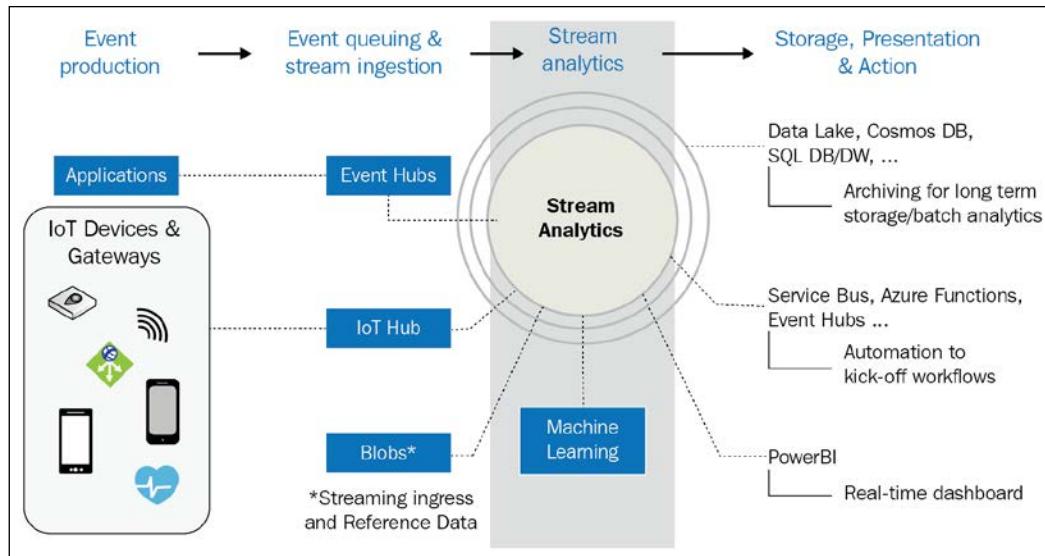


A primer on Stream Analytics

Event Hubs is only an event ingestion platform, so we need another service that can process these events as a stream rather than just as stored data. Stream Analytics helps in processing and examining a stream of big data, and Stream Analytics jobs help to execute the processing of events.

Stream Analytics can process millions of events per second and it is quite easy to get started with it. Azure Stream Analytics is a PaaS that is completely managed by Azure. Customers of Stream Analytics do not have to manage the underlying hardware and platform.

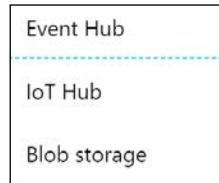
Each job comprises multiple inputs, outputs, and a query, which transforms the incoming data into new output. The whole architecture of Stream Analytics is shown in the following diagram:



In the preceding diagram, the event sources are displayed on the extreme left. These are the sources that produce the events. They could be IoT devices, custom applications written in any programming language, or events coming from other Azure platforms, such as Log Analytics or Application Insights.

These events must first be ingested into the system, and there are numerous Azure services that can help to ingest this data. We've already looked at Event Hubs and how they help in ingesting data. There are other services, such as IoT Hub, that also help in ingesting device-specific and sensor-specific data. IoT Hub and ingestion is covered in detail in *Chapter 15, Designing IoT Solutions*. This ingested data undergoes processing as it arrives in a stream, and this processing is done using Stream Analytics. The output from Stream Analytics could be a presentation platform such as Power BI, showing real-time data to stakeholders, or a storage platform such as Cosmos DB, Data Lake Storage, or Azure Storage, from which the data can be read and actioned later by Azure Functions and Service Bus queues.

Stream Analytics is capable of ingesting millions of events per second and has the capability to execute queries on top of them. At the time of writing, Stream Analytics supports the three sources of events listed in the following screenshot:



Input data is supported in any of the three following formats:

- **JavaScript Object Notation (JSON)**: This is a lightweight, plaintext-based format that is human readable. It consists of name-value pairs; an example of a JSON event is as follows:

```
{  
    "SensorId" : 2,  
    "humidity" : 60,  
    "temperature" : 26C  
}
```

- **Comma-Separated Values (CSV)**: These are also plaintext values, which are separated by commas. Examples of CSV is shown in next image. The first row is the header containing three fields followed by two rows of data.

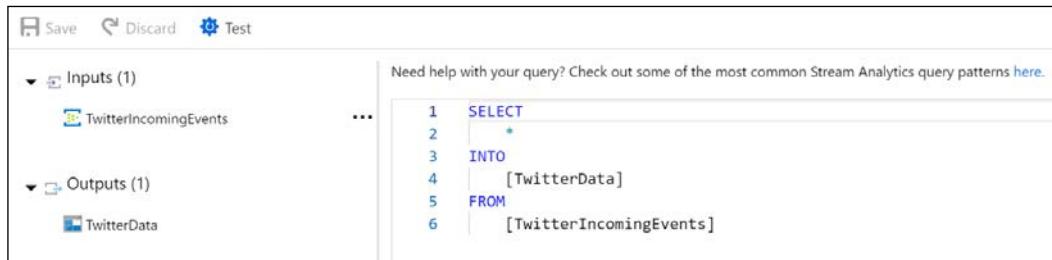
SensorId, humidity, temperature
2,60,26C
3,65,31C

- **Avro**: This format is similar to JSON; however, it is stored in a binary format rather than a text format.

Azure Stream Analytics and Event Hubs

Not only can Stream Analytics receive events, but it also provides advanced query capability for the data that it receives. The queries can extract important insights from the temporal data streams and output them.

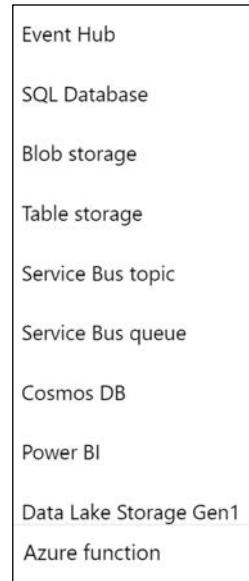
As shown in the following screenshot, there is an input and an output; the query moves the events from the input to the output. The `INTO` clause refers to the output location, and the `FROM` clause refers to the input location. The queries are very similar to SQL queries, so the learning curve is not too steep for SQL programmers:



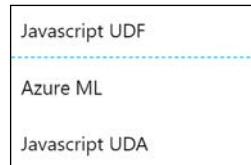
The screenshot shows the Azure Stream Analytics query editor interface. At the top, there are buttons for Save, Discard, and Test. Below that, the query editor area has sections for Inputs and Outputs. Under Inputs, there is one item: TwitterIncomingEvents. Under Outputs, there is one item: TwitterData. To the right of the inputs, a query editor window displays the following T-SQL code:

```
1  SELECT
2  *
3  INTO
4  [TwitterData]
5  FROM
6  [TwitterIncomingEvents]
```

Event Hubs also provides mechanisms for sending outputs from queries to target destinations. At the time of writing, Stream Analytics supports multiple destinations for events and query outputs. These destinations are shown in the following screenshot:

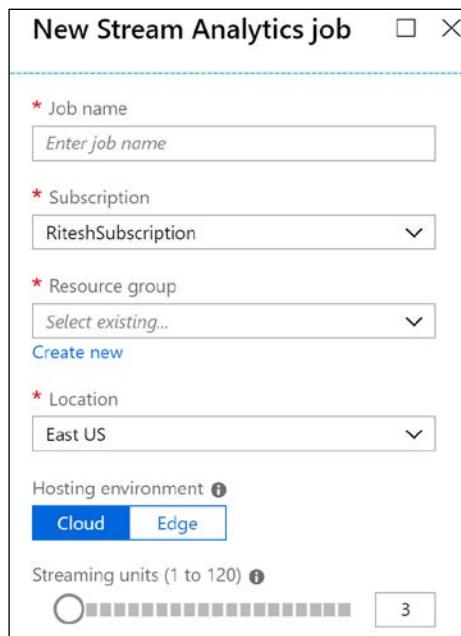


It is also possible to define custom functions that can be reused within queries. There are three options provided to define custom functions, as follows:



The hosting environment

Stream Analytics jobs can run on hosts that are running on the cloud or they can run on IoT edge devices. IoT edge devices are devices that are near to IoT sensors, rather than on the cloud. The following screenshot shows the **New Stream Analytics job** blade:



Streaming Units

From the previous screenshot, we can see that the only configuration that is unique to Stream Analytics is **Streaming Units**. Streaming units refer to the resources (that is, CPU and memory) that are assigned for running a Stream Analytics job. The minimum and maximum streaming units are 1 and 120, respectively.

Depending on the amount of data and the number of queries executed on that data, streaming units must be preallocated; otherwise, the job will fail.

It is possible to scale the streaming units up and down from the Azure portal.

A sample application using Event Hubs and Stream Analytics

In this section, we will be creating a sample application comprising multiple Azure services, including Azure Logic Apps, Azure Event Hubs, Azure Storage, and Azure Stream Analytics.

In this sample application, we will be reading all tweets containing the word "Azure" and storing them in an Azure Storage account.

To create this solution, we first need to provision all the necessary resources.

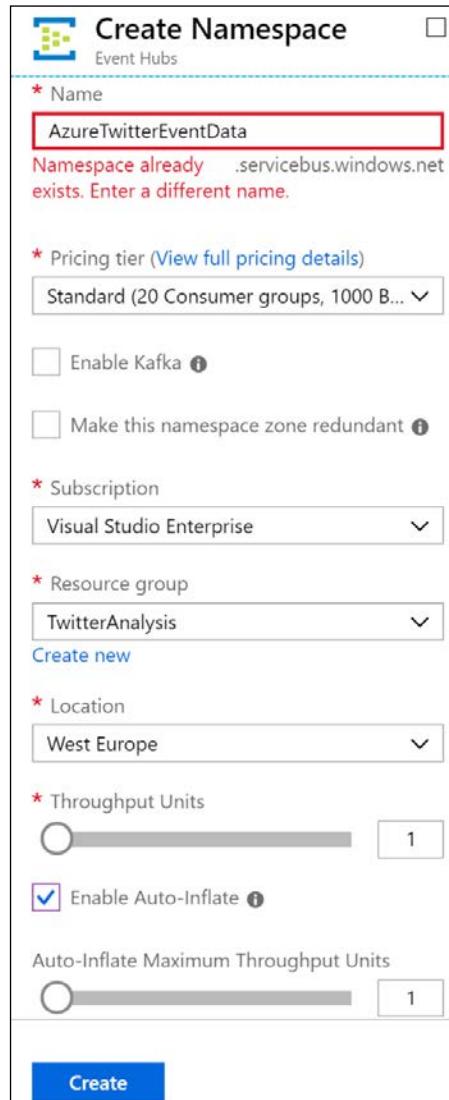
Provisioning a new resource group

Navigate to the Azure portal, log in using valid credentials, click on **+ Create a resource**, and search for **Resource group**. Select **Resource group** from the search results and create a new resource group. Then, provide a name and choose an appropriate location. Note that all resources should be hosted in the same resource group and location so that it is easy to delete them:

The screenshot shows the 'Create a resource group' wizard in the Azure portal. The URL in the address bar is: Home > New > Marketplace > Everything > Resource group > Create a resource group. The title is 'Create a resource group'. Below it, there are three tabs: 'Basics' (which is selected), 'Tags', and 'Review + Create'. The 'Basics' tab has a description: 'Resource group - A container that holds related resources for an Azure solution. The resource group can include all the resources for the solution, or only those resources that you want to manage as a group. You decide how you want to allocate resources to resource groups based on what makes the most sense for your organization.' Below this is a 'PROJECT DETAILS' section. It includes a 'Subscription' dropdown set to 'RiteshSubscription' and a 'Resource group' dropdown set to 'TwitterAnalysis', which is highlighted with a purple border and a green checkmark. At the bottom is a 'RESOURCE DETAILS' section with a 'Region' dropdown set to 'West Europe'.

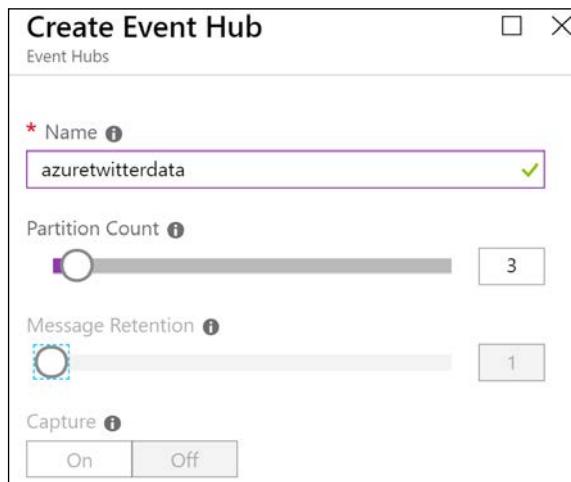
Creating an Event Hubs namespace

Click on **+ Create a resource** and search for **Event hubs**. Select **Event hubs** from the search results and create a new event hub. Then, provide a name and location, and select a subscription based on the resource group that was created earlier. Select **Standard** as the pricing tier and also select **Enable Auto-inflate**, as shown in the following screenshot:



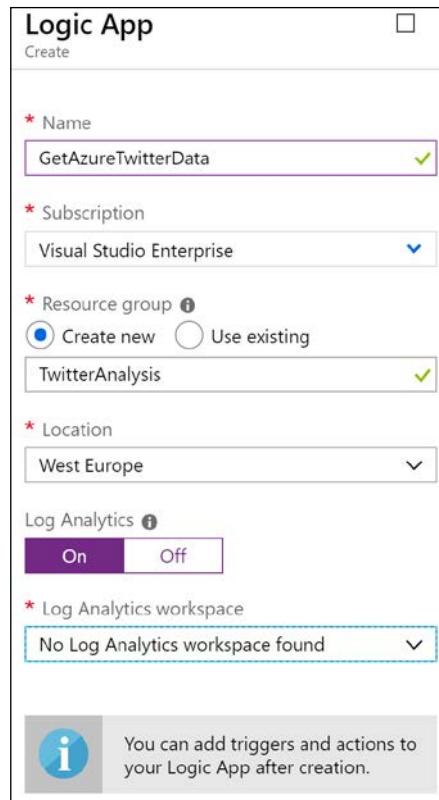
Creating an event hub

From the Event Hubs namespace service, click on **Events hubs** in the left-hand menu, and then click on **+ Event hubs** to create a new event hub. Name it `azuretwitterdata` and provide an optimal number of partitions and a **Message Retention** value:

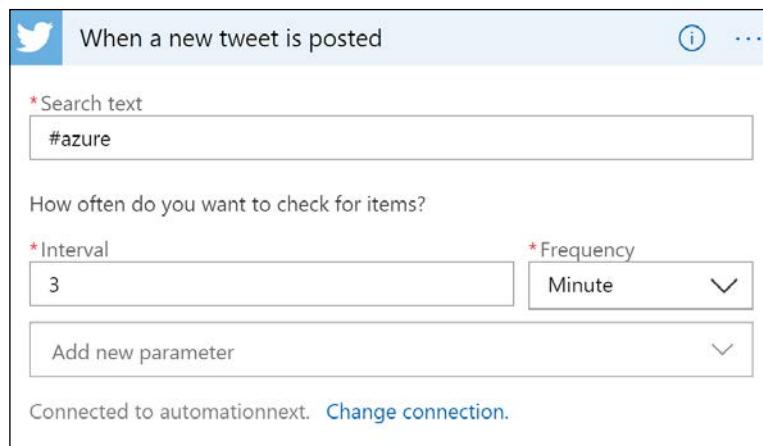


Provisioning a logic app

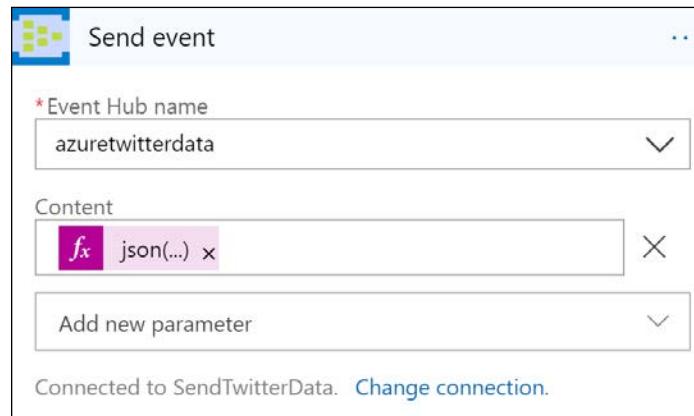
After the resource group is provisioned, click on **+ Create a resource** and search for **Logic Apps**. Select **Logic Apps** from the search results and create a new logic app. Then, provide a name and location, and select a subscription based on the resource group created earlier. It is a good practice to enable **Log Analytics**. Logic Apps is covered in more detail in *Chapter 7, Azure Integration Solutions*. The logic app is responsible for connecting to Twitter using an account and fetching all the tweets with Azure in them:



After the logic app is created, select the **When a new tweet is posted** trigger on the design surface, sign in, and then configure it as shown in the following screenshot. You will need a valid Twitter account before configuring this trigger:

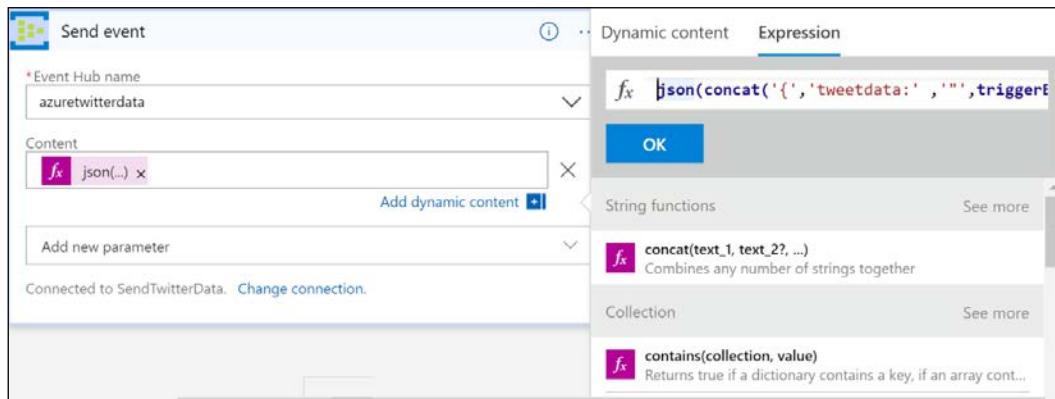


Next, drop a **Send event** action on the designer surface; this action is responsible for sending tweets to the event hub:



Select the event hub name that was created in an earlier step.

The value specified in the content textbox is an expression that has been dynamically composed using Logic Apps-provided functions and Twitter data. Clicking on **Add dynamic content** provides a dialog through which the expression can be composed:



The value of the expression is as follows:

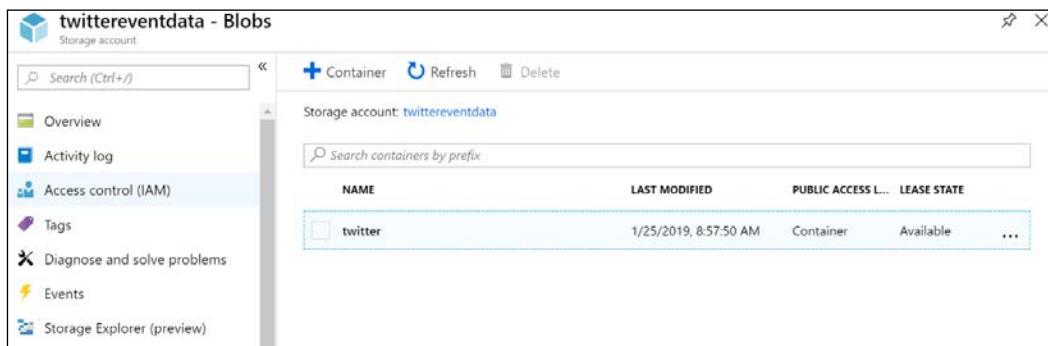
```
json(concat('{{', 'tweetdata:', ''', triggerBody()?['TweetText'], ''', '}}'))
```

Provisioning the storage account

Click on **+ Create a resource** and search for Storage Account. Select **Storage Account** from the search results and create a new storage account. Then, provide a name and location, and select a subscription based on the resource group that was created earlier. Finally, select **StorageV2** for **Account Kind**, **Standard for Performance**, and **Locally-redundant storage (LRS)** for the **Replication** field.

Creating a storage container

Stream Analytics will output the data as files that will be stored within a Blob Storage container. A container named **twitter** is created within Blob Storage, as shown in the following screenshot:

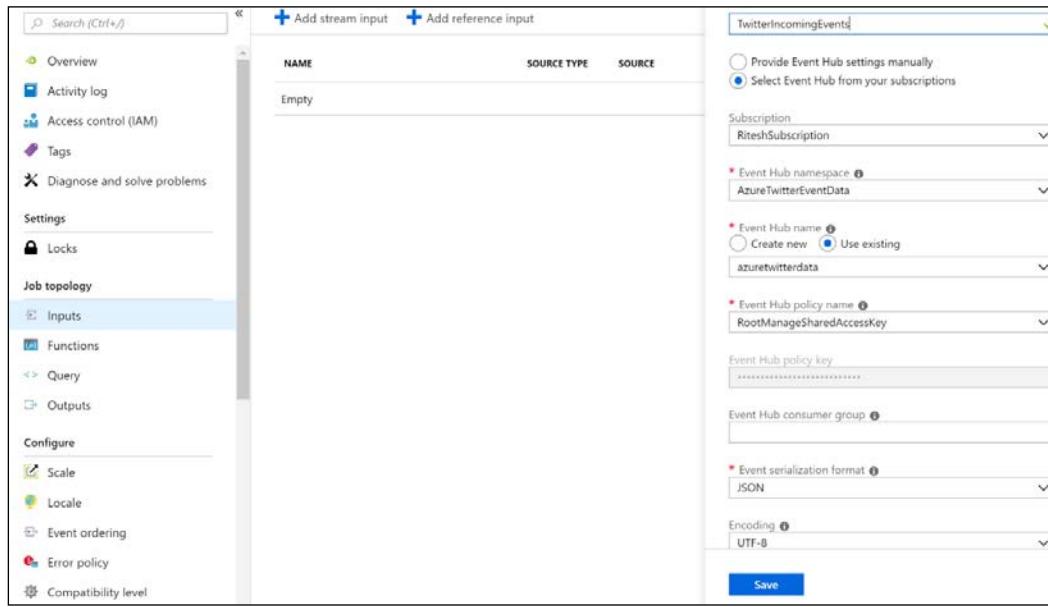


The screenshot shows the Azure Storage Explorer interface for the 'twittereventdata' storage account. On the left, there's a sidebar with options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Events, and Storage Explorer (preview). The main area displays a table of containers. The table has columns for NAME, LAST MODIFIED, PUBLIC ACCESS L., and LEASE STATE. There is one row visible with the NAME 'twitter', LAST MODIFIED as 1/25/2019, 8:57:50 AM, PUBLIC ACCESS set to Container, and LEASE STATE set to Available. There are also 'Container', Refresh, and Delete buttons at the top of the table.

NAME	LAST MODIFIED	PUBLIC ACCESS L...	LEASE STATE
twitter	1/25/2019, 8:57:50 AM	Container	Available

Creating Stream Analytics jobs

Let's create a new Stream Analytics job with a hosting environment on the cloud and set the streaming units to the default settings. The input for this Stream Analytics job comes from the event hub, and so we need to configure this from the **Inputs** menu:



The output for the Stream Analytics job is a Blob Storage account, so we need to configure the output accordingly. Provide a path pattern that is suitable for this exercise; for example, {datetime:ss} is the path pattern that we are using for this exercise:

StreamAzureTwitterJob - Outputs

NAME	SINK
TwitterData	Blob storage

TwitterData

Output alias: TwitterData

Provide Blob storage settings manually (radio button)

Select Blob storage from your subscriptions (radio button)

Subscription: RiteshSubscription

Storage account: twittereventdata

Storage account key: (redacted)

Container: Create new (radio button) Use existing (radio button)

Container name: twitter

Path pattern: (datetimestamp)

Date format: YYYY/MM/DD

Time format: HH

Save

The query is quite simple; we are just copying the data from the input to the output:

StreamAzureTwitterJob - Query

Inputs (1): TwitterIncomingEvents

Outputs (1): TwitterData

```

1 SELECT
2 *
3 INTO
4 [TwitterData]
5 FROM
6 [TwitterIncomingEvents]

```

Running the application

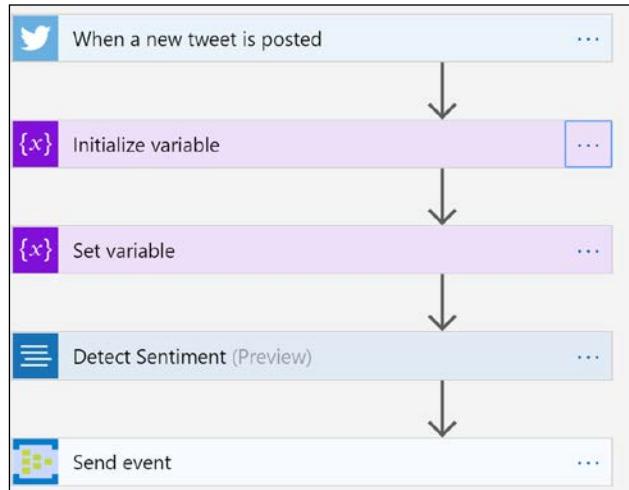
The logic app should be enabled and Stream Analytics should be running. Now, execute the logic app; it should have jobs created, as shown in the following screenshot:

The screenshot shows the Azure Logic Apps interface for the 'GetAzureTwitterData' logic app. The left sidebar includes options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Development Tools (Logic app designer, Logic app code view, Versions, API connections, Quick start guides, Release notes), Settings (Workflow settings, Access keys, Identity, Properties, Locks), and a Run trigger button. The main pane displays the 'Summary' section with details: Resource group (change) : TwitterAnalytic, Location : West Europe, Subscription (change) : RiteshSubscription, Subscription ID : 9755ffce-e94b-4332-9be8-1ade15e78909, Definition : 1 trigger, 1 action, Status : Enabled, Runs last 24 hours : 217 successful, 1275 failed, and Integration Account : ---. Below this is the 'Runs history' table, which lists two successful runs: one at 1/26/2019, 10:16 AM and another at 1/26/2019, 10:15 AM, both taking 299 Milliseconds.

The Storage Account container should get data, as shown in the following screenshot:

The screenshot shows the Azure Storage Explorer interface for the 'twitter' container. The left sidebar includes Overview, Access Control (IAM), Settings (Access policy, Properties, Metadata), and a search bar. The main pane shows a blob named '24/0_48c599701d414248959070c92c937b3a_1.json'. The blob properties table on the right provides detailed information: URL (https://twittereventdata.blob.core.windows.net/24/0_48c599701d414248959070c92c937b3a_1.json), LAST MODIFIED (1/25/2019, 9:06:25 AM), CREATION TIME (1/25/2019, 9:06:24 AM), TYPE (Block blob), SIZE (10.8 KiB), SERVER ENCRYPTED (true), ETAG (0x8D682CE4AF8642B), CONTENT-MDS (-), LEASE STATUS (Unlocked), LEASE STATE (Available), LEASE DURATION (-), COPY STATUS (-), and COPY COMPLETION TIME (-). A blue button at the bottom right says 'Undelete all snapshots'.

As an exercise, you can extend this sample solution and evaluate the sentiment of the tweets every three minutes. The Logic Apps workflow for such an exercise will be as follows:



For the exercise in detecting sentiment, you'll need to use the Text Analytics API, which should be configured before being used in Logic Apps.

Summary

This chapter focused on topics related to streaming events and temporal events. Events are raised from multiple sources, and in order to get insights in real time about activities and their related events, services such as Event Hubs and Stream Analytics play a significant role. Event Hubs is a service that helps in ingesting millions of events per second and temporally storing them. This data can be sent to Stream Analytics for processing. During processing, additional insights can be fetched using custom queries, and these insights can then be sent as outputs to multiple destinations. These services work on real-time data that isn't stored in any permanent storage.

In the next chapter, we will go through the process of creating an IoT solution using Azure IoT Hub and explore its various features.

15

Designing IoT Solutions

So far, we have been dealing with architectural concerns and their solutions in Azure in general. However, this chapter is not based on generalized architecture. In fact, it explores one of the most disruptive technologies of this century. This chapter will discuss the details of the **Internet of Things (IoT)** and Azure.

This chapter will specifically cover the following topics:

- Azure and IoT
- An overview of Azure IoT
- Device management:
 - Registering devices
 - Device-to-IoT-hub communication
- Scaling IoT solutions
- High availability of IoT solutions
- IoT protocols
- Using message properties for routing messages

IoT

To understand what IoT is, let's go back a few years.

The internet was invented during later half of last century and became widely available among us. During this time, almost everyone moved toward having a presence on the internet and started creating their own static web pages. Eventually, the static content became dynamic and content was generated on the fly, based on context. In nearly all cases, a browser was needed to access the internet. There was a plethora of browsers available, and without them, using the internet was a challenge.

During the first decade of this century, there was an interesting development that was emerging – the rise of handheld devices, such as mobile phones and tablets. Mobile phones started becoming cheaper and were available ubiquitously. The hardware and software capabilities of these handheld devices were improving considerably; so much so that people started using browsers on their mobile devices rather than on desktops. But one particularly distinct change was the rise of mobile apps. These mobile apps were downloaded from a store and connected to the internet to talk to backend systems. Toward the end of the last decade, there were millions of apps available with almost every conceivable functionality built into them. The backend system for these apps was built on the cloud so that they could be scaled rapidly. This was the age of connecting applications and servers.

But was this the pinnacle of innovation? What was the next evolution of the internet? Well, another paradigm has now been taking center stage: IoT. Instead of just mobile and tablet devices connecting to the internet, why can't other devices connect to the internet? Previously, such devices were available only in select markets; they were costly, not available to the masses, and had limited hardware and software capabilities. However, since the first part of this decade, the commercialization of these devices has been growing on a grand scale. These devices are becoming smaller and smaller, are more capable in terms of hardware and software, have more storage and compute power, can connect to the internet on various protocols, and can be attached to almost anything. This is the age of connecting devices to servers, applications, and other devices.

This has led to the formulation of the idea that IoT applications can, in fact, change the way that industries are operating. Newer solutions that were previously unheard of are beginning to be realized. Now, these devices can be attached to anything; they can get information and send it to a backend system that can assimilate information from all the devices and either take action on or report incidents.

Examples of IoT applications include vehicle tracking systems, which can track all the vital parameters of a vehicle and send details to a centralized data store for analysis; smart city services, such as tracking pollution levels, temperature, and street congestion; and agriculture-related activities, such as measuring soil fertility, humidity, and more.

IoT architecture

Before getting into Azure and its features and services with regard to IoT, it is important to understand the various components that are needed to create end-to-end IoT solutions.

Imagine that IoT devices across the globe are sending millions of messages every second to a centralized database. Why is this data being collected? Well, the answer is to extract rich information about events, anomalies, and outliers that are to do with whatever those devices are monitoring.

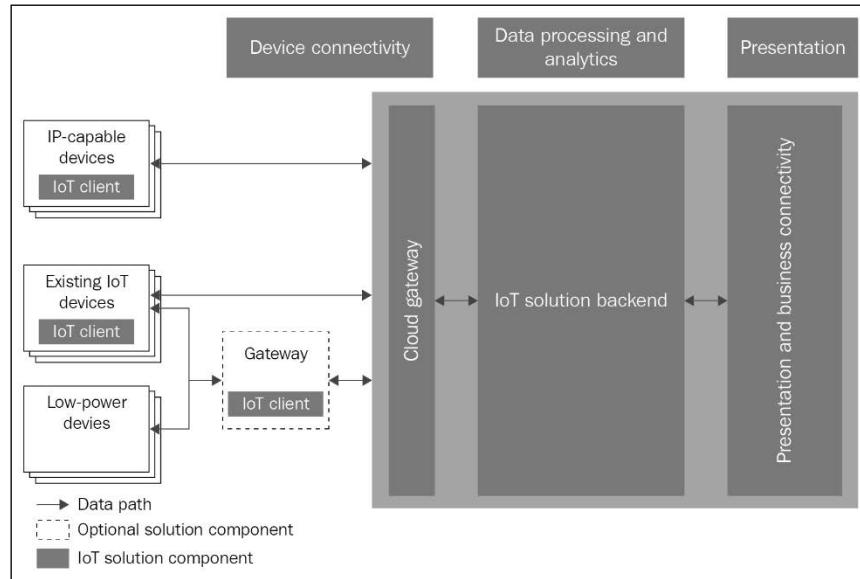
Let's understand this in more detail.

IoT architecture can be divided into distinct phases, as follows:

- Connectivity
- Identity
- Capture
- Ingestion
- Storage
- Transformation
- Analytics
- Presentation

The following diagram shows a generic IoT-based architecture. Data is generated or collected by devices and sent over to the cloud gateway. The cloud gateway, in turn, sends the data to multiple backend services for processing.

Cloud gateways are optional components; they should be used when the devices themselves are not capable of sending requests to backend services, either because of resource constraints or the lack of a reliable network. These cloud gateways can collate data from multiple devices and send it to backend services. The data can then be processed by backend services and shown as insights or dashboards to users:



Connectivity

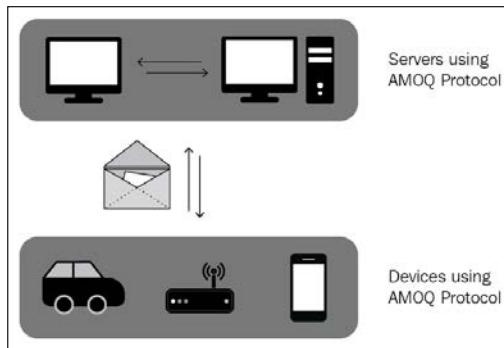
IoT devices need to communicate in order to connect to other devices. There are various connectivity types; for example, connectivity could exist between devices in a region, between devices and a centralized gateway, and between devices and an IoT platform.

In all such cases, IoT devices need connectivity capability. This capability could be in the form of internet connectivity, Bluetooth, infrared, or any other near-device communication.

However, some IoT devices might not have the capability to connect to the internet. In these cases, they can connect to a gateway which in turn has connectivity to the internet.

IoT devices use protocols to send messages. The major protocols are the **Advanced Message Queuing Protocol (AMQP)** and the **Message Queue Telemetry Transport (MQTT)** protocol.

Device data should be sent to an IT infrastructure. The MQTT protocol is a device-to-server protocol that devices can use to send telemetry data and other information to servers. Once the server receives a message through the MQTT protocol, it needs to transport the message to other servers using a reliable technology that is based on messages and queues. AMQP is the preferred protocol for moving messages between servers in the IT infrastructure in a reliable and predictable manner:



Servers receiving initial messages from IoT devices should send those messages to other servers for whatever processing is necessary, such as saving to logs, evaluation, analytics, and presentation.

Some devices do not have the capability to connect to the internet or do not support protocols that are compatible with other server technologies. To enable these devices to work with an IoT platform and the cloud, intermediate gateways can be used. Gateways help in onboarding devices whose connectivity and networking capability is slow and not consistent; such devices may use protocols that are not standard, or their capabilities may be limited in terms of resources and power.

In such circumstances, when devices need additional infrastructure to connect to backend services, client gateways can be deployed. These gateways receive messages from near devices, and forward (or push) them to the IT infrastructure and the IoT platform for further consumption. These gateways are capable of protocol translation if required.

Identity

IoT devices should be registered with the cloud platform. Devices that are not registered should not be allowed to connect to a cloud platform. The devices should be registered and be assigned an identity. A device should send its identity information when connecting to the cloud. If the device fails to send this identity information, the connectivity should fail. We will see, later in this chapter, how to generate an identity for a device using a simulated application.

Capture

IoT devices should be able to capture information. It should have the capability, for example, to read or monitor the moisture content in the air or in the soil. This information can be captured based on frequency – maybe even once per second. Once the information is captured, the device should be able to send it across to the IoT platform for processing. If a device does not have the capability to connect to the IoT platform directly, it can connect to an intermediary cloud gateway instead and have that push the captured information. The size of captured data and the frequency of capture are the most important aspects for the device. Whether a device should have local storage to be able to temporarily store captured data is another important aspect that should be considered. For instance, a device can work in offline mode if there is enough local storage available. Even mobile devices sometimes act as IoT devices connected to various instruments and have the capability to store data.

Ingestion

Data captured and generated by devices should be sent to an IoT platform that is capable of ingesting and consuming this data to extract meaningful information and insights out of it. The ingestion service is a crucial service because its availability and scalability affect the throughput of incoming data. If data starts getting throttled due to scalability issues, or if data is not able to be ingested due to availability issues, then it will be lost and the dataset might get biased or skewed.

Storage

IoT solutions generally deal with millions or even billions of records, spanning terabytes or even petabytes of data. This is valuable data that can provide insights on operations and their health. This data needs to be stored in such a way that analytics can be performed on it. Storage should be readily available for analytics, applications, and services to consume it. Storage solutions should provide adequate throughput and latency from a performance perspective, and be highly available, scalable, and secure.

Transformation

IoT solutions are generally data-driven and have considerably high volumes of data to deal with. Imagine that every car has a device and each one is sending messages every five seconds. If there were a million cars sending messages, this would be equal to 288 million messages per day and 8 billion messages per month.

Together, this data has lots of hidden information and insights; however, making sense of this kind of data just by looking at it is difficult. The data that is captured and stored by IoT devices can be consumed for solving business problems, but not all data that is captured is of importance. Just a subset of data might be needed to solve a problem. Additionally, the data that the IoT devices gather might not be consistent either. To ensure that the data is consistent and not biased or skewed, appropriate transformations should be executed upon it to make it ready for analysis. During transformation, data is filtered, sorted, removed, enriched, and transformed into a structure, such that the data can be consumed by components and applications further downstream.

Analytics

The data transformed in the previous step becomes the input for the analytics step. Depending on the problem at hand, there are different types of analytics that can be performed on transformed data.

The following are the different types of analytics that can be performed:

- **Descriptive analytics:** This type of analytics helps in finding patterns and details about the status of the IoT devices and their overall health. This stage identifies and summarizes the data for further consumption by more advanced stages of analytics. It will help in summarization, finding statistics related to probability, identifying deviation, and other statistical tasks.
- **Diagnostic analytics:** This type of analytics is more advanced than descriptive analytics. It builds on descriptive analytics and tries to answer queries about why certain things have happened. That is, it tries to find the root causes of events. It tries to find answers using advanced concepts, such as hypothesis and correlation.
- **Predictive analytics:** This type of analytics tries to predict things that have a high probability of happening in the future. It generates predictions that are based on past data; regression is one of the examples that is based on past data. Predictive analytics could, for example, predict the price of a car, behavior of stock in the stock market, when a car tire will burst, and more.
- **Prescriptive analytics:** This type of analytics is the most advanced. This stage helps in identifying the action that should be executed to ensure that the health of devices and solutions do not degrade, and identifying proactive measures to undertake. The results of this stage of analytics can help in avoiding future issues and eliminating the problems at their root causes.

Presentation

Analytics help in identifying answers, patterns, and insights based on data. These insights also need to be available to all stakeholders in formats that they can understand. To this end, dashboards and reports can be generated, statistically or dynamically, and then be presented to stakeholders. Stakeholders can consume these reports for further action and improve continuously on their solutions.

Azure IoT

Now we've learned about the various stages of end-to-end IoT solutions; each of these stages is crucial and their proper implementation is a must for any solution's success. Azure provides lots of services for each of these stages. Apart from these services, Azure provides IoT Hub, which is Azure's core IoT service and platform. It is capable of hosting complex, highly available, and scalable IoT solutions. We will dive deep into IoT Hub after going through some other services:

Devices	Device Connectivity	Storage	Analytics	Presentation & Action
	Events	SQL Database	Machine Learning	App Services
	Service Bus	Table/Blob Storage	Stream Analytics	Power BI
	External Data Sources	Cosmos DB	HDIInsight	Notification Hubs
		External Data Sources	Data Factory	Mobile Services
		Time Series Insights	Data Bricks	Logic Apps

Identity

Azure IoT Hub also provides services for authenticating devices. It provides an interface for generating unique identity hashes for each device. When devices send messages containing this hash, IoT Hub can authenticate them, after verifying in its own database for the existence of such hashes.

Capture

Azure provides IoT gateways that enable devices that do not comply with IoT Hub to be adapted and push data. Local or intermediary gateways can be deployed near devices in such a way that multiple devices can connect to a single gateway to send their information. Similarly, multiple clusters of devices with local gateways can also be deployed. There can be a cloud gateway deployed on the cloud itself, which is capable of accepting data from multiple sources and ingesting it for IoT Hubs.

Ingestion

An IoT Hub can be a single point of contact for devices and other applications. In other words, the ingestion of IoT messages is the responsibility of the IoT Hub service. There are other services, such as Event Hubs and the Service Bus messaging infrastructure, that can ingest incoming messages; however, the benefits and advantages of using IoT Hub for ingesting IoT data far outweigh those of using Event Hubs and Service Bus messaging. In fact, IoT Hubs have been made specifically for the purpose of ingesting IoT messages within the Azure ecosystem so that other services and components can act on them.

Storage

Azure provides multiple ways of storing messages from IoT devices. These storage accounts include storing relational data, schema-less NoSQL data, and blobs:

- **SQL database:** SQL database provides storage for relational data, JSON, and XML documents. It provides a rich SQL-query language and it uses a full-blown SQL server as a service. Data from devices can be stored in SQL databases if it is well defined and the schema will not need to undergo changes frequently.
- **Azure Storage:** Azure Storage provides table and blob storage. Table storage helps in storing data as entities, where the schema is not important. Blobs help in storing files in containers as blobs.
- **Cosmos DB:** Cosmos DB is a full-blown enterprise-scale NoSQL database. It is available as a service that is capable of storing schema-less data. It is a truly distributed database that can span continents, providing high availability and scalability of data.
- **External data sources:** Apart from Azure services, customers can bring or use their own data stores, such as a SQL server on Azure virtual machines, and can use them for storing data in a relational format.

Transformation and analytics

Azure provides multiple resources to execute jobs and activities on ingested data. Some of them are mentioned as follows:

- **Data Factory:** Azure Data Factory is a cloud-based data integration service that allows us to create data-driven workflows in the cloud for orchestrating and automating data movement and data transformation. Azure Data Factory helps to create and schedule data-driven workflows (called pipelines) that can ingest data from disparate data stores; process and transform data by using compute services such as **Azure HDInsight**, **Hadoop**, **Spark**, **Azure Data Lake Analytics**, and **Azure Machine Learning**; and publish output data to a data warehouse for **Business Intelligence (BI)** applications rather than a traditional **Extract-Transform-and-Load (ETL)** platform.
- **Azure HDInsight:** Microsoft and Hortonworks have come together to help companies by offering a big data analytics platform with Azure. HDInsight is a high-powered, fully-managed cloud service environment powered by Apache Hadoop and Apache Spark using Microsoft Azure HDInsight. It helps in accelerating workloads seamlessly using Microsoft and Hortonworks' industry-leading big data cloud service.
- **Azure stream analytics:** This is a fully-managed real-time data analytics service that helps in performing computation and transformation on streaming data. Stream analytics can examine high volumes of data flowing from devices or processes, extract information from the data stream, and look for patterns, trends, and relationships.
- **Machine learning:** Machine learning is a data science technique that allows computers to use existing data to forecast future behaviors, outcomes, and trends. Using machine learning, computers learn without being explicitly programmed. Azure Machine Learning is a cloud-based predictive analytics service that makes it possible to quickly create and deploy predictive models as analytics solutions. It provides a ready-to-use library of algorithms to create models on an internet-connected PC, and deploy predictive solutions quickly.

Presentation

After appropriate analytics have been conducted on data, the data should be presented to stakeholders in a format that is consumable by them. There are numerous ways in which insights from data can be presented. This includes presenting data through web applications that are deployed using Azure App Service, sending data to notification hubs that can then notify mobile applications, and more. However, the ideal approach for presenting and consuming insights is by using **Power BI** reports and dashboards. Power BI is a Microsoft visualization tool that is used for rendering dynamic reports and dashboards on the internet.

IoT Hubs

IoT projects are generally complex in nature. The complexity arises because of the high volume of devices and data. Devices are embedded across the world; for example, monitoring and auditing devices, which are used for storing data, transforming and analyzing petabytes of data, and finally taking action(s) based on insights. Moreover, these projects have long gestation periods, and their requirements keep changing because of timelines.

If an enterprise wants to embark on a journey with an IoT project, sooner rather than later, then they will quickly realize that the problems we mentioned are not easily solved. These projects require enough hardware in terms of computing and storage to cope, and services that can work with high volumes of data.

IoT Hub is a platform that is built to help ease and enable IoT projects for faster, better, and easier delivery. It provides all the necessary features and services, including the following:

- Device registration
- Device connectivity
- Field gateways
- Cloud gateways
- Implementation of industry protocols, such as AMQP and the MQTT protocol
- A hub for storing incoming messages
- The routing of messages based on message properties and content
- Multiple endpoints for different types of processing
- Connectivity to other services on Azure for real-time analytics and more

Protocols

Azure IoT Hub natively supports communication over the MQTT, AMQP, and HTTP protocols. In some cases, devices or field gateways might not be able to use one of these standard protocols and will require protocol adaptation. In such cases, custom gateways can be deployed. A custom gateway can enable protocol adaptation for IoT Hub endpoints by bridging the traffic to and from the IoT Hub.

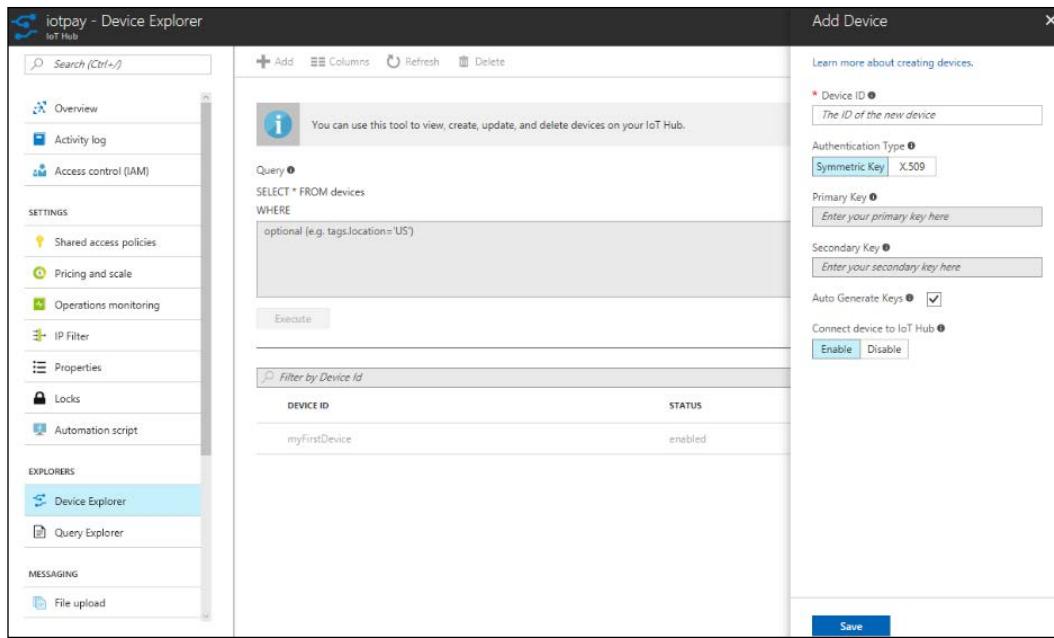
Device registration

Devices should be registered before they can send messages to an IoT Hub. The registration of devices can be done manually using the Azure portal or it can be automated using the IoT Hub SDK. Azure also provides sample simulation applications through which it becomes easy to register virtual devices for development and testing purposes. There is also a Raspberry Pi online simulator that can be used as a virtual device, and then, obviously, there are other physical devices that can be configured to connect to the IoT Hub.

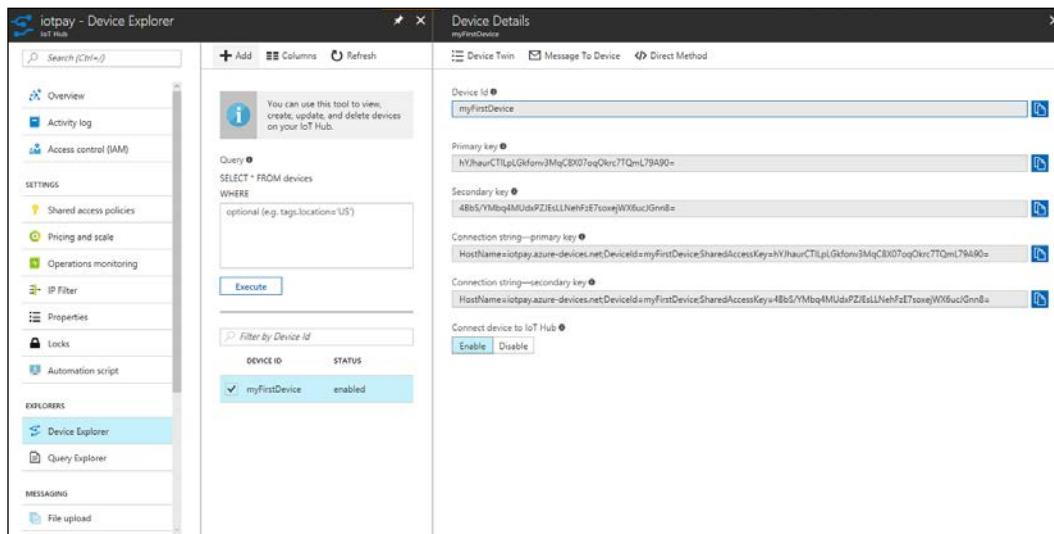
If readers want to simulate a device from a local PC that is generally used for development and testing purposes, then there are tutorials available in the Azure documentation in multiple languages. These are available at <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-get-started-simulated>.

The Raspberry Pi online simulator is available at <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-raspberry-pi-web-simulator-get-started>, and for physical devices that need to be registered with IoT Hub, the steps mentioned at <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-get-started-physical> should be used.

For manually adding a device using the Azure portal, IoT Hub provides a **Device Explorer** menu, which can be used for configuring a new device:



After the device identity is created, a primary key connection string for IoT Hub should be used in each device to connect to it:



Message management

After devices are registered with IoT Hub, they can start interacting with it. This interaction could be from the device to the cloud, or from the cloud to the device.

Device-to-cloud messaging

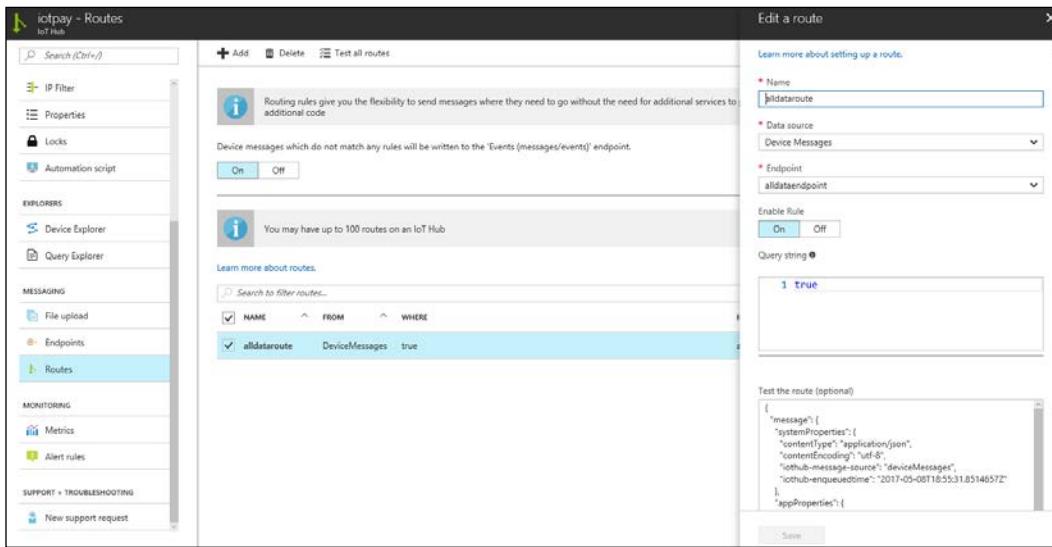
One of the best practices that must be followed in this communication is that although the device might be capturing a lot of information, only data that is of any importance should be transmitted to the cloud. The size of the message is very important in IoT solutions due to the fact that IoT solutions generally have very high volumes of data. Even 1 KB of extra data can result in a GB of storage and processing wasted. Each message has properties and payloads; properties define the metadata for the message. This metadata contains data about the device, identification, tags, and other properties that are helpful in routing and identifying messages.

Devices or cloud gateways should connect to IoT Hub to transfer data. IoT Hub provides public endpoints that can be utilized by devices to connect and send data. IoT Hub should be considered as the first point of contact for backend processing. IoT Hub is capable of further transmission and routing of these messages to multiple services. By default, the messages are stored in event hubs. Multiple event hubs can be created for different kinds of messages:

NAME	ENDPOINT
Cloud to device feedback	messages/servicebound/feedback
Events	messages/events

NAME	ENDPOINT TYPE	NAMESPACE/ENTITY	STATUS
alldataendp...	Event Hub	eventhubpay/allevents	Healthy.

Messages can be routed to different endpoints based on the message header and body properties, as shown in the following screenshot:



Messages in an IoT Hub stay there for seven days by default, and their size can go up to 256 KB.

There is a sample simulator provided by Microsoft for simulating sending messages to the cloud. It is available in multiple languages; the C# version can be viewed at <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-csharp-csharp-c2d>.

Cloud-to-device messaging

Azure IoT Hub is a managed service providing a bi-directional messaging infrastructure. Messages can be sent from the cloud to devices, and then based on the message, the devices can act on them.

There are three types of cloud-to-device messaging patterns:

- Direct methods require immediate confirmation of results. Direct methods are often used for the interactive control of devices, such as opening and closing garage shutters. They follow the request-response pattern.
- Setting up device properties using Azure IoT provides **device twin** properties. For example, you can set the telemetry send interval to 30 minutes. Device twins are JSON documents that store device state information (such as metadata, configurations, and conditions). An IoT Hub persists a device twin for each device in the IoT Hub.
- Cloud-to-device messages are used for one-way notifications to the device app. This follows the fire-and-forget pattern.

Security

Security is an important aspect of IoT-based applications. IoT-based applications comprise devices that use the public internet for connectivity to backend applications. Securing devices, backend applications, and connectivity from malicious users and hackers should be considered a top priority for the success of these applications.

Security in IoT

IoT applications are primarily built around the internet, and security should play a major role in ensuring that a solution is not compromised. Some of the most important security decisions affecting IoT architecture are listed as follows:

- Devices using HTTP versus HTTPS REST endpoints – REST endpoints secured by certificates ensure that messages transferred from a device to the cloud and vice versa are well encrypted and signed. The messages should make no sense to an intruder and should be extremely difficult to crack.
- If devices are connected to a local gateway, the local gateway should connect to the cloud using a secure HTTP protocol.
- Devices should be registered to IoT Hub before they can send any messages.
- The information passed to the cloud should be persisted into storage that is well protected and secure. Appropriate SAS tokens or connection strings that are stored in Azure Key Vault should be used for connection.
- Azure Key Vault should be used to store all secrets, passwords, and credentials, including certificates.

Scalability

Scalability for IoT Hub is a bit different than for other services. In IoT Hub, there are two types of messages:

- **Incoming:** Device-to-cloud messages
- **Outgoing:** Cloud-to-device messages

Both need to be accounted for in terms of scalability.

IoT Hub provides a couple of configuration options during provision time to configure scalability. These options are also available post-provisioning and can be updated to better suit the solution requirements in terms of scalability.

The scalability options that are available for IoT Hub are as follows:

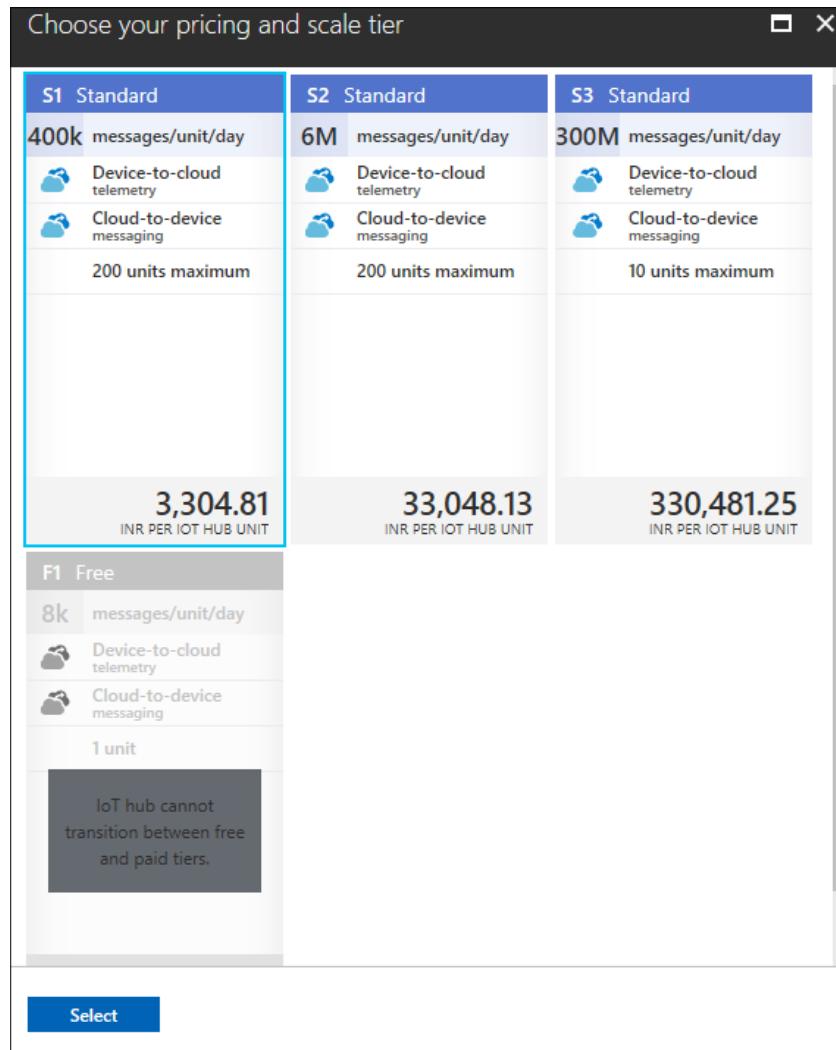
- The **Stock Keeping Unit (SKU)** edition, which is the size of the IoT Hub
- The number of units

The SKU edition

The SKU in IoT Hub determines the number of messages a hub can handle per unit per day, and this includes both incoming and outgoing messages. There are four tiers, as follows:

- **Free:** This allows for 8,000 messages per unit per day and allows both incoming and outgoing messages. A maximum of 1 unit can be provisioned. This edition is suitable for gaining familiarity and testing out the capabilities of the IoT Hub service.
- **Standard (S1):** This allows for 400,000 messages per unit per day and allows both incoming and outgoing messages. A maximum of 200 units can be provisioned. This edition is suitable for a small number of messages.
- **Standard (S2):** This allows for six million messages per unit per day and allows both incoming and outgoing messages. A maximum of 200 units can be provisioned. This edition is suitable for a large number of messages.

- **Standard (S3):** This allows for 300 million messages per unit per day and allows both incoming and outgoing messages. A maximum of 10 units can be provisioned. This edition is suitable for a very large number of messages:

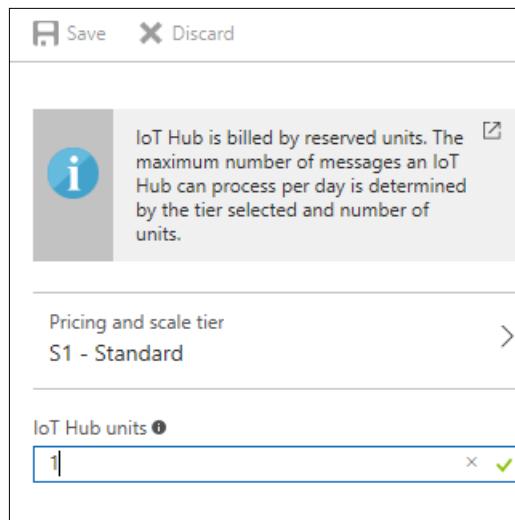


An astute reader will have noticed that the Standard S3 tier allows for a maximum of only 10 units, compared to other standard units that allow for 200 units. This is directly related to the size of the compute resources that are provisioned to run IoT services. The size and capability of virtual machines for Standard S3 are significantly higher compared to other tiers where the size remains the same.

Units

Units define the number of instances of each SKU running behind the service. For example, 2 units of the Standard S1 SKU tier will mean that the IoT Hub is capable of handling $400K * 2 = 800K$ messages per day.

More units will increase the scalability of the application:



High availability

IoT Hub is a **Platform as a Service (PaaS)** platform from Azure. Customers and users do not directly interact with the underlying number and size of virtual machines on which the IoT Hub service is running. Users decide on the region, the SKU of the IoT Hub, and the number of units for their application. The rest of the configuration is determined and executed by Azure behind the scenes. Azure ensures that every PaaS service is highly available by default. It does so by ensuring that multiple virtual machines provisioned behind the service are on separate racks in the data center. It does this by placing those virtual machines on an availability set and on a separate fault and update domain. This helps in high availability for both planned and unplanned maintenance. Availability sets take care of high availability at the data center level.

Summary

IoT is one of the biggest upcoming technologies of this decade and it is already disrupting industries. Things that sounded impossible before are now suddenly possible. IoT Hub is a platform that eases the creation and delivery of IoT solutions to the customer in a faster, better, and cheaper way. It provides implementations of all industry protocols, such as the MQTT protocol and AMQP, along with field gateways that can adapt non-standard devices. It provides high availability, scalability, and security features to both messages and the overall solution. It provides connectivity to a large number of Azure services and helps in routing messages to multiple different endpoints, each capable of processing messages. IoT can fast-track the entire development life cycle and help expedite time-to-market for companies.

Azure provides numerous data storage options and services. This is the last chapter of the book, and I hope you really enjoyed reading all the chapters and are ready to architect solutions on Azure.

Cheers!

Other Books You May Enjoy

If you enjoyed this book, you may be interested in these other books by Packt:



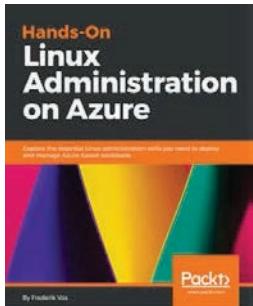
Hands-On Azure for Developers

KamilMrzyg艂d

ISBN: 978-1-78934-062-4

- ▶ Implement serverless components such as Azure functions and logic apps
- ▶ Integrate applications with available storages and containers
- ▶ Understand messaging components, including Azure Event Hubs and Azure Queue Storage
- ▶ Gain an understanding of Application Insights and other proper monitoring solutions
- ▶ Store your data with services such as Azure SQL and Azure Data Lake Storage
- ▶ Develop fast and scalable cloud applications

Another Book You May Enjoy —————



Hands-On Linux Administration on Azure

Frederik Vos

ISBN: 978-1-78913-096-6

- ▶ Understand why Azure is the ideal solution for your open source workloads
- ▶ Master essential Linux skills and learn to find your way around the Linux environment
- ▶ Deploy Linux in an Azure environment
- ▶ Use configuration management to manage Linux in Azure
- ▶ Manage containers in an Azure environment
- ▶ Enhance Linux security and use Azure's identity management systems
- ▶ Automate deployment with Azure Resource Manager (ARM) and Powershell
- ▶ Employ Ansible to manage Linux instances in an Azure cloud environment

Leave a review – let other readers know what you think

Please share your thoughts on this book with others by leaving a review on the site that you bought it from. If you purchased the book from Amazon, please leave us an honest review on this book's Amazon page. This is vital so that other potential readers can see and use your unbiased opinion to make purchasing decisions, we can understand what our customers think about our products, and our authors can see your feedback on the title that they have worked with Packt to create. It will only take a few minutes of your time, but is valuable to other potential customers, our authors, and Packt. Thank you!

Index

A

- ACID** 367
- actions** 188
- Active Directory (AD)** 72, 381
- activity, Logic Apps**
 - about 188
 - actions 188
 - triggers 188
- activity logs** 82
- Advanced Message Queuing Protocol (AMQP)** 422, 448
- alerts**
 - about 94-97
 - runbooks, executing on 97-101
- AlwaysOn configuration** 368
- analytics, IoT**
 - descriptive analytics 451
 - diagnostic analytics 451
 - predictive analytics 451
 - prospective analytics 451
- Apache Kafka** 422
- API server** 300
- application logs** 83
- Application Request Routing (ARR)** 36
- App Service containers**
 - about 306
 - deploying 306-310
- App Service function**
 - settings 154
- ARM architecture** 7
- ARM, features**
 - extensible 11
 - idempotent 11
- locks 11
- multi-region 11
- policies 11
- RBAC 10
- tags 11
- ARM templates**
 - about 16, 106-108
 - deployment modes 17
 - used, for deploying resource groups 109-112
 - writing, with composed templates 129
 - writing, with leaf-level templates 129
- attack surface area**
 - reducing 68, 69
- availability zone**
 - about 24
 - reference 24
- Avro** 431
- Azure**
 - about 4, 5
 - activity logs 82
 - application logs 83
 - as intelligent cloud 6
 - deployment patterns 3
 - diagnostic logs 83
 - flexible cloud 361
 - for DevOps 361, 362
 - guest operating system logs 83
 - host operating system logs 83, 84
 - load balancers 32
 - monitoring 82
 - open cloud 361
 - unified management 361
- Azure Active Directory (Azure AD)** 10
- Azure application gateway** 36

Azure Application Insights	configuration 152 cost plans 154, 155 creating 156-160 features 146 keys 152 monitoring 150 platform, configuration 152 runtime 147 triggers 147-149 triggers, reference 150 types 156 use cases 155, 156
Azure Automation	about 199, 355 account, provisioning 356 configuration, assigning to nodes 360 DSC configuration, compiling 359 DSC configuration, creating 357 DSC configuration, importing 358 server, browsing 360
Azure CLI 16	
Azure CLI 2	reference 16
Azure cloud services 366	
Azure Container Instances	using 293-296
Azure Container Registry	about 282, 283 advantages 283 SKUs 285 working with 284-293
Azure Cosmos DB 30	
Azure Data Lake Analytics 454	
Azure DevOps	about 328-330 and Jenkins 353, 354 Git 331 TFVC 330, 331
Azure DevOps organization	provisioning 333
Azure Event Grid	about 177, 178 architecture 178, 180 custom events 185-187 reference 180 resource events 181-185
Azure Functions	about 200 advantages 145, 146 App Service function, settings 154 authentication 151, 152 authorization 151, 152 bindings 147-149 bindings, reference 150 components 147
Azure Functions proxies	about 163 reference 164
Azure governance features	background 277 implementing, example 277
Azure HDInsight 454	
Azure high availability	about 23 concepts 23 load balancing 25
Azure high availability, concepts	availability sets 23 availability zones 24, 25 fault domain 24 update domain 24
Azure high availability, load balancing	resources 25
Azure Hybrid Benefit 256	
Azure IoT	about 452 capture 453 identity 452 ingestion 453 presentation 455 storage 453 transformation and analytics 454
Azure Key Vault	about 78 provisioning 333
Azure Kubernetes architecture 301	
Azure Kubernetes Service	about 297, 298, 312 provisioning 301-306

Azure load balancer
Azure Traffic Manager 37, 38
Azure application gateway 36
Azure load balancers 33
features 32
internal load balancing 34, 35
port forwarding 36
public load balancing 33, 34

Azure locks
about 279
using 273, 274

Azure Logic Apps
about 187, 200
activity 188
connectors 188

Azure Machine Learning 454

Azure Monitor
about 79, 84
logs 79

Azure policies
about 269, 270, 278
allowed fields 272
Application Insights 279
built-in policies 270
deployments to certain locations 278
diagnostic logs 279
languages 270, 272
resource groups, tags 278
resources tags 278

Azure portal 15

Azure pricing calculator
about 257-259
reference 257

Azure pricing models 255

Azure RBAC
about 274, 275
custom roles 277

Azure regions
high availability, across 40
high availability 39

Azure reserved virtual machine instances 256

Azure Resource Manager (ARM)
about 6, 82, 105, 121, 414
advantages 8
concepts 8
resources 10

resource groups 9
resource instances 10
resource providers 9
resource types 9

Azure REST API 16

Azure scalability
concepts 44

Azure scalability, concepts
auto scaling 46
scaling 44
scaling down 45
scaling in 46
scaling out 45
scaling up 45

Azure security
about 63
features 63
reference 64

Azure Security Center 80

Azure Service Manager (ASM)
limitations 7

Azure solutions
guest updates 56
maintenance 55
upgrade 55

Azure SQL 75-77

Azure SQL Database
about 369
Elastic pool 369
Managed Instance 370
Single Instance 369

Azure SQL Database, application features
about 370
columnar store 370
In-memory OLTP 370

Azure SQL elastic pools
reference link 382

Azure SQL replication 31

Azure Storage
data migrating, to Data Lake
Gen2 Storage 396

Azure Storage account 334

Azure Subscription 7

Azure table storage 31

Azure tags
about 266
benefits 266, 267

resource groups versus resources 269
with Azure Resource Manager
 templates 268
with PowerShell 268
Azure Traffic Manager
 about 37, 38
 methods 38

B

best practices
 about 260
 compute best practices 260, 261
 general best practices 263
 Platform as a Service (PaaS) best
 practices 262
 storage best practices 261, 262
big data processing
 about 395
 data, ingesting 395
 data, presenting 396
 data, processing 395
 data, storing 396
billing 248-251
billing APIs 255
build 61
Business Intelligence (BI) 454

C

cloud computing
 about 2
 advantages 3
cloud provider 2
Cloud Solution Provider (CSP) 257
cloud-to-device messaging 459, 460
Comma-Separated Values (CSV) 431
computing high availability 26, 27
configuration management
 about 320
 Ansible 322
 Chef 322
 Desired State Configuration (DSC) 321
 Puppet 322

configuration-management server/service
 provisioning 333
connected architecture
 creating, with functions 172-176
connectors
 about 188
 reference 188
consumer groups 428
container options
 comparing 311
containers
 about 12, 350
 advantages 144
 benefits 13
 disadvantages 144
 Docker 351
 Dockerfile 351
 in Azure Container Instances 313
 in Azure Functions 314
 in Service Fabric 314
 on Azure App Service 313
 on virtual machines 311
 on virtual machines, with Kubernetes
 as orchestrator 312
Content Delivery Network (CDN) 262
continuous deployment
 about 325
 acceptance testst 327
 staging environment deployment 327
 test automation 326
 test environment deployment 326
 to production 327
continuous integration
 about 322, 323
 build automation 324
 packaging 324
 test automation 324
controller manager 300
copy data activity 412
create, retrieve, update, or delete (CRUD) 16
Cross-Origin Resource Sharing (CORS) 74
cross-subscription
 deploying, with linked templates 116-119
 deployments, example 113, 116
custom Azure Event Grid topic 200

custom topics
 endpoint 180
 keys 180

D

Database Throughput Units (DTUs) 385

Data Factory
 activities 393
 datasets 393
 integration runtime 394
 linked services 393
 pipelines 393
 primer on 393
 versions 394

data high availability 30

data high availability, resources
 Azure Cosmos DB 30
 Azure SQL replication 31
 Azure table storage 31

data integration 391, 392

Data Lake Gen2 Storage
 data migrating, from Azure Storage 396

Data Lake Storage
 big data processing 395
 primer on 394

data migrating
 Azure Data Factory Pipeline,
 provisioning 399, 400
 copy data activity, adding 412, 413
 Data Lake Gen2 service, creating 398
 dataset, creating 405-410
 final result 417
 from Azure Storage, to Data Lake Gen2
 Storage 396
 new resource group, provisioning 396
 pipeline, creating 411, 412
 pipeline, publishing 413-416
 repository settings 401-404
 source storage account, preparing 396
 Storage account, provisioning 397, 398

demilitarized zone (DMZ) 69

deployment models
 about 368
 databases, hosted as managed services 369
 databases, on Azure virtual machines 368

deployment modes, ARM templates
 complete 17
 incremental 17

deployment patterns, Azure
 about 3
 IaaS 4
 PaaS 4
 SaaS 4

deployments 124

Destination Network Address Translation (DNAT) 300

device-to-cloud messaging 458, 459

DevOps
 about 316-318
 for PaaS solutions 335
 implementation 317
 practices 319
 preparing for 331, 333

DevOps for container-based (IaaS) solutions
 about 350
 build pipeline 351
 containers 350
 release pipeline 352

DevOps for PaaS solutions
 about 335, 336
 Azure App Services 336
 Azure SQL 337
 build-and-release pipeline 337-344
 deployment slots 337

DevOps for virtual machine (IaaS)-based solutions
 about 346
 Azure public load balancers 347
 Azure Virtual Machines 347
 build pipeline 348
 release pipeline 349

DevOps practices
 configuration management 320
 continuous delivery 327
 continuous deployment 325
 continuous integration 322, 323
 continuous learning 327, 328

diagnostic logs 83

disaster recovery
 versus high availability 22

DNS CNAME 37

Docker 14

Docker client 14
Docker daemon 14
Docker Hub 282
Domain Name Service (DNS) 37
DTU-based pricing 385-387
Durable Functions
 about 165, 166
 creating, steps 166-172
durable orchestrator trigger 166

E

elastic pool 381, 382
end-to-end solution, creating with serverless technologies
 about 197
 architecture 198
 implementation 200
 implementation, steps 201-244
 problem statement 197
 solution 198
 testing 244
 vision 197
Enterprise Agreement
 about 257
 references 253
Enterprise Agreement customers 253
etcd 299
Event Consumer 424
event-driven design 177
event-driven function
 creating 160-163
Event Hubs
 about 422
 architecture 423-427
 consumer groups 428
 pricing tiers 422
 throughput 429
Event Producers 424
events 420
event streaming 420, 421
Extract-Transform-Load (ETL)
 about 391, 392, 454
 Extract 392
 Load 393
 Transform 392

F

finaldata 398
firewalls
 about 66
 designing 67
free-flow configurations 128
functions
 connected architecture, creating with 172-176
Functions as a Service (FaaS) 145, 147

G

General Availability (GA) 9
Git 331
guest operating system logs 83, 84

H

Hadoop 454
Hardware Security Modules (HSM) 197
high availability
 about 20
 across, Azure regions 40
 affecting, factors 21
 application 41
 architectural considerations 38
 best practices 41
 building, with software in application 31
 data management 42
 deployment strategy 41
 monitoring 42
 versus disaster recovery 22
 platforms 29, 30
 versus scalability 22
 within Azure regions 39
high availability, affecting factors
 application deployment architecture 22
 planned maintenance 21
 unplanned maintenance 21
Host Container System Shim (HCSShim) 14
host operating system logs 83
HTTP 422
hybrid cloud 5

I

IaaS scalability 50
IaaS security
 about 64
 attack surface area, reducing 68, 69
 firewalls 66
 jump servers, implementing 69
 network security groups 64, 65
 NSG, designing 66
idempotent template 106
Infrastructure as a Service (IaaS) 4, 28, 141, 293, 365
Infrastructure as Code (IaC) 8, 105
integration runtime
 Azure 394
 Azure SQL Server Integration Services (SSIS) 394
 self-hosted 394
intelligent cloud
 interacting with 14
Internet Information Services (IIS) 27
Internet of Things (IoT) 179, 445
Internet Service Provider (ISP) 1
Invoice Download API 255
invoicing 252
IoT architecture
 about 447
 analytics 451
 capture 450
 connectivity 448, 449
 identity 449
 ingestion 450
 presentation 452
 storage 450
 transformation 450
IoT Hubs
 about 455
 device registration 456
 high availability 463
 message management 458
 protocols 456
 scalability 461
 security 460

J

JavaScript Object Notation (JSON) 16, 322, 431
Jenkins
 reference 354
jump servers
 implementing 69
 reference 69
 using 69

K

known configurations 129-139
Kubelet 299, 300
Kube-Proxy 299, 300
Kubernetes 297
Kubernetes architecture
 about 299
 API server 300
 controller manager 300
 Kubelets 300
 Kube-Proxy 300
 master nodes 299
 pods 300
 replication controller 300
Kudu 147

L

linked templates
 about 125, 126
 used, for deploying
 cross-subscription 116-119
 used, for deploying
 resource-group 116-119
Linux
 deployment, reference 354
load balancers
 in Azure 32
Locally-redundant storage (LRS) 397
log analytics
 about 70, 84, 87
 agents 90, 92
 alerts 94-97
 provisioning 334
 categories 70

provisioning 88-90
search capabilities 92
solutions 93
logic app
working on 188-196

M

Managed Instance 383
managed services 369
management tools 335
master nodes 299
message management, IoT Hubs
 cloud-to-device messaging 459, 460
 device-to-cloud messaging 458, 459
Message Queue Telemetry Transport (MQTT) 448
Microsoft.Compute Namespace 9
monitoring tools 334

N

N+1 design 58
nested templates 126, 128
Network Address Translation (NAT)
 rules 34
network security groups
 about 64, 65
 designing 66

O

OLTP applications
 about 367
 Atomicity 367
 Consistency 367
 Durability 367
 Isolation 367
Online Transaction Processing (OLTP) 365
operating system (OS) 20
Operations Management Suite (OMS) 178

P

PaaS high availability 28, 29
PaaS scalability
 about 46, 47
 auto scaling, properties 49, 50

scaling down 48
scaling in 49
scaling out 49
scaling up 48
PaaS security
 about 69
 Azure Key Vault 78
 Azure SQL 75-77
 Log Analytics 70
 storage 71-75
partition 427
partition key 427
pay-as-you-go accounts 256
pay-as-you-go payment mechanism 247
Platform as a Service (PaaS) 4, 19, 141, 312, 365, 463

pods 300
Postman
 about 171
 download link 171

PowerBI
 download link 102
 integrating 101, 103
Power Query Formula Language (M Language) 102
PowerShell 15

PowerShell Gallery 15
pricing tiers, Event Hubs

 basic 422
 standard 422

primer
 on Data Factory 393
 on Data Lake Storage 394
Private (no anonymous access) 116
private repositories 282
public repository 282
pub/sub pattern 178

R

Raspberry Pi online simulator
 reference 456
RateCard API 255
rawdata 398
redundancy 23
registry 282
relational databases 367

Replication 397
replication controller 300
Representational State Transfer (REST) 9
resource groups 9
 deploying, with ARM templates 109-111
 deployments, deploying with linked
 templates 116-119
 deployments, example 113, 116
 resources, deploying across 112
resource instances 10
resource providers 9, 254
resources 10
resource types 9
Resource Usage API 255
Rest 379
Role-Based Access Control (RBAC)
 about 6, 72, 265
 comparing, with locks 277
 for Company Inc 278
runbooks
 about 199
 executing, on alerts 97-101

S

sample application, with Event Hubs and Stream Analytics
 about 434
 event hub, creating 436
 Event Hubs namespace, creating 435
 logic app, provisioning 436-438
 new resource group, provisioning 434
 running 442
 storage account, provisioning 439
 storage container, creating 439
 Stream Analytics jobs, creating 440

scalability
 about 42, 43
 versus high availability 22
 versus performance 44

scalability, IoT Hubs
 SKU edition 461, 462
 units 463

Secure Socket Layer (SSL) 422

security
 about 60
 auditing 78

Azure Monitor 79
Azure Security Center 80
life cycle 61, 62
measures 60
monitoring 78, 81, 82
security, IoT 460
security, life cycle
 reference 62
security, measures
 authentication 60
 authorization 60
 confidentiality 60
 integrity 60
serverless technology
 principles 145
service level agreement (SLA) 20, 21, 142
services, Azure
 reference 5

severless
 about 142
 advantages 143, 144
 disadvantages 142
 evolution of 142, 143

shared access signature (SAS) 4, 72, 407
Simple Mail Transfer Protocol (SMTP) 219
Single Instance
 about 370
 backups 372
 geo-replication 373, 374
 high availability 370
 scalability 375

Single Instance, security
 about 375
 Azure Active Directory (AD),
 integrating 381
 Azure SQL Server, on dedicated
 networks 377, 378
 dynamic data masking 380
 encrypted databases, at rest 379, 380
 firewall 376

Single Responsibility Principle
 about 123
 deployments resources 124, 125
 features 123
 modular templates 124

single template, issues
about 122
dependency abuse 122
large templates, troubleshooting 122
no reusability 123
reduced agility 122
reduced flexibility, in changing
templates 122
source images 334
**Source Network Address Translation
(SNAT)** 300
Spark 454
SQL database pricing
about 385
appropriate pricing model,
selecting 388, 389
DTU-based pricing 385-387
vCPU-based pricing 387, 388
stock keeping units (SKU) 47
storage, Azure IoT
Azure Storage 453
Cosmos DB 453
external data sources 453
SQL database 453
Storage Explorer tool
URL, for downloading 417
storage high availability 28
Stream Analytics
about 430-433
hosting environment 433
Streaming Units 433, 434
subscribers 178
subscription
about 254
resources, deploying across 112
**System Center Operations Manager
(SCOM)** 84, 363

T

TFVC 330
threat mitigation 61
transformation and analytics, Azure IoT
Azure HDInsight 454
Azure stream analytics 454
Data Factory 454
machine learning 454

Transmission Control Protocol (TCP) 25
Transparent Data Encryption (TDE) 76, 380
Transport Layer Security (TLS) 422
triggers 188
T-shirt sizing configurations 129

U

uniform resource locator (URI) 407
unmanaged databases 368
usage and quota information 254

V

vCPU-based pricing 387, 388
virtual CPUs (vCPUs) 387
Virtual Hard Disk (VHD) files 28
virtualization 12
virtual machine (VM)
about 5, 19
advantages 142
disadvantages 143
Virtual Private Network (VPN) 5
Visual Studio Team Services (VSTS) 4
VM high availability 26
VM scale sets (VMSS)
about 50, 51
application, updates 56
architecture 52
auto scaling 53, 54
capacity 53
Horizontal scaling, versus vertical
scaling 53
image, updates 56
scaling 52
vertical scaling, versus horizontal
scaling 53
VMs, planned maintenance
reference 21
VMSS, scaling
bare-metal instances, versus dormant
instances 57
best practices 57
caching 58
concurrency 57
Content Distribution Network (CDN) 58

dormant instances, versus bare-metal
instance 57
maximum instances, configuring 57
minimum instances, configuring 57
N+1 design 58
scaling out, preference 57
stateless 58

W

web application firewall (WAF) 40
workflows
about 164, 165
features 164

