GIGAOM

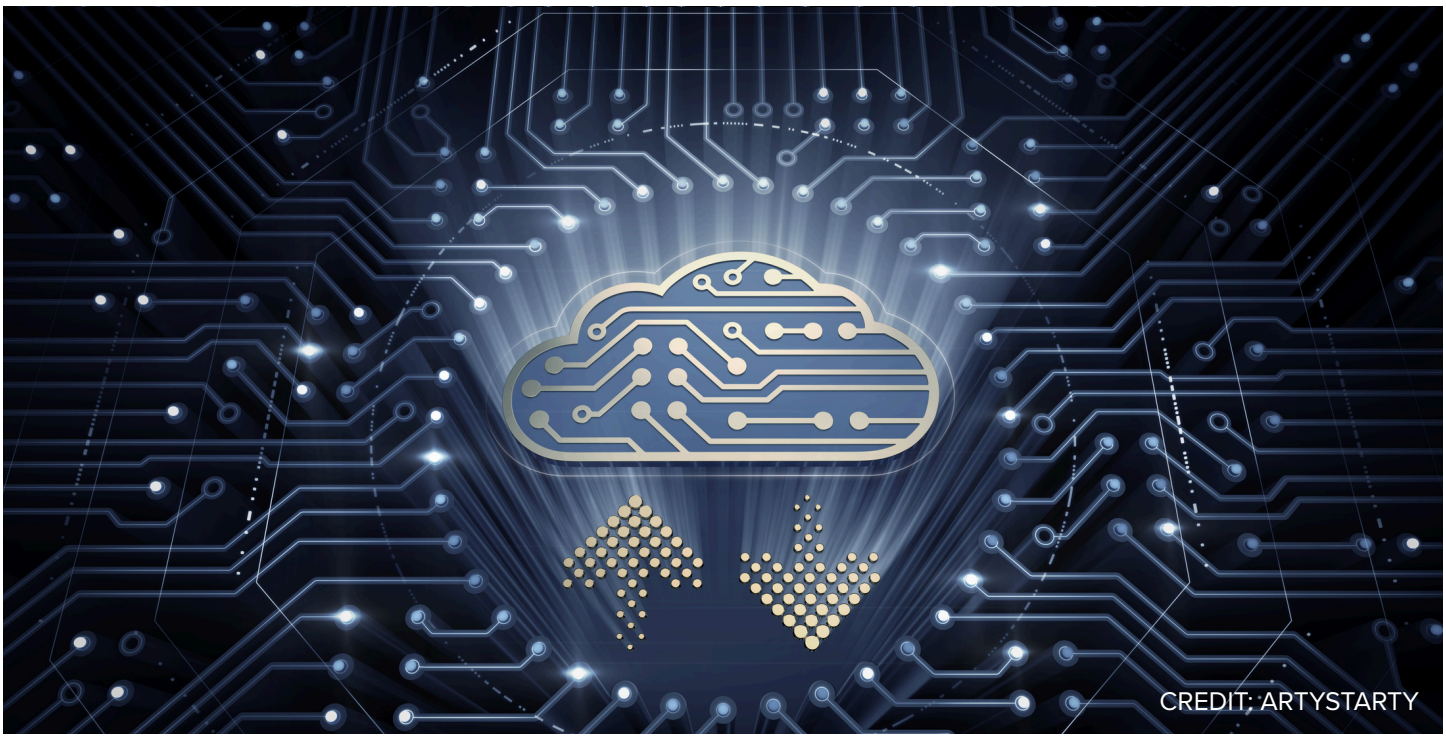
BENCHMARK

Costs and Benefits of .NET Application Migration to the Cloud v1.1

An Updated Guide to Cloud-Based Modernization of .NET Applications

NED BELLAVANCE | NOV 3, 2020 - 6:55 AM CST

TOPIC: CLOUD INFRASTRUCTURE



SPONSORED BY Microsoft

GIGAOM

Costs and Benefits of .NET Application Migration to the Cloud

An Updated Guide to Cloud-Based Modernization of .NET Applications

TABLE OF CONTENTS

1. Summary

It's hard to overstate the massive impact that cloud computing and services have had across industries of every type over the past several years. New and updated capabilities are emerging constantly, enabling businesses to augment and transform their existing applications to increase customer engagement, streamline operations, and lower costs. It's the era of digital transformation, and companies must innovate and accelerate to compete in this dynamic environment. If only it were that simple.

All that breakneck innovation poses a serious challenge, as organizations must balance doing the next new thing against their hefty investments in existing systems. Technology strategists face a complex series of trade-offs around a simple question: Do you build new applications, or modernize existing systems and migrate them to more scalable, cloud-based platforms? To answer that, you need to understand the return on investment that comes from modernizing and migrating applications to the cloud.

In this report, we consider this question head-on, assessing the benefits and costs of application migration. Moving from theory to practice, we ran a field test to walk through a migration scenario and evaluate the costs, performance, and benefits of migration. We learned that:

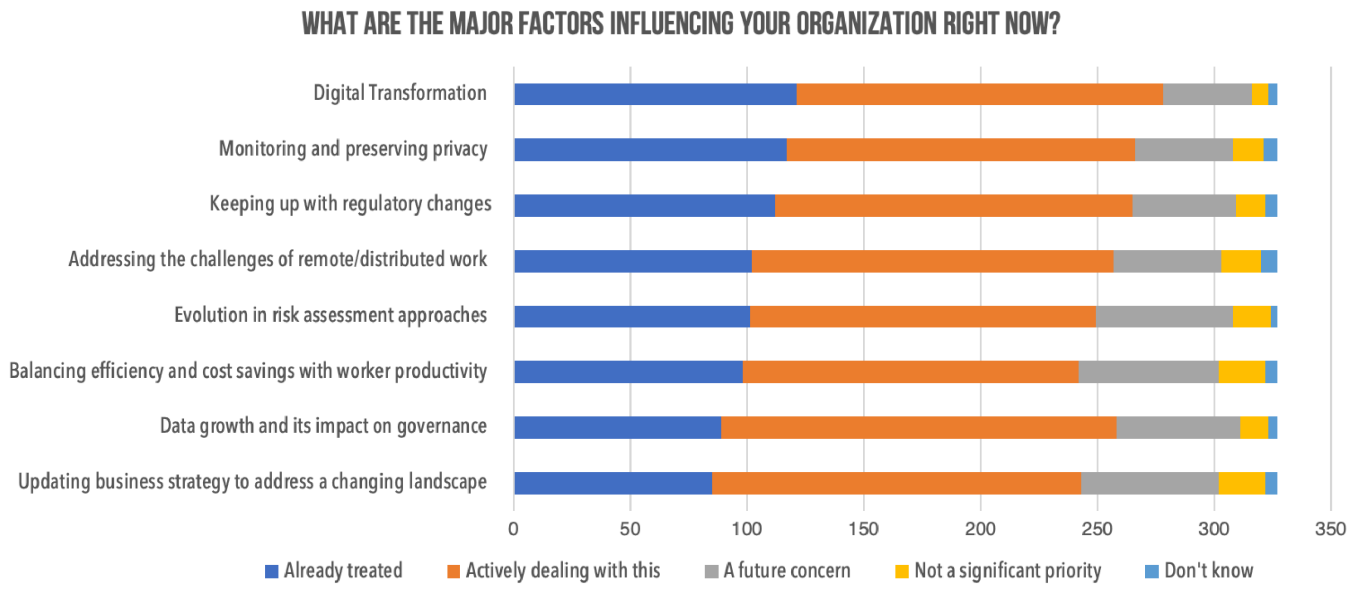
- Significant cost benefits can be gained from application migration to the cloud from on-premises infrastructure.
- Traditional .NET applications can benefit from such a migration without refactoring underlying code.
- The Microsoft Azure solution offered potential total cost of ownership (TCO) savings of up to 54% over running on-premises and 30% over running on AWS.
- Streamlined operations, simplified administration, and proximity to advanced cloud services are additional benefits.
- It is critical to assess traditional applications for migration to the cloud and balance the requirements of the application with the potential advantages of the cloud.

Looking specifically at .NET applications, our field test study enabled us to assess different approaches for migration and weigh the pros and cons of each. We determined that moving to Microsoft Azure has a measurable TCO advantage over Amazon Web Services, and that Azure offers additional benefits beyond TCO.

Overall, we learned that a cloud migration of any form frees up considerable time and resources to work on tasks that drive value instead of just keeping the lights on. As a result, we can say that modernization may be viewed as an imperative element of a forward-thinking application strategy. The best starting point is to evaluate existing applications and determine the best migration strategy for each, based on priority.

2. What's Driving Application Modernization?

Digital transformation – the adoption of leading-edge technologies to drive business value and customer success – is seen by most organizations as a critical element of success, a reality illuminated by Covid-19 and its consequences. Recent GigaOm research conducted on behalf of Microsoft shows that while 37% of enterprise organizations have already treated their digital transformation needs, an additional 48% are still addressing them. (**Figure 1**)



Source: GigaOm 2020

Figure 1. Digital Transformation as a Driver of Success

This need to use technology as a value driver pervades every aspect of business today, but no large organization is operating a greenfield operation with carte blanche to innovate and accelerate. They need to work with, through, and around existing systems and applications. These so-called called legacy or heritage applications live in on-premises data centers and are the workhorse of most organizations, responsible for a large portion of their revenue generation.

And there is the rub. Legacy systems are a core asset that drive both revenue and value and ensure the smooth operation of the business. While maintaining these systems and infrastructure can preserve value, it generally won't add to it, and that's a problem for enterprises looking to transform. These legacy systems are a source of friction and bottlenecks. For example, enterprise organizations today have many thousands of traditional ASP.NET applications running on-premises, delivering value to their owners, but consuming valuable resources, instead of driving innovation.

Rationalization has long been the dream in such environments, but the reality is that rewriting applications from scratch is rarely an option. As businesses assess their portfolio of applications, there

is a question of whether it makes sense to modernize, rather than replace, an application. Clearly there needs to be some tangible benefit, or the effort to transform the application would be wasted. We consider options, benefits and costs in the section below.

3. Options and Benefits for Application Modernization

Among the options for application modernization, there is a spectrum running from a full rewrite of the application to a simple lift-and-shift of the application to a new platform. Terminology may vary, but we can focus on the following options, depicted in **Figure 2**:

- **Rehost** – Virtual Machines running on on-premises servers are "lift-and-shifted" to cloud-based servers.
- **Replatform** – Application logic (for example, written in ASP.NET) is migrated from an on-premises platform to cloud-based PaaS (Platform as a Service).
- **Refactor** – Existing code is reviewed and restructured to take better advantage of cloud-based models and services.
- **Rewrite** – An on-premises application is replaced by a cloud-native version that delivers similar, if not enhanced functionality.

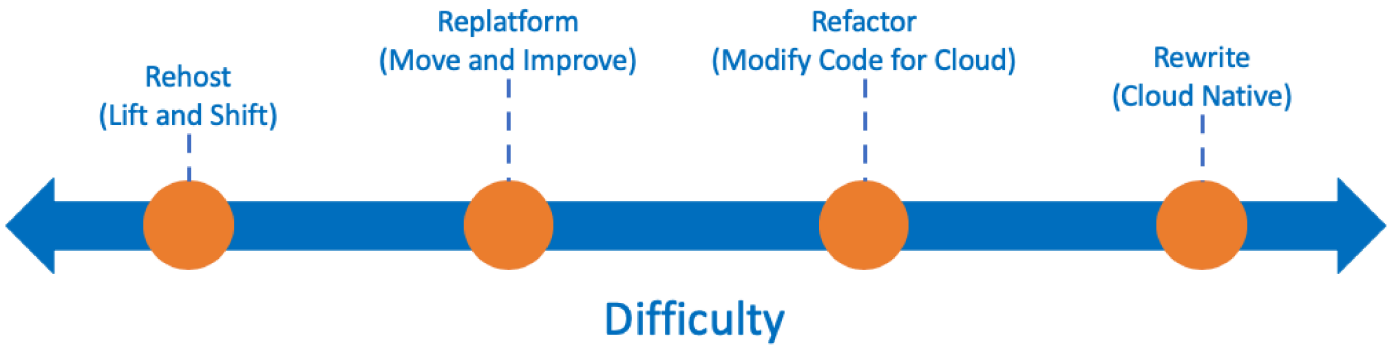


Figure 2. Methods of Application Modernization

Completely rewriting or refactoring a traditional application to be cloud native is a complex and time-consuming process. While there may be long-term benefits to undertaking such a project, in the short term, costs will be high and productivity may suffer. Many organizations will choose to forgo a complete rewrite of a traditional application in favor of simply migrating the virtual machines hosting the application to the cloud. These migrations are the simplest and often the fastest, but they cause the organization to miss out on many of the benefits of moving to the public cloud.

Table 1. Benefits and Challenges of Modernization Approaches

	BENEFITS	CHALLENGES
Rehost	Simple and rapid	Still requires infrastructure maintenance
Replatform	Simple, managed infrastructure	Possible minor code changes
Refactor	Cloud-focused, deep PaaS integration	Major code changes and rearchitecting of application components
Rewrite	Loosely coupled and independently scalable	Complex and time consuming

Source: GigaOm 2020

As we can see in **Table 1**, replatforming an application, aka "move-and-improve," represents a happy medium point between the two extremes of rewriting and lift-and-shift. In a replatforming migration, the application components are moved to a Platform-as-a-Service offering like Microsoft Azure App Service or AWS Elastic Beanstalk. Companies get the benefit of having the cloud provider manage the underlying infrastructure and software of the platform, while only needing to make minor changes to their applications.

Alongside potential cost reductions, benefits of replatforming include:

- Increased productivity
- Enhanced features
- Access to innovation

All of these benefits respond to the need for greater agility in support of digital transformation. We cover each of them below, with illustrations from the Microsoft Azure portfolio.

Increased Productivity

Migrating an application from an on-premises deployment to a fully managed service in the public cloud can increase productivity in the following distinct ways:

- **Reduce administrative overhead:** Adopting a managed platform helps optimize costs and lowers administrative overhead associated with the application. The operations team no longer needs to patch operating systems, manually scale for capacity, or manage underlying database engines.
- **Improve operational efficiency:** Reduced overhead means the operations team is freed up to focus on other aspects of the system, working to improve operational efficiency and reduce downtime. This moves operations onto the front foot, enabling teams to operate more confidently.
- **Speed application and feature delivery:** Rather than just keeping the lights on, the ultimate goal of operations is to act as an enabler of innovation, empowering development teams to deliver new applications and features faster by reducing deployment times and enabling access to leading-edge services.

Providers like Microsoft Azure provide rich monitoring, reporting, and analytical tools that will assist the operations team in diagnosing potential issues, finding the root cause, and verifying that a solution was effective. Both Azure App Service and Azure SQL Database are able to scale dynamically and automatically to meet customer needs and application demand. Using these capabilities, operations teams no longer have to spend time manually spinning up new instances or vertically scaling their database server to keep pace with demand. They also do not need to plan for capacity in their data centers to accommodate peak load and growth. The cloud can scale to meet bursts of activity, and then contract when demand subsides.

Cloud providers also offer the ability to automate the provisioning of additional operational environments for development, quality assurance, staging, and more. The operations team no longer needs to spend weeks creating an additional environment, and they can use infrastructure as code to ensure the various operational environments stay consistent.

Enhanced Features

Beyond operational efficiency, cloud providers offer many features that enhance both the operational and development experience. For example, the Azure App Service platform enables developers to take advantage of deployment slots on a web application. Each deployment slot runs its own version of the application, with an independent namespace. Traffic can be distributed across multiple slots for canary deployments, blue/green testing, or a regular cutover between staging and production. Because there is also deep integration between Visual Studio and Azure App Service, additional capabilities, such as live-site debugging, are unlocked for the developer.

Azure App Service also includes integration with GitHub, Azure Repos, and BitBucket to enable continuous deployment. New commits to a specific branch in a code repository will trigger a build process and deployment of the new code to Azure App Service. The continuous deployment feature can be integrated with Azure Pipelines to ensure that proper code coverage and testing is performed before a new build is deployed.

Backup and recovery of application data and replication to a separate region is simplified when using

the cloud. Azure SQL Database can easily replicate data to a designed partner region. A scaled down instance of Azure App Service could be running in the same partner region, with the whole solution placed behind a Traffic Manager to make failover a relatively simple process. Since capacity can be allocated on demand, it is no longer necessary to maintain a disaster recovery environment with matching capacity at all times.

Access to Innovation

Migrating on-premises applications places them directly adjacent to the rapid innovation happening in the cloud today. New services and features are constantly being introduced in clouds like Microsoft Azure. In some cases, portions of the application could be updated to use a different service, like moving from traditional Microsoft SQL Server to Azure SQL Database serverless to handle frequent scaling and unpredictable use. In other cases, an application could be enhanced by consuming another service, like adding a chat bot to an existing application using Azure Bot Service, or adding voice commands with Azure Cognitive Services.

By moving an application to the cloud, it will have direct access to all of these services and more. Additionally, developers and operations managers are freed up to investigate and experiment with these services rather than simply keeping the lights on.

4. Comparing PaaS Offerings For Migration

While newer applications might be developed in a cloud native manner, traditional ASP.NET applications were written prior to advances like .NET Core and microservice architectures (**Figure 3**). It is not simple or cost effective to rewrite these applications to support a new programming paradigm, but that does not mean they cannot take advantage of what the cloud has to offer.

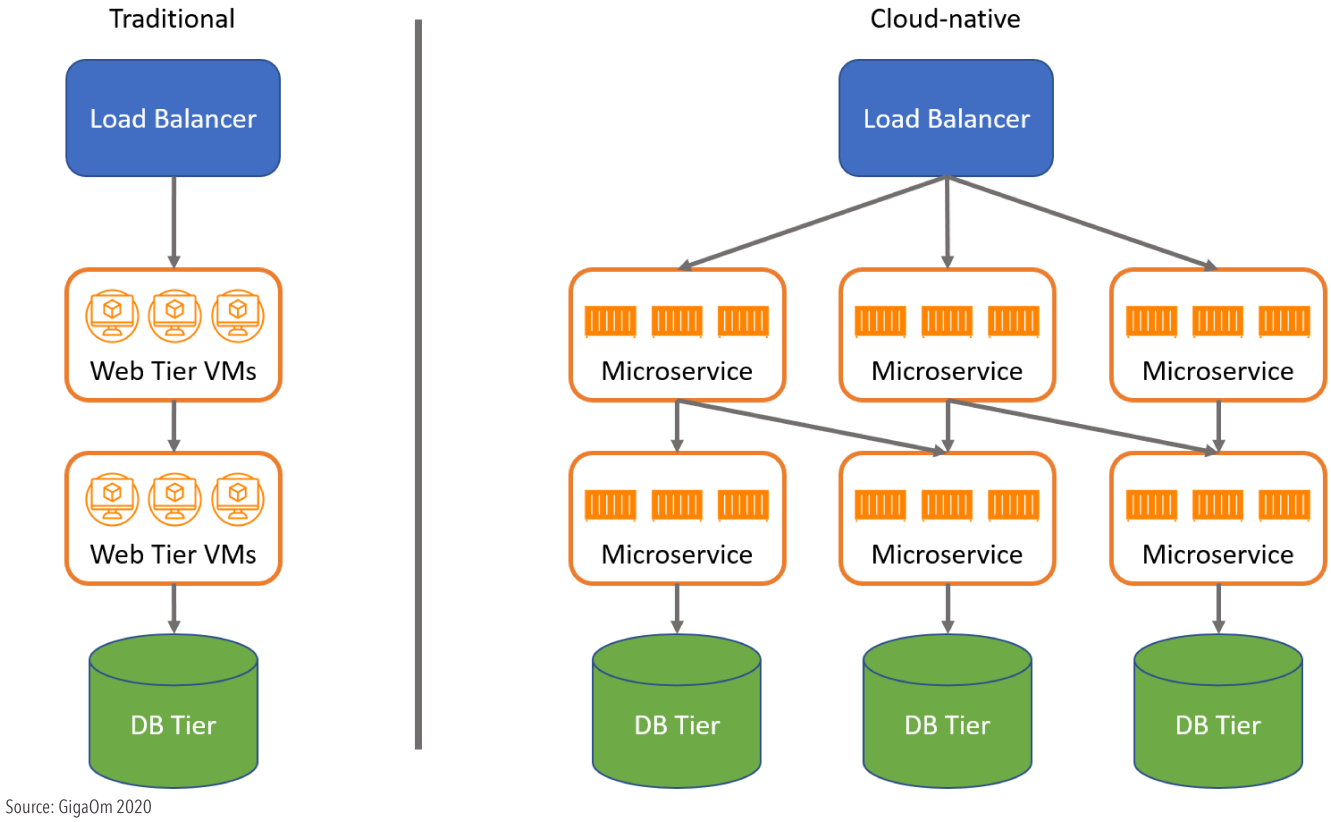


Figure 3. Traditional and Cloud-Native Architectures

Traditional ASP.NET applications backed by Microsoft SQL Server can be moved to the Platform as a Service (PaaS) offerings on the major public cloud providers. Microsoft offers Azure App Service and Azure SQL Database to provide the web front-end and database backend required by most applications. The underlying systems are managed by Microsoft, reducing the operational overhead associated with managing and maintaining web and database servers.

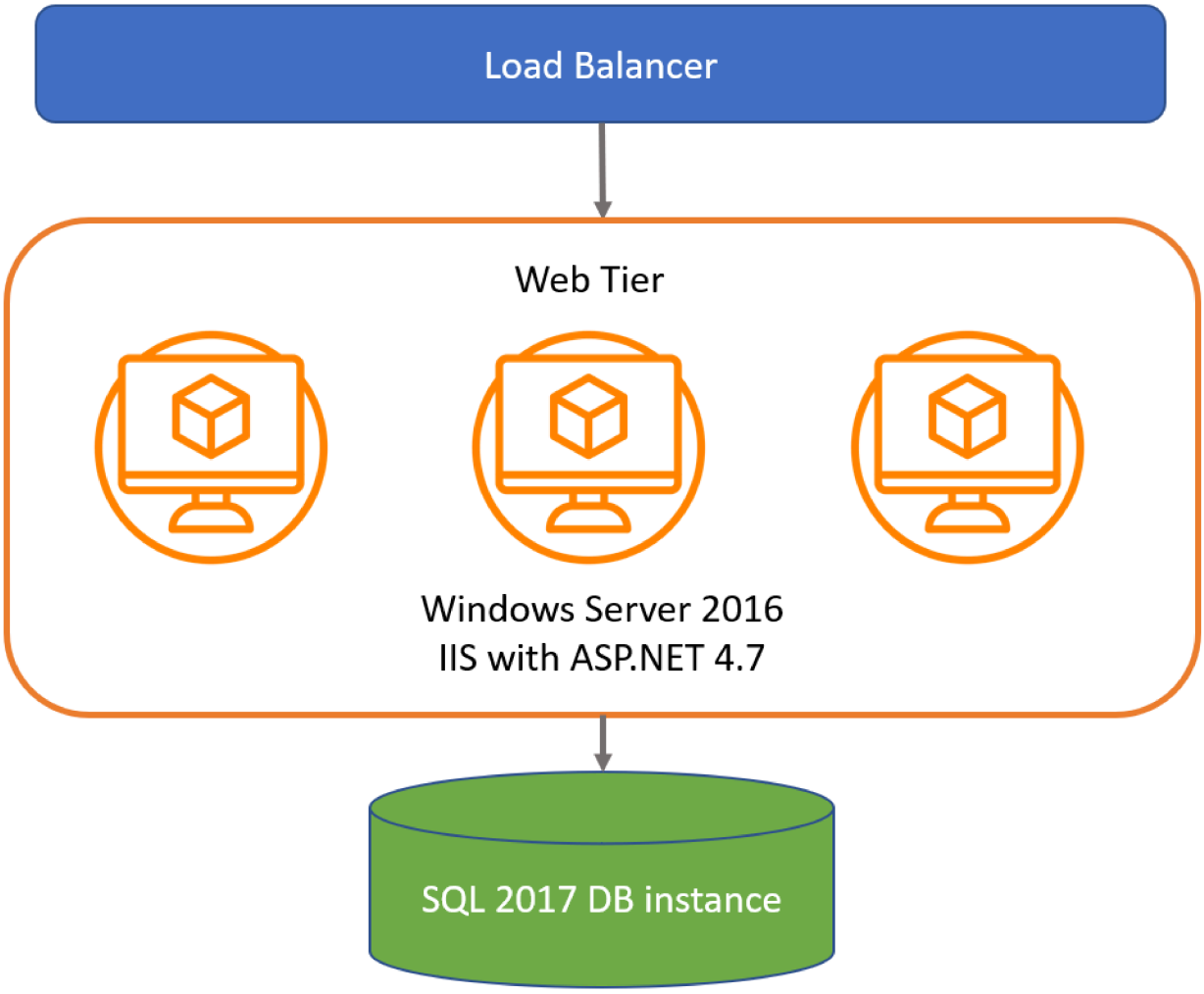
Amazon Web Services has some similar capabilities in its Elastic Beanstalk service, which is capable of generating environments that include an autoscaling group of EC2 instances and a deployment of their Relational Database Service. Both AWS and Microsoft are providing managed virtual machines running Windows Server and IIS, mirroring the current deployment environment of traditional ASP.NET applications. Microsoft's PaaS solution abstracts most of the underlying infrastructure and maintenance details while AWS' offering exposes these to the customer.

The similarities of these platforms with the current deployment of ASP.NET applications on-premises makes the migration process simple and predictable, while providing benefits over a lift-and-shift migration that simply uses Infrastructure as a Service (IaaS) components. Essentially, without refactoring or rewriting code, traditional applications can be moved seamlessly to a PaaS deployment in the cloud. At least, that is the theory. To test that theory, GigaOm chose to do a field test migration of an ASP.NET application from a traditional on-premises environment to both Microsoft Azure and AWS.

In order to understand the migration process and gain insight into available solution options, GigaOm performed a field test based on the scenario of migrating a web-based application from an on-premises environment to the cloud. The field test compared three environments:

1. Windows VMs running on VMware
2. Microsoft Azure using Azure App Service and Azure SQL Database
3. AWS using Elastic Beanstalk, EC2, and Amazon RDS

The application deployed was the Parts Unlimited web store, an ASP.NET application using IIS for a web server and Microsoft SQL Server for the backend, as shown in **Figure 4**. The web store has an inventory of items that can be searched and purchased by registered users. The application was installed in each environment and assessed for performance, ease of migration, and additional features available from each cloud provider.



Source: GigaOm 2020

Figure 4. The Parts Unlimited Application Architecture

We look at the top-level results of the field test, and their ramifications, below. Further detail on the field test is provided in the appendix.

5. Field Test Findings, Migration Benefits and TCO Analysis

As part of this study, we were able to compare and contrast two of the leading solutions on the market – Amazon AWS and Microsoft Azure - with the option of leaving the application in place. Our top-level findings across ease of migration, performance, and TCO were as follows:

The migration process from on-premises to both Azure and AWS was relatively straightforward, with Azure leading the way. In both cases, the SQL database was backed up and restored to the target migration environment, then the application code was deployed using Visual Studio. The process of code deployment was simpler on Microsoft Azure than AWS due to the tightly integrated nature of Visual Studio and Azure services. Installing the AWS Toolkit for Visual Studio made the process simpler, but it still was not as seamless as the Azure experience.

One particular aspect was the configuration of the application front end to connect to the database. Visual Studio was able to leverage the built-in functionality of the App Service configuration properties to update the database connection string. The AWS Elastic Beanstalk deployment required the definition of a Web.config transform file to handle the database connection. This is one of a few instances where it was clear that Azure App Service is more of a guided experience than AWS Elastic Beanstalk.

The performance of resulting application deployments was roughly the same. This result was expected because care was taken to select similar virtual machine sizes, hardware generations, and software. More information on the specifics of the hardware can be found in the appendix.

The intention of the test was to show that performance of a migrated application would be equivalent to or better than the on-premises instance. Once the application has been migrated, there are many updates that could be employed to improve performance, including the addition of a content-delivery network (CDN), dynamic scaling of resources, upgrading to HTTP/2, and adding a local cache to the web servers.

The Microsoft Azure solution offered savings of up to 54% compared to on-premises and 30% compared to AWS – There is significant savings to be had by migrating the application to Microsoft Azure. Our benchmark showed that on-premises was the most expensive option. Running in AWS was more expensive than Microsoft Azure as well and did not allow the reuse of existing Microsoft licensing. Potentially, additional savings could be had by introducing dynamic scaling to the environment to reduce consumption during periods of lower demand. More detail on the TCO calculation and price performance considerations can be found below.

TCO Analysis

Calculating a true TCO for application modernization or migration is difficult because of variable qualitative aspects, such as administrative overhead and operational headcount. For the analysis of each environment, we therefore assumed that the headcount remains consistent, and we did not take

into consideration the larger data center or cloud deployment that might be part of an organization considering the move. Instead we looked at the cost of running each environment for three years.

Hosting considerations also need to be taken into account. Migrating the entire application from an on-premises environment would likely result in the removal of one virtualization host. Therefore, for the on-premises environment, we included the cost of two physical hosts, four vSphere licenses for the CPUs, and an estimate for the power and cooling consumed by those hosts.

In terms of results, Microsoft Azure with Azure Hybrid Benefit (AHB) licensing had the lowest overall cost, saving an estimated 54% over the on-premises deployment and 30% over AWS.

AHB licensing allows existing Windows Server and SQL Server licenses to be applied to Azure virtual machines and Azure SQL Database instances. The estimated cost of a SQL Standard license is $10,800, based on official pricing published in June 2020. Assuming a migration scenario with an existing SQL Standard License being recouped for use in Microsoft Azure, the Azure SQL Database cost would be reduced from $18,000 to $7,200, which would reduce the total Microsoft Azure cost to $31,088. The use of existing SQL licenses is not an option on Amazon RDS. The addition of AHB creates a compelling savings of 30% over AWS.

These relative costs are shown in **Table 2**. Note that we do not incorporate dynamic scalability and bundled features, which would also have a bearing. In addition, the pricing calculation for Azure uses reserved capacity pricing for App Service and Azure SQL and the US East region for hosting, while AWS uses reserved instance pricing and the us-east-1 region for hosting. More information regarding the configuration and calculation can be found in the appendix.

Table 2. Estimated Cost by Environment

ENVIRONMENT	ESTIMATED COSTS
Microsoft Azure (AHB)	$31,088
Amazon Web Services	$44,237
On-Premises	$69,300

Source: GigaOm 2020

There are other, less quantitative aspects that we believe give Microsoft Azure an edge when it comes to calculating a true TCO figure. One important aspect is operational efficiency and administrative burden. Each approach offers a different combination of benefits and costs:

- **Azure App Service and Azure SQL Database** are designed to be true, managed platforms, abstracting the underlying components that compose each service. To a certain extent, this limits the amount of customization and control that a customer has over the web service or database. The tradeoff is that upgrades, patching, and maintenance are largely handled by Microsoft.
- **Amazon Elastic Beanstalk** acts more like an automation platform to provision an environment than a fully managed offering. The base constructs used by Elastic Beanstalk - EC2, RDS, ALB - are not abstracted from the customer, providing a large range of flexibility. The tradeoff for AWS customers is a potential for greater administrative load that is hardly any different from consuming Infrastructure as a Service (IaaS). In fact, a more apt comparison might be Azure Virtual Machine Scale Sets, Application Gateway, and Azure SQL Managed Instances.
- **On-premises deployment** does not offer any form of managed services. It is up to the existing operations teams to continue to support the application. Any migration to the cloud should be considered in terms of the additional administrative load and complexity introduced by a new environment, compared to the lower cost and operational efficiency that come from using a managed service.

Finally, we note that some applications may require the additional flexibility provided by an IaaS type solution. The Parts Unlimited application used in our benchmarking test was not one of those, and based on Microsoft's intense focus supporting traditional ASP.NET, most applications should work on Azure App Service.

Price/Performance Considerations

During the benchmarking process, we ran an average of 15,000 sessions per hour through each environment, or about 11 million sessions per month. None of the web servers exceeded 25% utilization, meaning there was ample headroom. By reducing the number of web servers to two with an autoscale policy to add instances when the CPU reaches 70%, we can further reduce the cost of the environment without sacrificing performance.

Table 3 illustrates the cost per month for each environment and the cost per 10K sessions, including where the web servers have been scaled back to two instead of four. **Figure 5** compares the cost per 10K sessions at the 4 VM and 2 VM levels.

Table 3. Cost per Month for Each Environment (lower is better)

	COST 4 VM	COST 2 VM	PER 10K SESSION 4 VM	PER 10K SESSION 2 VM
Microsoft Azure (AHB)	$864	$582	$0.79	$0.53
Amazon Web Services	$1229	$1013	$1.12	$0.92
On-Premises	$1925	$1897	$1.75	$1.73

Source: GigaOm 2020

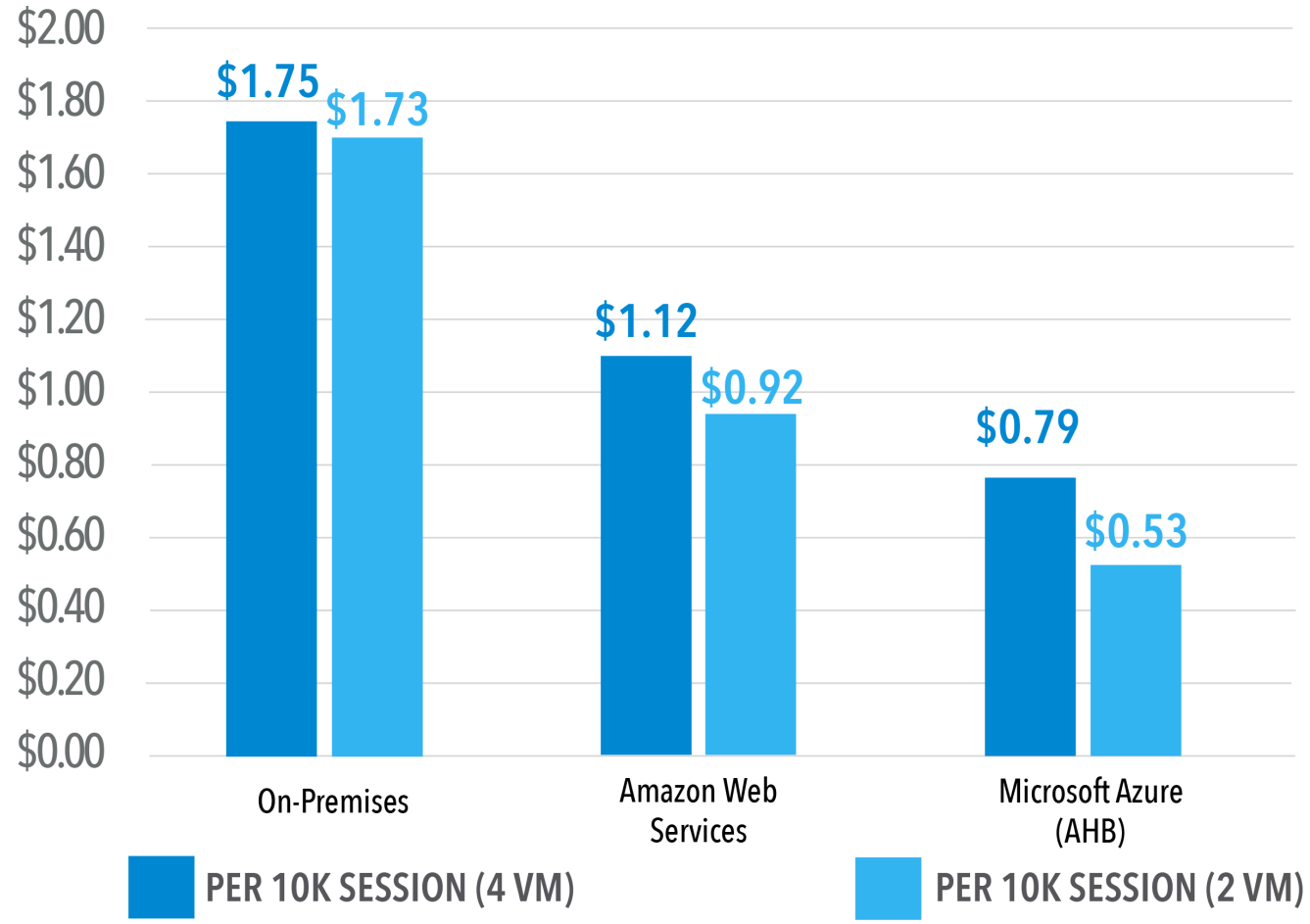


Figure 5. Cost per 10,000 Sessions for Each Environment (lower is better)

Whether the application is running on-premises or on Microsoft Azure, the same SQL Standard licenses would be necessary. If the application is moved to AWS, the license cannot be migrated and instead the license cost is rolled into RDS. The Microsoft Azure with AHB environment therefore represents a cost savings of 30% over AWS at 4 VMs and a savings of 42% at 2 VMs.

6. Conclusion

Organizations looking to transform can see application migration as a clear opportunity to reduce on-premises legacy workloads, and benefit from the advantages of cloud-based models. As illustrated by the field test for .NET applications, application migration to a hosted platform environment can be straightforward and offers significant benefits over on-premises solutions, both in tangible (cost/performance) and less tangible (access to innovation) terms.

We would recommend that organizations looking to modernize applications in general, and .NET applications in particular, consider the following:

- Existing legacy and heritage applications can be run at a significantly lower cost using a cloud-based platform, freeing resources and unlocking the potential for new application delivery.
- Replatforming applications is minimally disruptive and can be simple to test, enabling migration to be piloted, risks to be assessed, and costs to be evaluated prior to full deployment.
- Fully managed platforms provide the highest level of simplicity and lowest administrative burden compared to in-house infrastructure and platform stacks.
- For .NET applications, it is easy to get started today by taking advantage of Microsoft's App Service and Database Migration Assistance tools, which help automate migration to the Azure App Service and Azure SQL Database.

To get started down a path to migration, we recommend taking the following actions:

- **Assessment:** Categorize existing applications by revenue generation and cost to maintain. While there may be hundreds or thousands of applications to treat, consider working on applications according to a specific group — for example by department, application, or data type.
- **Prioritization:** Identify applications that make good migration candidates, according to criteria such as existing cost of maintenance, risk of migration, and opportunity created — for example to unlock new functionality.
- **Evaluation:** You can map application dependencies across data sources, services, and infrastructure to help identify group migration candidates.
- **Validation:** Perform test migrations to validate functionality and performance, and to build experience around target environments, for example, in terms of what operational management dashboards are available.

Overall, the opportunity to move applications to the cloud does not have to be a seen as an all-or-nothing deal that requires either moving existing virtual machines or partially or completely rewriting applications. By considering replatforming as an approach, organizations can derive benefits rapidly without cost and risk, and unlock new opportunities for innovation and transformation.

7. Appendix

Environment Summary

The on-premises environment had four web servers running Windows Server 2016 and a single database server running Windows Server 2016 and SQL Server 2017. The load balancer was a single virtual machine running Ubuntu LTS 18.04 with Nginx installed. **Figure 6** shows the architecture.

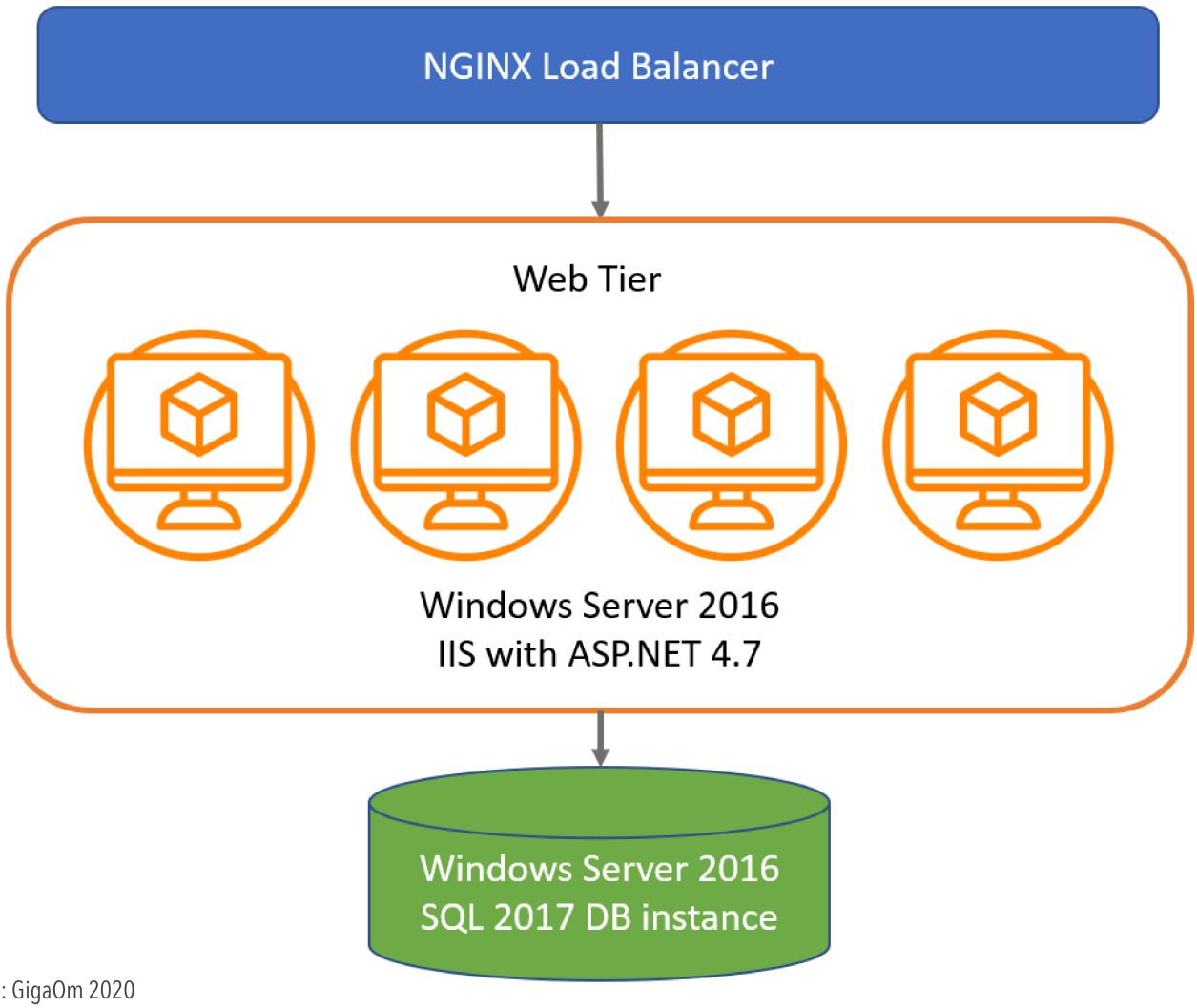
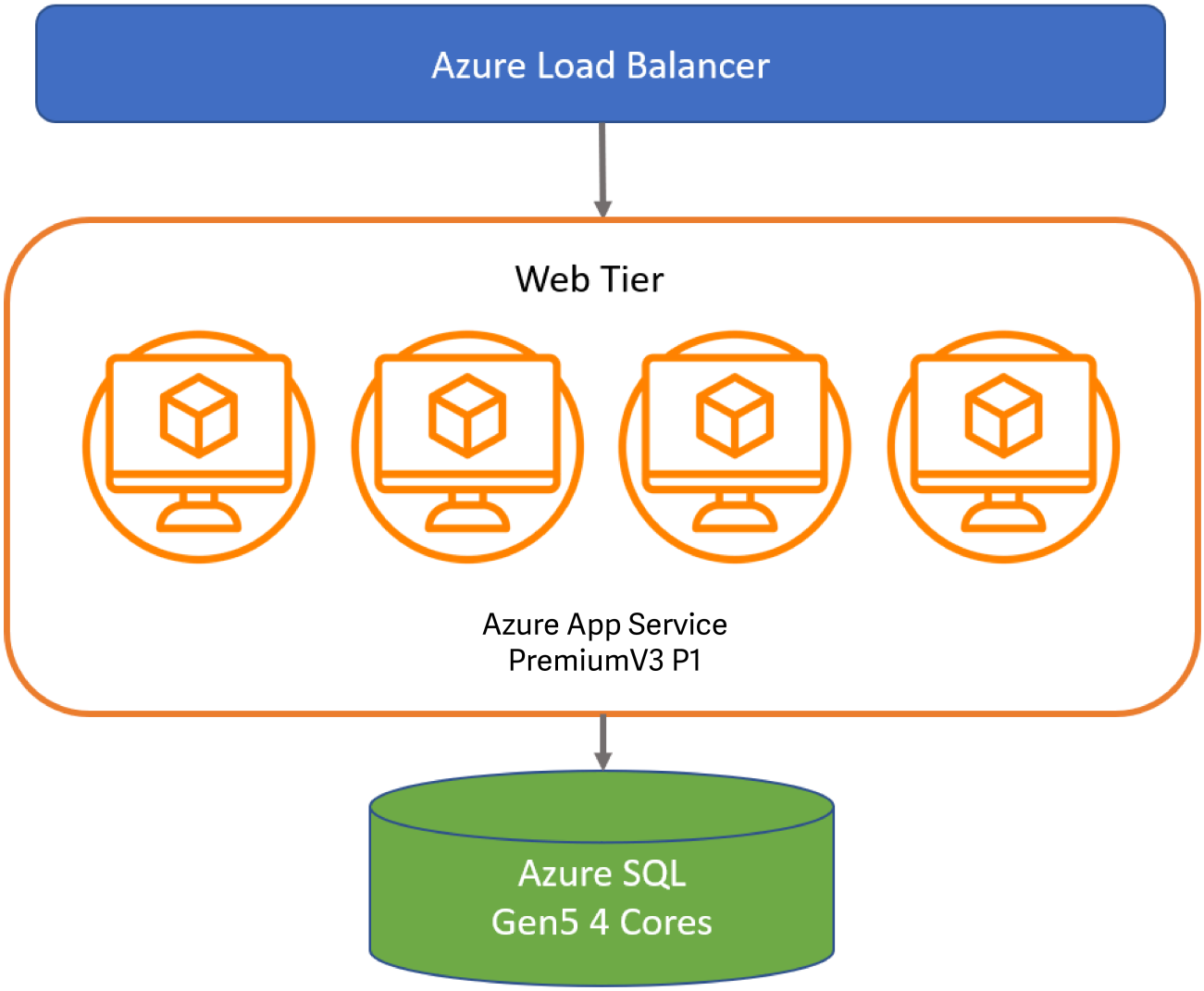


Figure 6. The On-Premises Benchmark Environment

The Microsoft Azure environment shown in **Figure 7** used the Azure App Service with a Premium V3 App Service Plan, configured to run a minimum of four instances. Database services were provided by a Gen5 instance of Azure SQL Database with four vCores. Load balancing services are included as part of the Azure App Service.



Source: GigaOm 2020

Figure 7. The Microsoft Azure Benchmark Environment

The AWS environment shown in **Figure 8** used m5d.large EC2 instances provisioned by Elastic Beanstalk. The instances were part of an autoscale group with a minimum of four instances. Database services were provided by Amazon RDS and provisioned by Elastic Beanstalk. The RDS instance was a db.m5.xlarge size running Microsoft SQL Server 2017. Load balancing services were supported by an Application Load Balancer provisioned by Elastic Beanstalk.

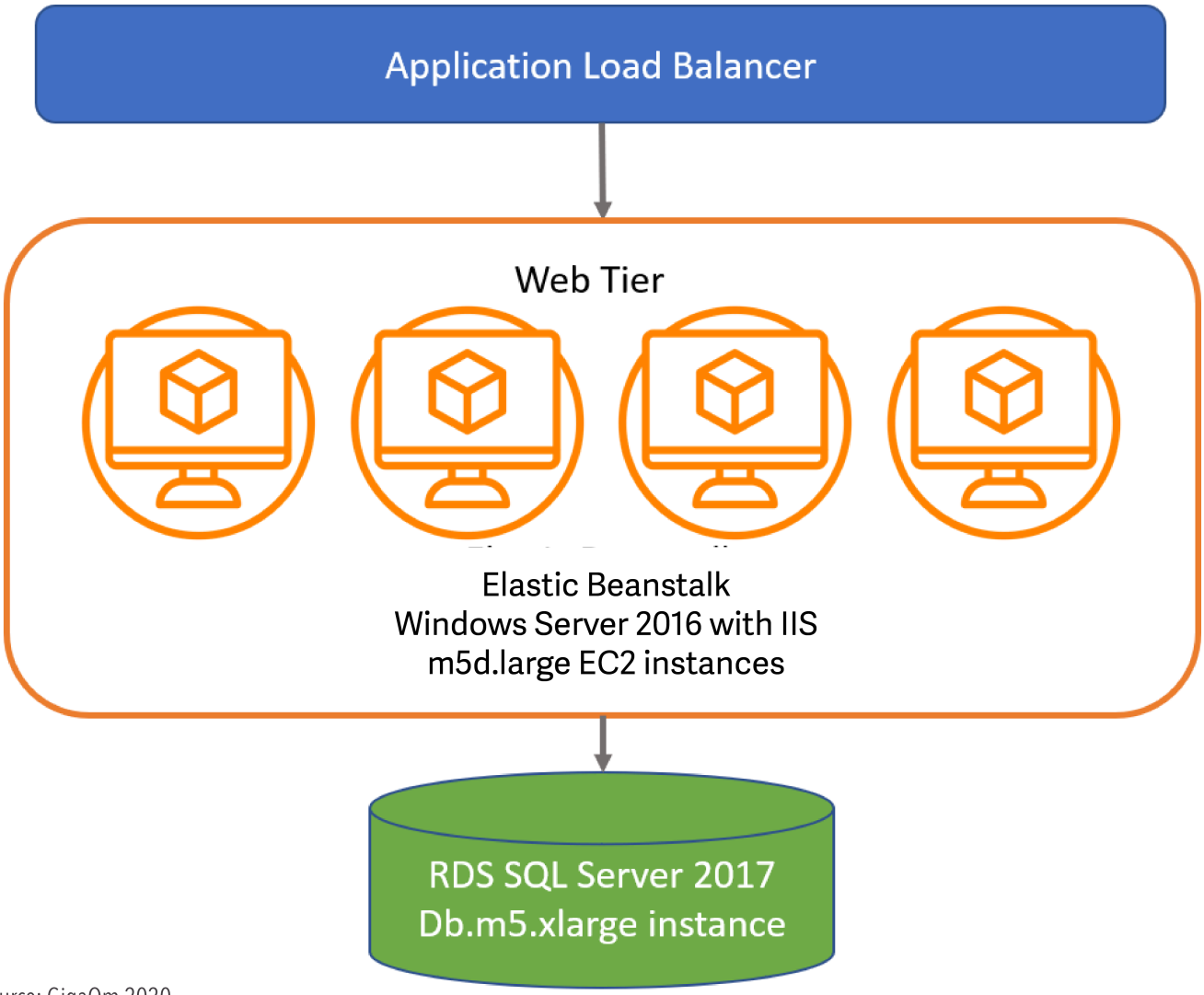


Figure 8. The AWS Benchmark Environment

Testing Configuration

The performance for each instance of the application was assessed by running three different load tests against the application. The *Home Page Load* test simply loaded the home page of the application in a browser window. The *Item Search* test entered a search term on the home page and clicked on the first result. The *Item Purchase* test logged a user into the site and walked through the purchase of an item from the site. Each test was run for 60 minutes, with a sustained load of concurrent users accessing the site. More details on the tests can be found later in this appendix.

The two measures being presented below are the total number of successful sessions executed during each test, and the average duration of each test. The Azure environment led the way in successful sessions completed, especially on the Item Purchase testing, as shown in **Figure 9**.

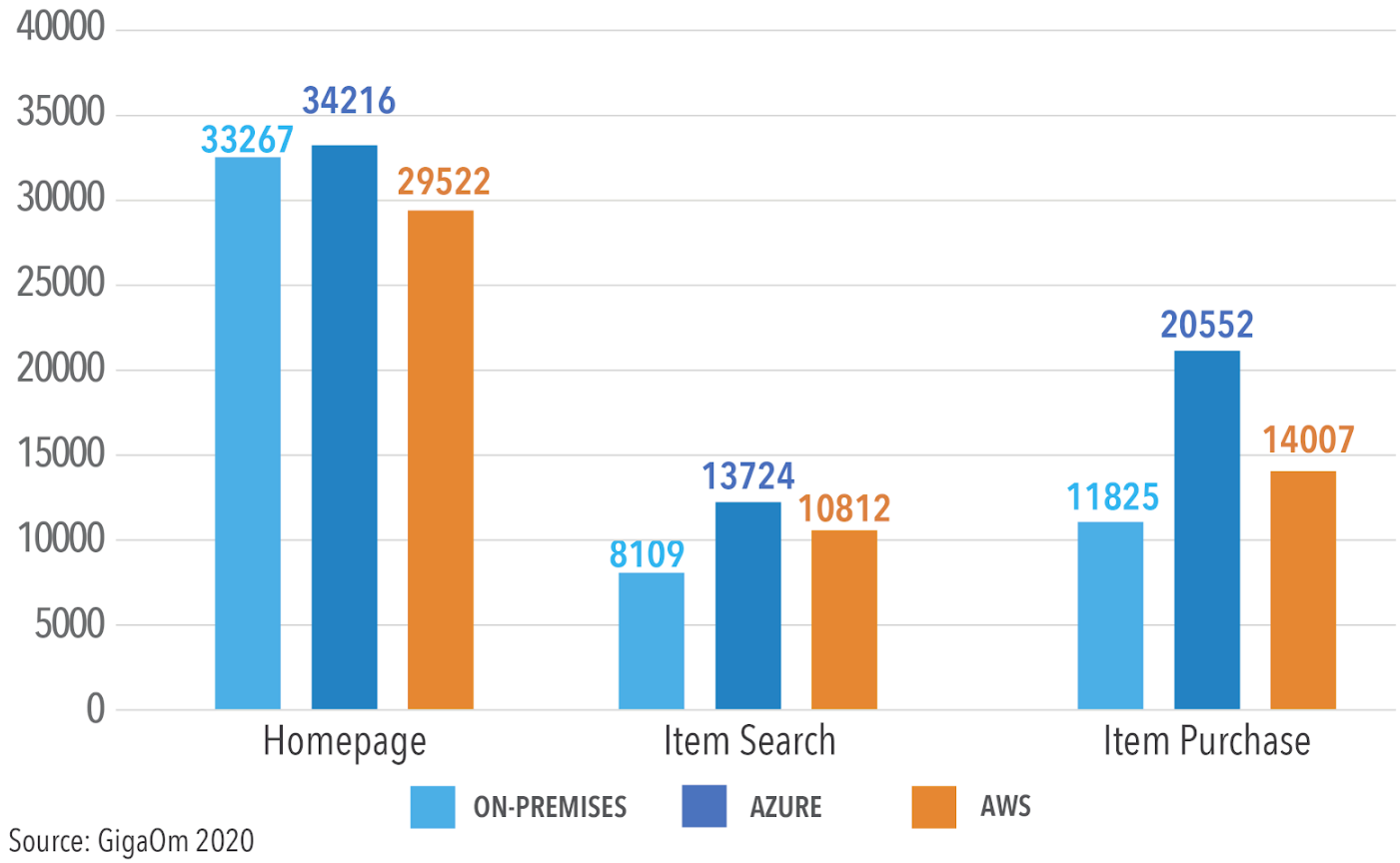
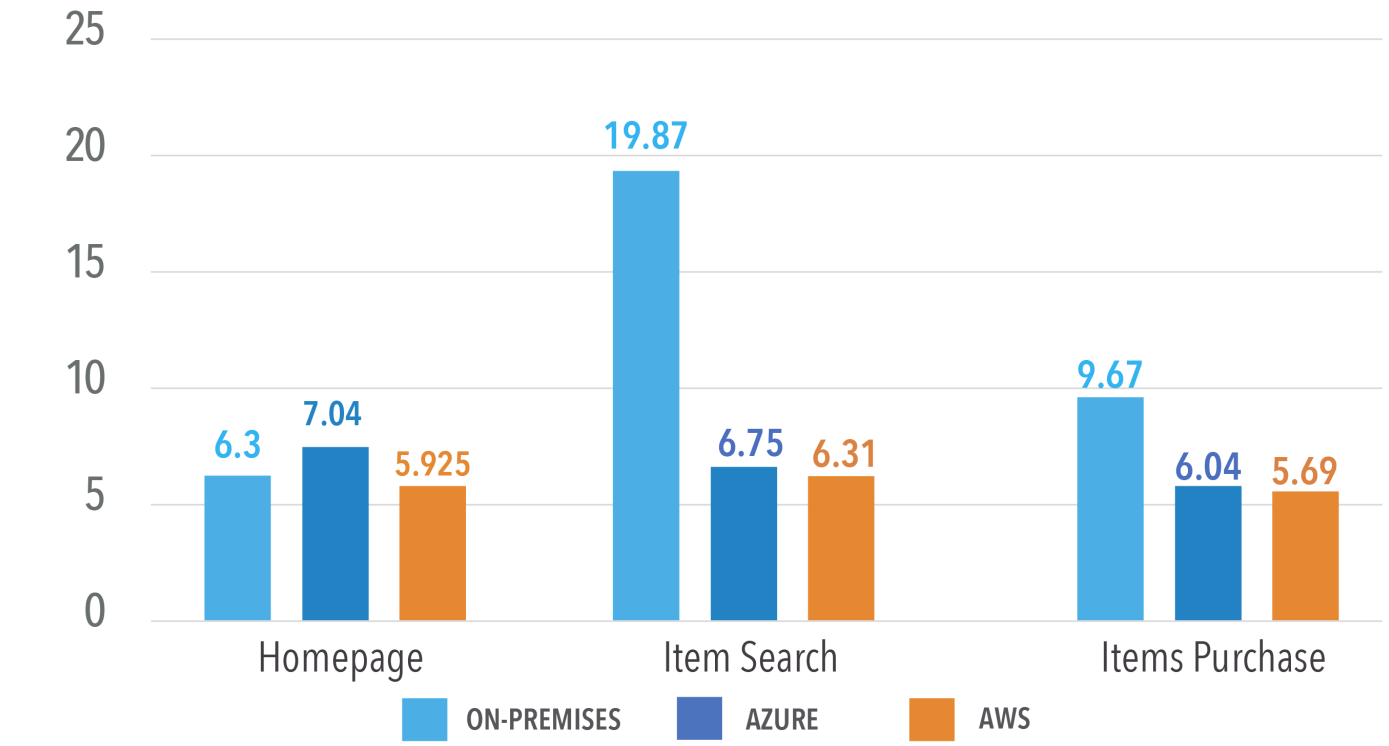


Figure 9. Total Sessions Across Different Load Tests (higher is better)

The AWS environment had the lowest average duration for each test, with Azure demonstrating comparable numbers. **Figure 10** shows the results.



Source: GigaOm 2020

Figure 10. Average Duration Across Load Tests (lower is better)

Overall, the performance of all the environments was remarkably consistent, with both cloud services showing better performance over the on-premises environment. This makes sense because of the similar hardware and software being used for each environment.

Scenario Configuration

The goal of the simulation was to migrate an existing ASP.NET MVC application from a traditional on-premises deployment to Microsoft Azure and Amazon Web Services. Three environments were built out to test the migration process and performance of the application, Microsoft Azure, Amazon Web Services, and VMware vSphere running on bare metal servers supplied by Packet. To keep the comparison valid, we attempted to make each environment as equivalent to the others as possible. In some cases, the same CPU or exact hardware and software specifications were not available in each environment. For instance, Azure SQL Database doesn't use the same exact software as Microsoft SQL Server. Where there is a divergence, we tried to use the closest equivalent.

Application

The application is a version of the Parts Unlimited application, running ASP.NET 4.7 with MVC. The application uses an IIS front-end and a Microsoft SQL Server backend. The version of the Parts Unlimited application can be found on GitHub under the aspnet45 branch.

The application was loaded and configured using Visual Studio 2019 Community Edition. For the Microsoft Azure environment, the application was published directly from Visual Studio using the built-in Azure deployment options. The application was deployed to a pre-existing Azure Web App and Azure SQL Database, instead of creating one through the wizard. Changes to the database connection information were handled by the wizard.

The AWS Toolkit for Visual Studio 2019 extension was installed to simplify deployment of the application to Elastic Beanstalk. The application was published to a pre-existing Elastic Beanstalk environment, instead of creating one through the wizard. Changes to the database connection information were handled through a web transform file.

For the On-premises environment, the application was published to a local folder on the desktop and then copied to the web servers manually. Changes to the database connection information were handled through IIS configuration changes to the Web.config file.

It should be noted that in a true production workflow, publishing a new version of the application would likely be handled by an automated deployment process as part of a CI/CD pipeline backed by a source code repository. For the purposes of this test, it was more important to get the application deployed simply than to build out a full deployment pipeline. We will discuss the various deployment scenarios later on.

On-Premises

To simulate an on-premises environment, we chose to create a VMware installation running on bare-metal servers supplied by Packet. The environment was composed of two bare-metal servers running in Packet's Dallas-Fort Worth region. Note that for CPU configuration that Skylake (2015) is one of four architectures in play, including Cascade Lake (2019), Broadwell (2014), and Haswell (2013). Wikipedia provides an overview of Intel CPU architectures. The specifications for the servers are listed here:

- Size: m2.xlarge.x86
- CPU: 2x Intel Scalable Gold 5120 (Skylake) 28-Core @ 2.2GHz
- RAM: 384GB
- Disk: 3.2 TB NVMe
- Network: 2 X 10Gbps
- OS: ESXi 6.5

On top of the ESXi hosts, virtual machines were created for the web servers, database server, vCenter server, routing VM, and a load balancer. The specifications for each server are listed in **Table 4**:

Table 4. On-Premises Server Specifications

	QUANTITY	O/S	CPU	RAM (GB)	STORAGE (GB)
vCenter	1	N/A	2	10	250
Load Balancer	1	Ubuntu 18.04	2	8	40
Routing	1	Windows Server 2016	2	8	40
Web Server	4	Windows Server 2016	2	8	40
Database Server	1	Windows Server 2016	4	16	90

Source: GigaOm 2020

The VM performed routing between the private address space of the internal virtual machine network and the public IP address space provided by Packet.

The database server had Windows SQL Server 2016 Standard installed.

There are four web servers with IIS and the .NET Framework 4.7 installed. The physical hosts are capable of supporting up to 30 web servers, although it would be a manual process to scale them.

The load balancer VM was using Nginx 1.18 as a load-balancer for the web server. Nginx was configured to use the ip-hash load balancing algorithm. Doing so ensures that each client is directed to the same web server.

Microsoft Azure

The Microsoft Azure environment made use of Azure App Service and Azure SQL Database to host the application running in the East US region. The App Service Plan used for the App Service deployment used the Premium 1 v3 SKU. The virtual machines backing the App Service plan were the D2v4 Azure VM SKU running Windows Server 2016. The minimum number of instances was set to 4, with an autoscale rule to add instances when the average CPU exceeded 70%. There is a maximum of 30 instances for a Web Application.

The D2v4 SKU has 2 vCPU, 8GB of RAM, and 250GB of remote file storage. The CPU used by the Dv4 series of Azure VMs is Intel Xeon Platinum 8272CL (Cascade Lake).

The Azure SQL Database server was created using hardware generation Gen5 with 4 vCores and 20GB of RAM. Gen5 hardware uses an Intel Broadwell or Skylake processor 5.1GB of provisioned memory per vCore.

The Azure App Service does not provide direct access to a load balancing mechanism for apps deployed on the platform. Within the general settings it is possible to control Application Request Routing (ARR) affinity. For the deployment we set ARR affinity to "enabled."

Amazon Web Services

The Amazon Web Services environment made use of the Elastic Beanstalk Service and Relational Database Service (RDS) to host the application running in the us-east-1 (N. Virginia) region. The Elastic Beanstalk environment used the .NET on Windows Server platform with IIS on Windows Server 2016.

The target group behind the platform used m5d.large size EC2 instances with an Intel Xeon Platinum from the Skylake-SP or Cascade Lake generation 2 vCPUs, 8GB of RAM, and 250GB of EBS storage. The group was set to have a minimum of 4 instances, with an autoscale rule to add instances when the average CPU exceeded 70%. The maximum number of instances is a function of the quota for EC2 instances for the AWS region and account. The default limit is 64 vCPUs for M-family instances, which would be the equivalent of 32 m5d.large instances. For parity with the other environments, a limit of 30 instances was set for the target group.

The RDS instance was created using the db.m5.xlarge instance size. The db.m5.xlarge instance uses the Skylake-SP or Cascade Lake processor with 4 vCPUs and 16GB of RAM. The instance was configured with 250GB of EBS SSD storage.

The load balancer for the Elastic Beanstalk Service was an Application Load Balancer (ALB) with session persistence enabled and a round-robin load balancing approach.

Environment Comparison

Table 5. Web Server and Database Components Across Three Environments

	WEB VM				WEB TOTAL				DB VM			
Environment	Size	Cores	RAM (GB)	SSD (GB)	Size	Cores	RAM (GB)	SSD (GB)	Size	Cores	RAM (GB)	SSD (GB)
On-Premises	n/a	2	8	40	4	8	32	160	n/a	4	16	50
Azure	D2S v4	2	8	250	4	8	32	1000	Gen5	4	20	50
AWS	m5d.large	2	8	250	4	8	32	1000	db.m5.xlarge	4	16	200

Source: GigaOm 2020

The largest variation among the environments is storage and CPU. For the web servers, the Azure App Service instances are provisioned with a non-configurable 250 GB of remote storage, while the AWS EC2 m5d.large instances default to 30GB, and VMware suggests 40GB for Windows Server 2016. Additional storage was added to the m5d.large instances for parity. When trying to match hardware across the platforms, the m5 family came the closest in terms of memory and processor.

On the database side, the minimum for the RDS instance size db.m5.xlarge is 200 GB. Azure SQL Database has a configurable amount of storage, as does the VMware implementation.

Since storage is not a constraining factor for the Parts Unlimited application, we made sure all environments had more than adequate storage for the web application and database. The more important factor for storage is having a consistent media type. Azure and AWS are using SSD storage and the on-premises environment is using NVMe. The difference in media type for the on-premises environment may result in slightly better performance, but the net impact should be negligible.

The more important factors are number and type of CPU cores and quantity of RAM, which have been kept nearly identical across the environments (shown in **Table 6**). Likewise, the CPU generations were kept as similar as possible as well.

Table 6. Type of CPU Cores Used in Each Environment

	WEB	DATABASE
On-Premises	Skylake	Skylake
Azure	Cascade Lake	Skylake or Broadwell
AWS	Skylake-SP or Cascade Lake	Skylake or Cascade Lake

Source: GigaOm 2020

By keeping as many factors as possible the same, the expectation is that the performance of the application should be similar across all three environments.

Testing Configuration

Three different tests were performed across the three environments to check on their performance in common usage scenarios. The load testing was performed using LoadView by Dotcom-Monitor. LoadView is a hosted solution capable of generating web page loads from multiple AWS regions

around the world. Each test is composed of multiple load injector nodes, each running one or more instances of the test. LoadView assesses each test to determine the number of test instances supported by a single node.

It's important to note that the tests are running from within AWS and one of the environments being tested is also running in AWS. Specifically, the AWS environment is running in the us-east-1 (N. Virginia) region. The testing environment used a mix of four different AWS regions:

- 40% - Amazon US East (Ohio)
- 40% - Amazon US West (N. California)
- 10% - Amazon EU (London)
- 10% - Amazon Asia Pacific (Tokyo)

The arrangement may have led to slightly better performance numbers for the AWS environment, although the data do not seem to support that.

The tests performed by LoadView can be a simple HTTP request, a complete page load, or a multi-step website interaction. For the performance test, we chose to go with three scenarios:

- **Homepage load:** Access the main page of the application and evaluate how long it takes to load.
- **Item search:** Access the main page of the application and search for an item using the search box. Click on the first result from the search.
- **Item purchase:** Access the main page of the application and log in as a customer. Select an item category and click on the first item in the category. Add the item to the shopping cart and then purchase the item.

These tests are not leveraging an API or REST call. Each test simulates the actions in a browser – Chrome in this case – and measures the time it takes to complete each action, as well as the overall time of the test.

Testing Results

Home Page Load

The *Home Page Load* test was a simple test to load the home page of the application (picture below):

The load type was a step curve performed over one hour, with a gradual increase in the number of users for the first 10 minutes, and then a sustained load for the following 50 minutes. The test started

with 10 users, increased by 10 users for 10 minutes and then held steady at 110 users for 50 minutes.

The test was distributed across four AWS regions with the following allocations and number of load injectors, as shown in **Table 7**:

Table 7. Home Page Load Test Specifications

	USER ALLOCATION	LOAD INJECTORS	TOTAL USERS
US East (Ohio)	40%	7	44
US West (N. California)	40%	6	44
Asia Pacific (Tokyo)	10%	2	11
EU (London)	10%	2	11

Source: GigaOm 2020

Each environment was subjected to the same test with the same parameters. As noted above, the AWS environment is running in the same region as 40% of the configured load. We should expect that response times will be a little lower for those tests due to the lower latency of the network.

Table 8 shows the results of the *Home Page Load* test against all three environments.

Table 8. Home Page Load Test Results

	ON-PREMISES	AZURE	AWS
Success Sessions	33267	34216	29507
Failed Sessions	1084	0	0
Average Duration	6.3069	7.0417	5.925
Max Duration	68.586	17.384	14.488
Std. Deviation	5.0234	1.1544	1.2223

Source: GigaOm 2020

Each environment handled a similar number of sessions, with Azure processing the most sessions. The average duration of the session for each environment was around six seconds, with Azure showing the most consistent performance of the three.

Item Search

The *Item Search* test was meant to simulate a user loading the website and searching for a product. The test loaded the home page, typed 'battery' in the search box, executed the search, and clicked on the first result.

The load type was a step curve performed over one hour, with a gradual increase in the number of users for the first 10 minutes, and then a sustained load for the following 50 minutes. The test started with 5 users, increased by 5 users for 10 minutes and then held steady at 55 users for 50 minutes.

The test was distributed across four AWS regions with the following allocations and number of load injectors, indicated in **Table 9**:

Table 9. Item Search Test Specifications

	USER ALLOCATION	LOAD INJECTORS	TOTAL USERS
US East (Ohio)	40%	4	22
US West (N. California)	40%	3	21
Asia Pacific (Tokyo)	10%	1	6
EU (London)	10%	1	6

Source: GigaOm 2020

Each environment was subjected to the same test with the same parameters. **Table 10** shows the results of the *Item Search* test against all three environments.

Table 10. Item Search Test Results

	ON-PREMISES	AZURE	AWS
Success Sessions	8109	13274	10812
Failed Sessions	6	0	0
Average Duration	19.8659	6.7506	6.3018
Max Duration	148.339	13.553	11.631
Std. Deviation	3.9256	0.8009	1.2755

Source: GigaOm 2020

In this case, the Azure environment was able to process the most sessions. The Azure and AWS environments had roughly the same average duration, with the on-premises duration being the biggest outlier. Once again, the Azure environment had the most consistent performance with the lowest standard deviation.

Item Purchase

The *Item Purchase* test was meant to simulate a user purchasing an item from the website. Ten users were created in each environment to be used for the login and purchase process. During each test, a user went to the home page and logged into the site. Then they clicked on the Lights category and clicked on the first item in the results. They then added the item to their shopping cart and went to checkout. In checkout, they filled out the fields for purchase and completed the purchase.

The load type was a step curve performed over one hour, with a gradual increase in the number of users for the first 10 minutes, and then a sustained load for the following 50 minutes. The test started with 5 users, increased by 5 users for 10 minutes and then held steady at 55 users for 50 minutes.

The test was distributed across four AWS regions with the following allocations and number of load injectors, indicated in **Table 11**:

Table 11. Item Purchase Test Specifications

	USER ALLOCATION	LOAD INJECTORS	TOTAL USERS
US East (Ohio)	40%	4	22
US West (N. California)	40%	3	21
Asia Pacific (Tokyo)	10%	1	6
EU (London)	10%	1	6

Source: GigaOm 2020

Each environment was subjected to the same test with the same parameters. **Table 12** shows the results of the *Item Purchase* test against all three environments.

Table 12. Item Purchase Test Results

	ON-PREMISES	AZURE	AWS
Success Sessions	11825	20546	14007
Failed Sessions	74	6	2
Average Duration	9.6732	6.0413	5.6862
Max Duration	903.277	808.668	811.983
Std. Deviation	56.4916	5.9173	7.2441

Source: GigaOm 2020

In this case, the Azure environment was able to process significantly more sessions, while the AWS environment had a lower average session duration. The on-premises environment appears to produce inconsistent performance with a much higher standard deviation.

Test Summary

Overall the performance of each environment was consistent across all three environments. Azure was able to process the most sessions in each test, while AWS had the lowest average duration. The Azure environment also produced the most consistent performance, with the lowest standard deviation for each test. In each test, the on-premises environment lagged behind.

Although the AWS and Azure environments were configured to autoscale the number of web servers, an autoscaling event never occurred. The four instances were able to handle the load generated by the tests without ever surpassing the 70% CPU utilization mark. In fact, the CPU utilization of all the web servers in every environment never exceeded 40%.

If the tests represent a normal load on the Parts Unlimited application, it would be safe to reduce the number of web server instances to three or possibly two. With the dynamic scaling features of Azure and AWS, it would be trivial to scale out as load increased and scale back when load dropped below a certain threshold.

TCO Details

The cost of each environment was calculated using the AWS TCO calculator, AWS Pricing Calculator, and Azure Pricing Calculator. Reserved capacity was used whenever possible, for the maximum duration available (one or three years). The total cost is an estimate of running the application for three years.

The cost calculations are focused solely on the costs of running the applications, and not on all of the supporting services like monitoring, data protection, and security. Migrating a single application from on-premises to the cloud would not have a significant impact on the cost structure of the remaining supporting services. Additionally, the supporting services in each cloud often come free of charge or bundled as part of the service.

On-Premises Costs

The benchmark testing was performed with four web servers running, yielding the costs in **Table 13**.

Table 13. On-Premises Estimated Costs

ON-PREMISES	ESTIMATED COSTS
Hosts Hardware (2x hosts)	$20,000
Hosts SnS (15% per year)	$9,000
Power	$4,000
VMware Licenses (4x CPU)	$10,500
VMware SnS (30% per year)	$10,500
Windows License (5x)	$4,500
Windows SQL License	$10,800
Total Cost	**$69,300**

Source: GigaOm 2020

Microsoft Azure Costs

The benchmark testing was performed with App Service running the PremiumV3 SKU, at the P1 size, with four instances and Azure SQL Database running the Gen5 server type with 4 vCPUs. The resulting cost breakout is shown in **Table 14**.

Table 14. Microsoft Azure Estimated Costs

AZURE	ESTIMATED COSTS
App Service	$20,288
Azure SQL	$18,000
Support	$3,600
Total Cost	**$41,888**

Source: GigaOm 2020

Note that the Azure SQL Database cost shown here does not reflect the Azure Hybrid Benefit (AHB) licensing options that allow existing SQL licenses to be used on Microsoft Azure. In a migration scenario, an organization could bring their existing SQL License to Azure and save on the cost of paying for the SQL license, reducing the Azure SQL Database cost to $7,200 for the three-year time period. Both the Azure SQL Database and App Service Plan are using reserved capacity to further reduce the cost of the environment.

This cost structure, which is reflected in the calculations used in Table 2 and Table 3, yields significant savings. For instance, the estimated cost to migrate to Azure without AHB was calculated to be $41,888, compared to $31,088 with Azure AHB, as shown in **Table 2**.

The cost per month for an Azure migration likewise changes significantly with AHB applied. A 4 VM deployment that was shown to cost $864 for Azure with AHB in **Table 3** would cost $1,164 without AHB. The $582 cost of a 2 VM Azure deployment with Azure AHB goes up to $882 without AHB.

The impact on sessions costs is comparable. At 4 VM, the cost goes from $0.79 per 10K sessions with Azure using AHB, to $1.06 per 10K sessions with Azure without AHB. At 2 VM, those figures are $0.53

and $0.80, respectively.

AWS Costs

The benchmark testing was performed with EC2 running four instances at the m5d.large size and an RDS instance using the db.m5.xlarge size. The Application Load Balancer is not bundled with Elastic Beanstalk and neither is network traffic—both are broken out as separate line items. **Table 15** shows the rundown of estimated costs.

Table 15. AWS Estimated Costs

AWS	ESTIMATED COSTS
EC2 Instances	$15,542
RDS Instance	$22,921
Network Traffic	$720
Application LB	$720
Support	$4,334
Total Cost	**$44,237**

Source: GigaOm 2020

8. About Ned Bellavance

Ned is an experienced IT practitioner with experience in the field. Ned has worked with Fortune 500 companies and SMBs across multiple verticals, developing and deploying both on-premises and cloud-based architectures. Ned has authored two books on the Azure Kubernetes Service and HashiCorp Terraform and holds several industry certifications from vendors including but not limited to Microsoft, VMware, AWS and Citrix.

9. About GigaOm

GigaOm provides technical, operational, and business advice for IT's strategic digital enterprise and business initiatives. Enterprise business leaders, CIOs, and technology organizations partner with GigaOm for practical, actionable, strategic, and visionary advice for modernizing and transforming their business. GigaOm's advice empowers enterprises to successfully compete in an increasingly complicated business atmosphere that requires a solid understanding of constantly changing customer demands.

GigaOm works directly with enterprises both inside and outside of the IT organization to apply proven research and methodologies designed to avoid pitfalls and roadblocks while balancing risk and innovation. Research methodologies include but are not limited to adoption and benchmarking surveys, use cases, interviews, ROI/TCO, market landscapes, strategic trends, and technical benchmarks. Our analysts possess 20+ years of experience advising a spectrum of clients from early adopters to mainstream enterprises.

GigaOm's perspective is that of the unbiased enterprise practitioner. Through this perspective, GigaOm connects with engaged and loyal subscribers on a deep and meaningful level.

10. Copyright