GIGAOM

Image credit: ImagesRouges

Michael Delzer, KK Verma, Evan Chisholm

Mar 28, 2022

Costs and Benefits of .NET Application Migration to the Cloud v2.0

A Guide to Cloud-Based Modernization of .NET Applications

Container Workload Migration, DevOps & Applications

Costs and Benefits of .NET Application Migration to the Cloud

A Guide to Cloud-Based Modernization of .NET Applications

Table of Contents

1. Summary

Thousands of legacy applications written using development frameworks such as ASP.NET are still being used by organizations, often running in on-premises environments. At the same time, many organizations have been moving to cloud computing as part of their digital transformation strategies. The COVID-19 pandemic has accelerated these digital transformation initiatives and increased cloud adoption as organizations have had to evolve business models to survive. They have augmented and transformed their existing applications to modern, cloud-based platforms to increase customer engagement, streamline operations, support a remote workforce, and lower costs.

Ripping and replacing applications is no longer feasible, and it is becoming increasingly difficult to justify for organizations struggling to cope with shrinking budgets. Technology decision makers need to decide what to do with these legacy applications. Three main choices are available: maintain the status quo and do nothing; migrate and modernize the applications to a modern, cloud-based environment; or rewrite and replace them. While the third option might appear to be the optimal solution at first glance, the overheads in terms of the effort, time, and cost required to rewrite what could amount to thousands of applications may not always be the best approach.

To help technology decision makers understand some of the dynamics behind modernizing applications, we conducted a field test to assess the benefits and costs of application migration. This is a repeat of a test that we ran last year, as we wanted to ensure that migrating legacy applications was still a feasible option. The field test walks through a migration scenario and evaluates its costs, performance, and benefits. The results were similar to the previous test, and they reinforced the following:

- Significant cost savings can be achieved by migrating applications to the cloud from on-premises infrastructures.
- Traditional .NET applications can benefit from such migration without refactoring underlying code.
- The Microsoft Azure solution offered a potential total cost of ownership (TCO) savings of up to 54% over running on-premises and 35% over running on AWS.
- Streamlined operations, simplified administration, and proximity to advanced cloud services are additional benefits.
- Built-in tools for Visual Studio and MSSQL Manager provide ease of use for database and application migrations.

For an exercise of this nature, it is critical to use legacy applications for the migration to the cloud, as this is a scenario that many organizations find themselves in, with the need to balance the application's requirements against the potential advantages of the cloud.

The field study looked specifically at .NET applications, which allowed different approaches to migration to be assessed, looking at the advantages and disadvantages of each. We discovered that moving to Microsoft Azure has a measurable TCO advantage over Amazon Web Services—with cost savings of 35% when applying Azure Hybrid Benefit (AHB) licensing—and that Azure offers additional benefits in terms of ease of migration. (**Figure 1**)

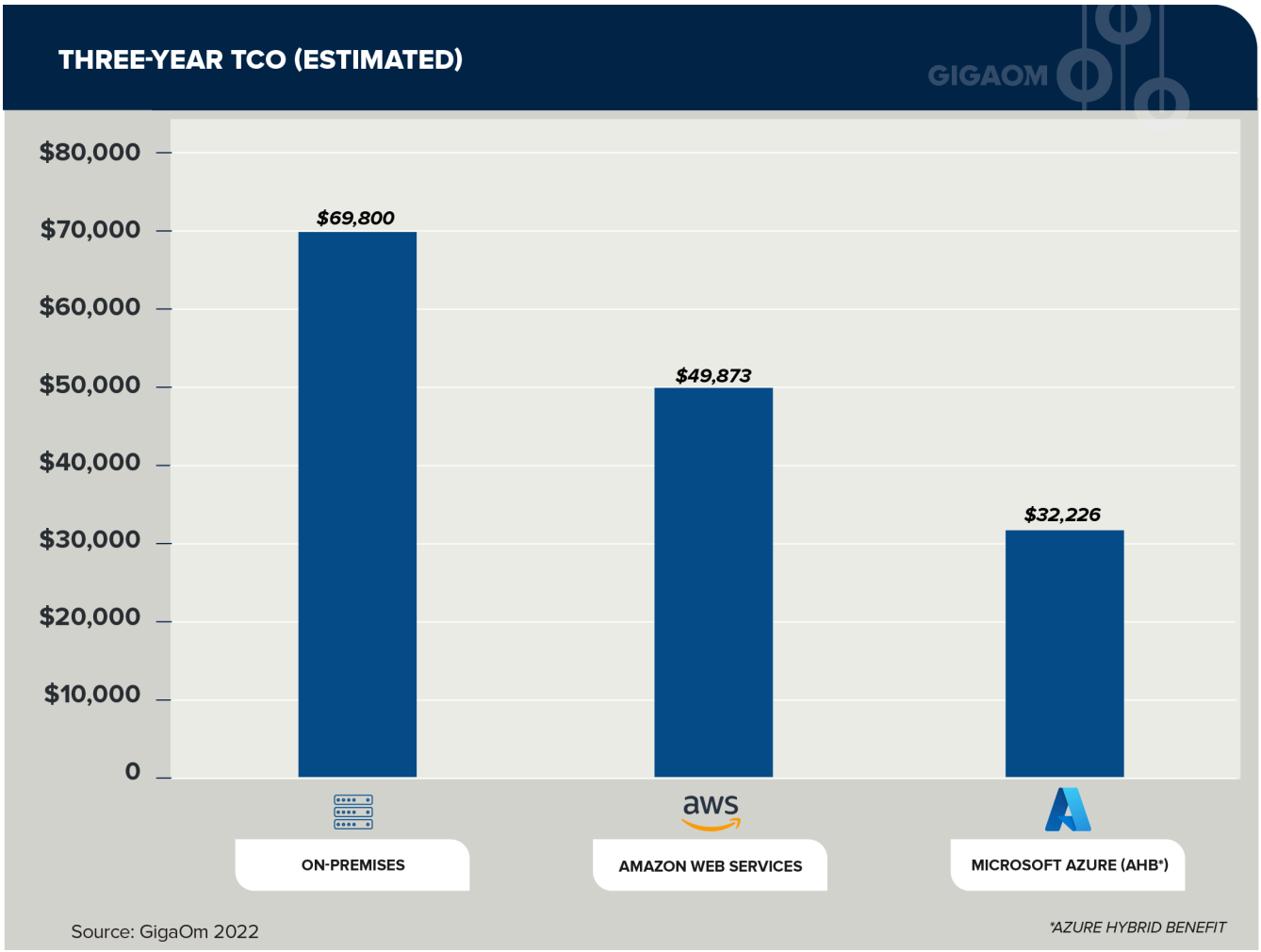


Figure 1. Three-Year Total Estimated Operating Cost

We concluded that any cloud migration frees up considerable time and resources to work on more strategic tasks that drive value to the business rather than maintaining resource-heavy applications. 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. Digital Transformation Is Being Driven by the Need for Modernization

Digital transformation initiatives have been underway for the last few years, with many organizations running multi-year strategies. COVID-19 has accelerated many of these strategies but has created a conundrum as many organizations have taken a hit to their profits due to the pandemic and suddenly found themselves having to combine accelerated digital transformation with reduced budgets. A consequence of this is to retain more legacy applications, modernize them, and migrate them onto more cost-effective cloud platforms.

There are thousands of legacy .NET applications still in use, and rewriting them all is not feasible. While they provide value to the organization, they are often expensive to maintain, inefficient, and often cause bottlenecks. Furthermore, application modernization has advantages over developing new applications. Employees are familiar with the applications, and as modernizations improve usability and productivity, they are easier to use, easier to maintain, and do not require the training that is needed if applications were replaced.

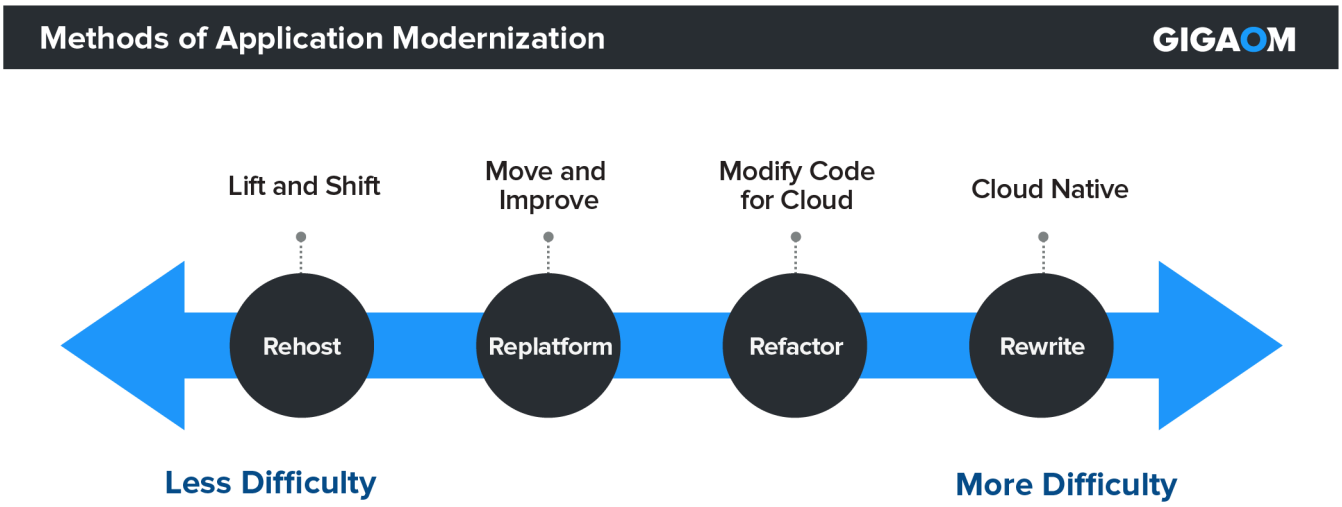
Migrating the modernized applications to the cloud enables organizations to benefit from the security built into the cloud platforms, rather than continually reviewing and updating security measures for on-premises applications. An additional benefit of cloud adoption is that with many employees now working from home and accessing corporate applications from outside the firewall following the pandemic, organizations can benefit from the cost benefits of the security measures provided by cloud providers. By contrast, securing on-premises systems for use by remote workers has required additional measures adding to the overall cost of on-premises environments.

Cloud migration is therefore important to businesses, which need to assess whether it is preferable to modernize rather than replace an application. Unless there are tangible benefits from transforming an application, any effort would be wasted. Options, benefits, and costs are considered in the section below.

3. Options and Benefits for Application Modernization

Among 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 platform as a service (PaaS).
- **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.



Source: GigaOm 2022

Figure 2. Methods of Application Modernization

Each option has its benefits. At one end of the spectrum, rehosting is relatively quick and simple to achieve in terms of the effort and cost involved. However, the organization cannot take advantage of the additional capabilities provided by the cloud providers, and it does not address the problem of running an outdated, legacy application with all of its inefficiencies. At the other end of the scale, a

complete rewrite will provide a modern application that is much more efficient, is optimized to the organization's requirements, and allows additional features provided by the cloud host such as analytics and artificial intelligence to be embedded in the applications. The major disadvantage to this approach is the budget and time required to rewrite applications.

Table 1. Benefits and Challenges of Modernization Approaches

Benefits and Challenges of Modernization Approaches GIGAOM

	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 2022

Table 1 shows that a replatform can provide a happy medium as it removes the need to maintain the infrastructure, which is still required with a lift-and-shift approach, but it is not as time-consuming or complex as a rewrite. 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 benefit from 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

These benefits respond to the need for greater agility in support of digital transformation. Each is covered 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 can 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 can 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 use deployment slots on a web application. Each deployment slot runs its 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 proper code coverage and testing are performed before a new build is deployed.

Backup and recovery of application data and replication to a separate region are simplified when using the cloud. Azure SQL Database can easily replicate data to a designated 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 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 to handle frequent scaling and unpredictable use. In other cases, an application could be enhanced by consuming another service, like adding a chatbot 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 before advances like .NET Core and microservice architectures . 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 offers.

Traditional ASP.NET applications backed by Microsoft SQL Server can be moved to the 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 back end required by most applications. Microsoft manages the underlying systems, 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 can generate environments that include an autoscaling group of EC2 instances and a deployment of its 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.

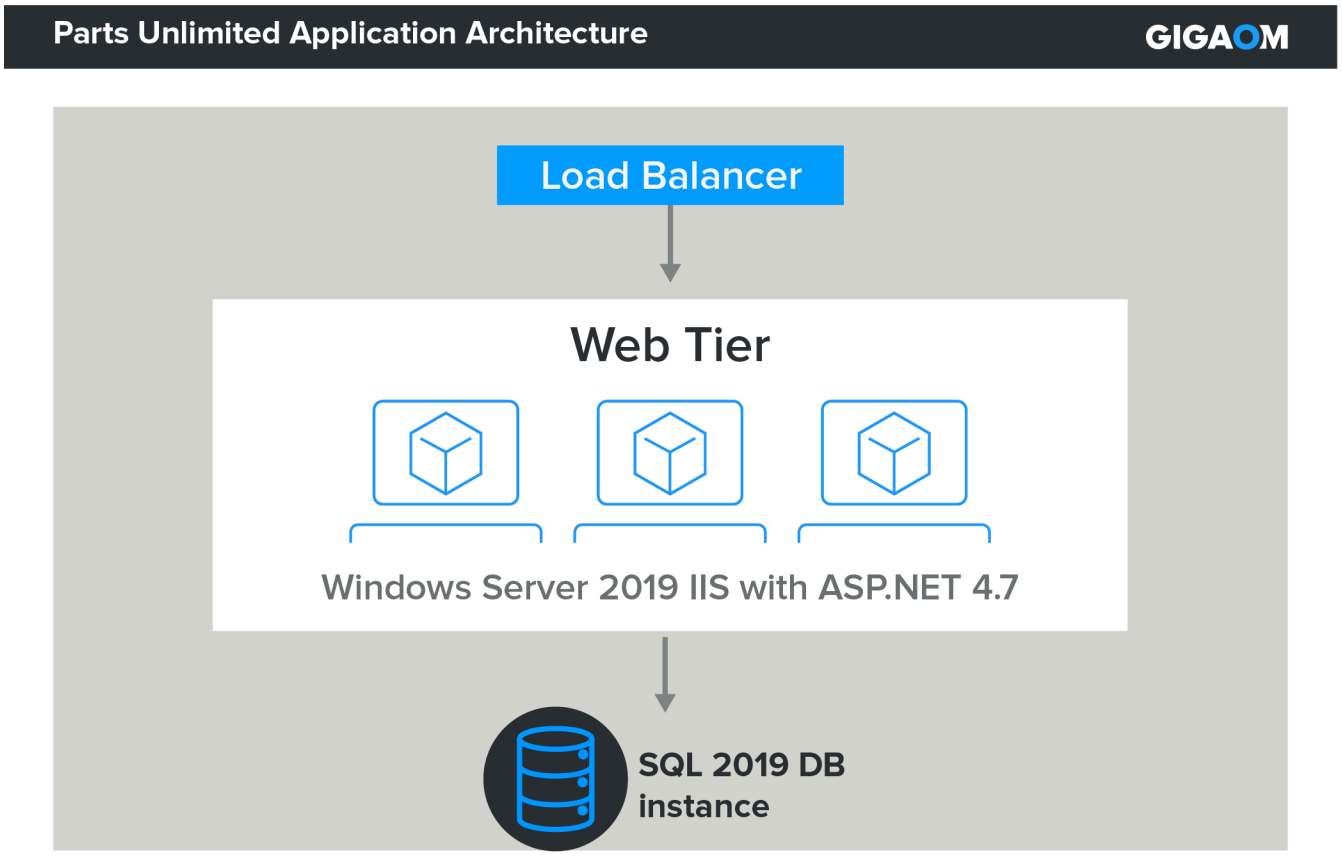
The similarities of these platforms make the migration of ASP.NET applications from on-premises environments simple and predictable while providing benefits over a lift-and-shift migration that simply uses infrastructure-as-a-service (IaaS) components.

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 perform a field test migration of an ASP.NET application from a traditional on-premises environment to both Microsoft Azure and AWS. 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 back end, as shown in **Figure 3**. The web store has an inventory of items that registered users can search and purchase. The application was installed in

each environment and assessed for performance, ease of migration, and additional features available from each cloud provider.



Source: GigaOm 2022

Figure 3. 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

Part of the test involved comparing and contrasting the performance of two leading cloud solutions on the market, Amazon AWS and Microsoft Azure, with a further option being to leave the application in an on-premises scenario. Each option was tested for ease of migration, performance, and TCO.

Migration

Overall, the migration process from on-premises to Azure and AWS was relatively straightforward, with Azure slightly easier. Microsoft's development suite has built-in functionality for migrating to Azure. This made the application deployment seamless via Visual Studio 2017. SQL migration from on-premises to Azure required no extra tooling, as it was able to use the built-in functionality within MSSQL Management Studio.

Using Azure, the entire infrastructure was spun up via Visual Studio. There are plenty of Terraform options available as well. On the application side, the Azure application deployment and infrastructure was fulfilled simultaneously via Visual Studio. The ability to change web.config without performing a reupload was a huge time saver, as it allowed the database connection strings <App Service Editor> to be tweaked. Troubleshooting was a much more pleasant experience with Debug Console.

In terms of AWS migration, the infrastructure was deployed via webGUI, which made for a relatively easy spin up. Migrating the database was a bit cumbersome, as we needed to upload .bak to an S3 bucket and import it. There was no built-in tooling or plugins for MSSQL Management Studio, which would have been helpful.

During application deployment, there were a few issues with the stickiness setting on the load balancer and opaqueness of logging. There was no way to reconfigure the application once deployed other than redeploying, which could take 10 to 15 minutes. In addition, it was only possible to request the previous 100 lines of logs via webGUI.

The on-premises option required the spinning up of the entire VCenter environment, the creation of a windows VM template, and the deployment of the servers to create the infrastructure. This was time consuming but made slightly better by using a Terraform project. Many steps would not have been necessary if we were deploying to an existing environment. Migrating the application from the development machine to IIS nodes and then cloning additional nodes was relatively easy as this is a well-established pattern for application deployments, and Visual Studio had built-in tools to address this.

The performance was similar across all environments, which is not surprising as we tried to ensure that similar resources were used for each option. The cloud environments performed slightly better, most likely due to more robust networking as individual machine resources never went above 10% utilization for CPU or RAM. More information on the specifics of the hardware can be found in the appendix.

The test was designed to show that the performance of a migrated application can be at least as good, if not better than, an on-premises instance. Once the application has been migrated, many updates can be added to improve performance, including 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 35% compared to AWS, demonstrating significant potential savings through migrating applications to Microsoft Azure. Despite slight price changes from the previous benchmark, Azure remains the most cost-effective option by some margin, particularly when the ease of migration is factored in. In addition to being more expensive than Azure, running in AWS 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 assumed that the headcount remains consistent, and we did not consider the larger data center or cloud deployment that might be part of an organization. Instead, we looked at the cost of running each environment for three years.

Hosting considerations also need to be accounted for. 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.

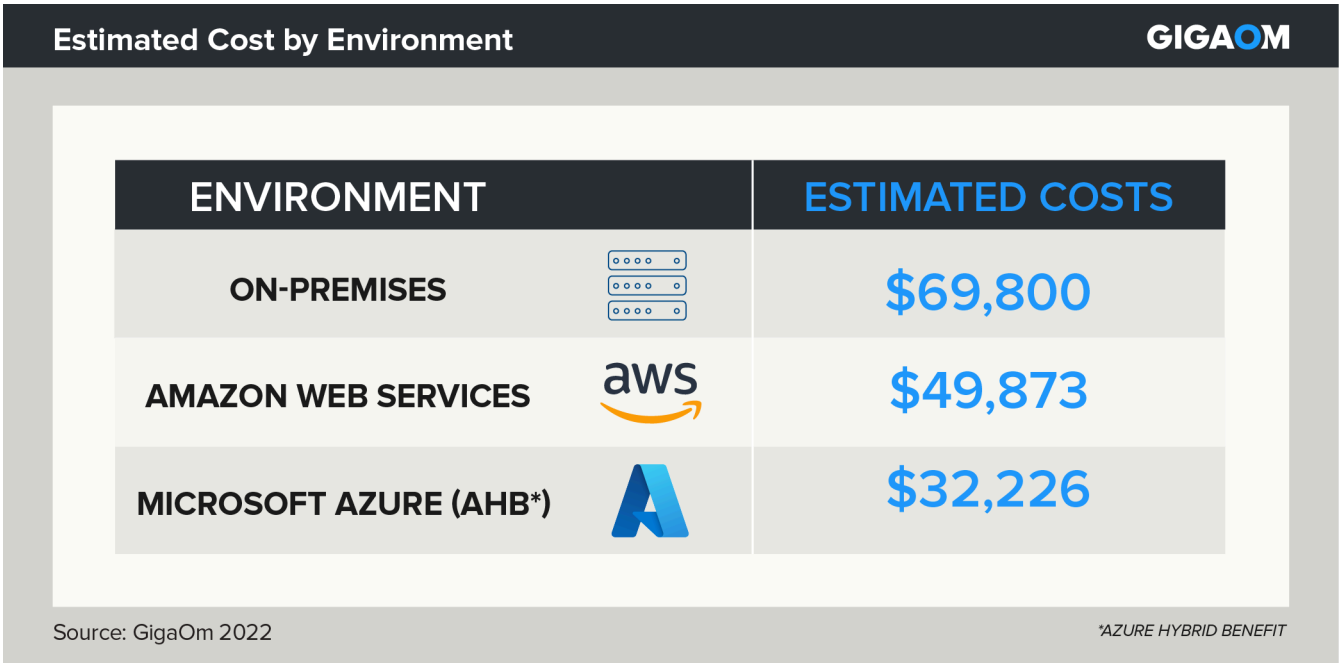
In terms of results, Microsoft Azure with Azure Hybrid Benefit (AHB) licensing had the lowest overall cost, saving an estimated 54% over the on-premises deployment and 35% over AWS.

AHB licensing allows existing Windows Server and SQL Server licenses with active Software Assurance to be applied to Azure virtual machines and Azure SQL Database instances. The estimated

cost of a SQL Standard license is $18,842, based on official pricing published in December 2021. 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 $18,842 to $8,334, which would reduce the total Microsoft Azure cost to $32,226. The use of existing SQL licenses is not an option on Amazon RDS. The addition of AHB creates a compelling savings of 35% over AWS.

These relative costs are shown in **Table 2**. Note that we do not incorporate dynamic scalability and bundled features, which also have a bearing. In addition, the pricing calculation for Azure uses reserved capacity pricing for App Service and Azure SQL and the East US 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



There are other, less quantitative aspects that we believe give Microsoft Azure an edge when 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 Microsoft handles upgrades, patching, and maintenance.

- **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 a greater administrative load that is hardly any different from consuming 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 of 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 4** 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 per Month for Each Environment (lower is better)

GIGAOM

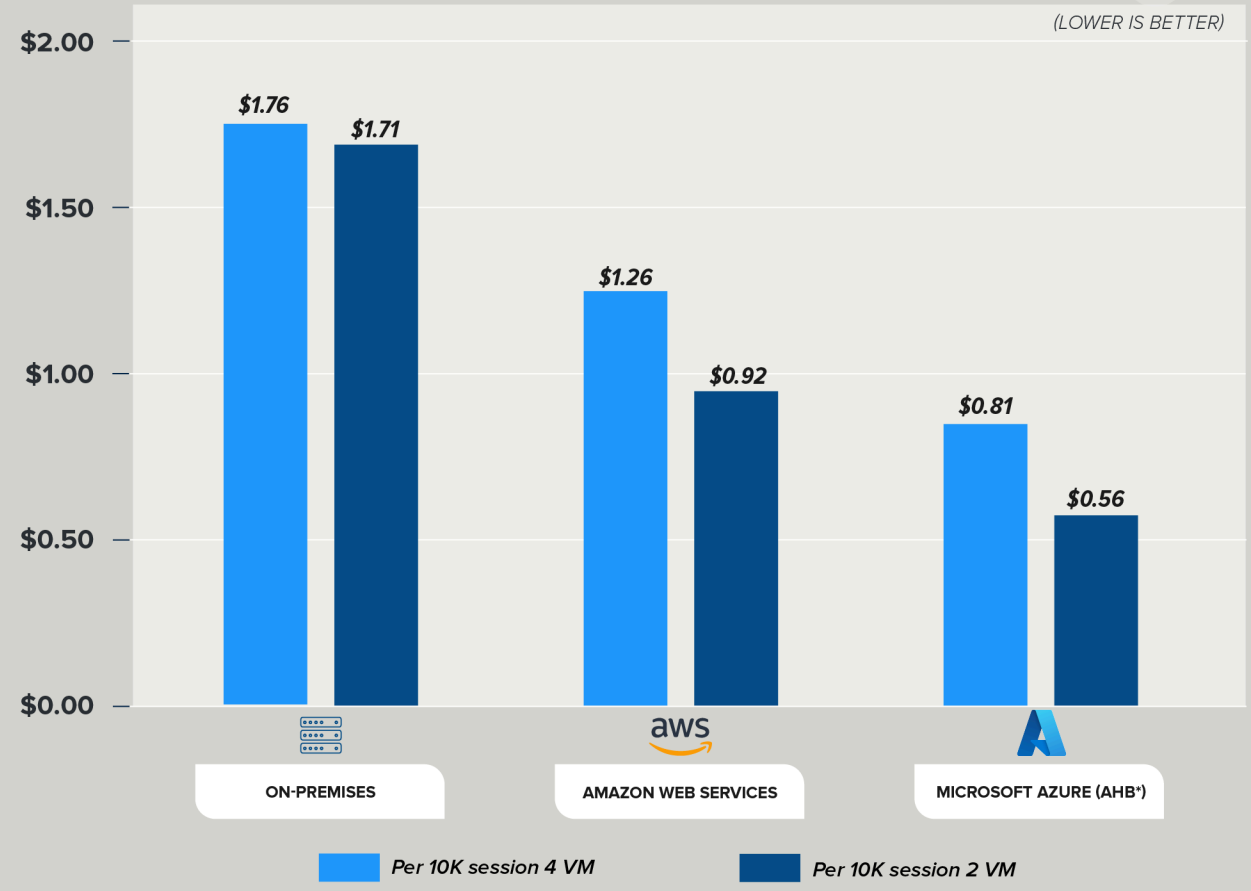
	COST 4 VM	COST 2 VM	PER 10K SESSION 4 VM	PER 10K SESSION 2 VM
On-Premises	$1,939	$1,883	$1.76	$1.71
Amazon Web Services	$1,385	$1,013	$1.26	$0.92
Microsoft Azure (AHB*)	$895	$613	$0.81	$0.56

Source: GigaOm 2022

*AZURE HYBRID BENEFIT

COST PER 10,000 SESSIONS FOR EACH ENVIRONMENT

GIGAOM



Source: GigaOm 2022

*AZURE HYBRID BENEFIT

Figure 4. Cost per 10,000 Sessions for Each Environment (lower is better)

Regardless of 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. Therefore, the Microsoft Azure with AHB environment represents a cost savings of 35% over AWS with four VMs and a savings of 39% with two 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, 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 are minimally disruptive and can be simple to test, enabling piloted migration, risk assessment, and cost evaluation before full deployment.
- Fully managed platforms provide the highest simplicity and the lowest administrative burden compared to in-house infrastructure and platform stacks.
- Migrating applications to the cloud provides access to additional capabilities offered by the cloud providers.
- 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 migration path, 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, such as 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, to unlock new functionality.
- Evaluation: You can map application dependencies across data sources, services, and

infrastructure to help identify group migration candidates.

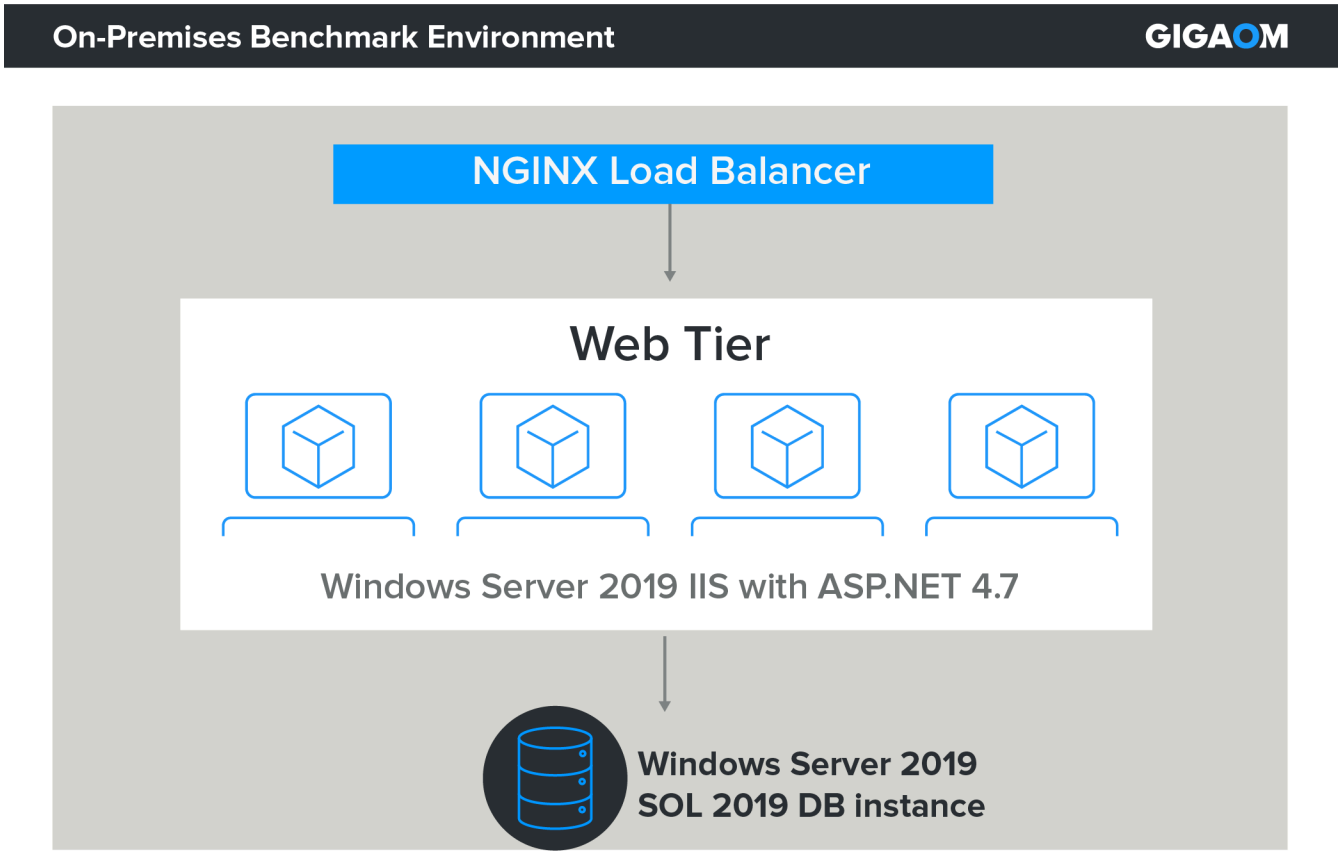
- Validation: Perform test migrations to validate functionality and performance and build experience around target environments, for example, in terms of what operational management dashboards are available.

Overall, the opportunity to move applications to the cloud does not have to be 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

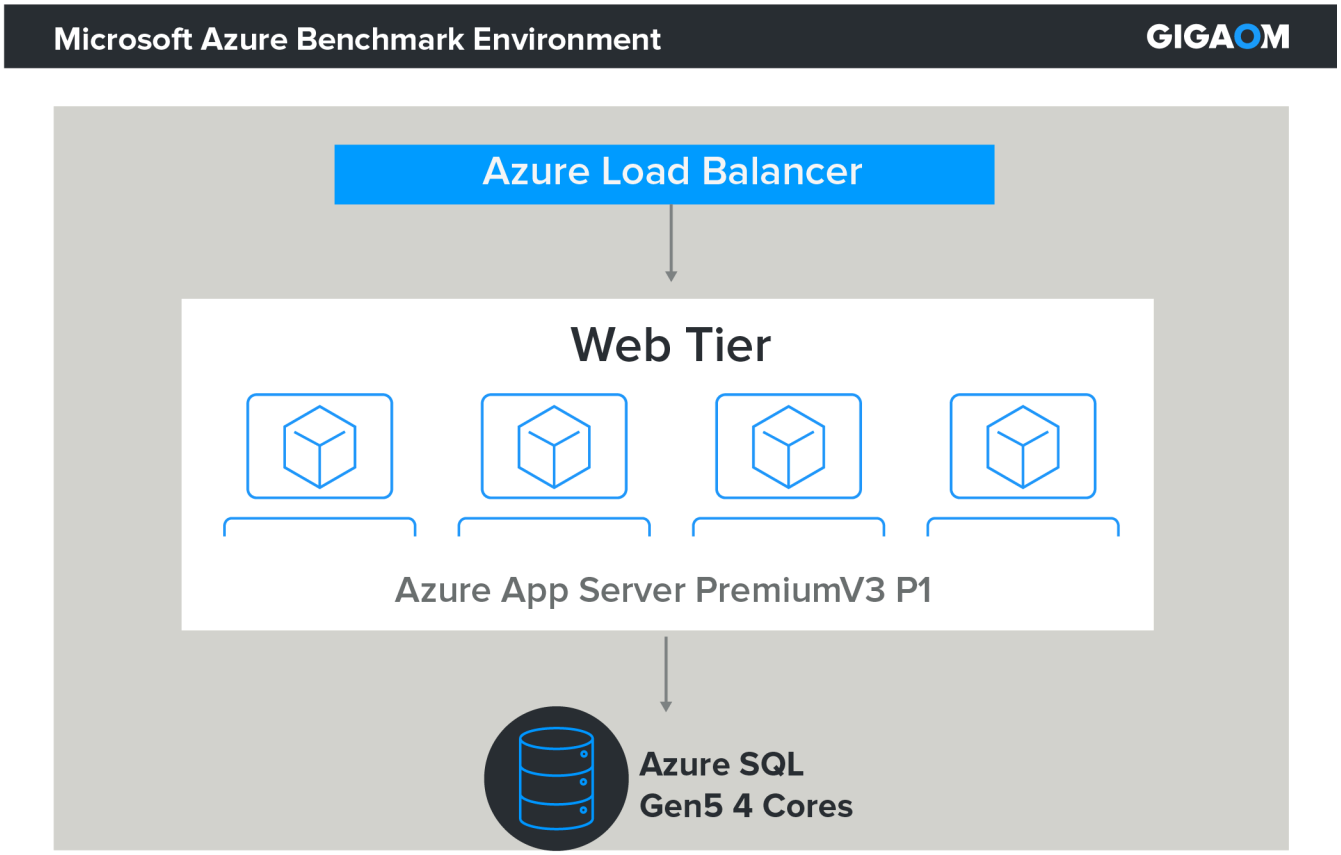
The on-premises environment had four web servers running Windows Server 2019 and a single database server running Windows Server 2019 and SQL Server 2019. The load balancer was a single virtual machine running Ubuntu LTS 20.04 with Nginx. **Figure 5** shows the architecture.



Source: GigaOm 2022

Figure 5. The On-Premises Benchmark Environment

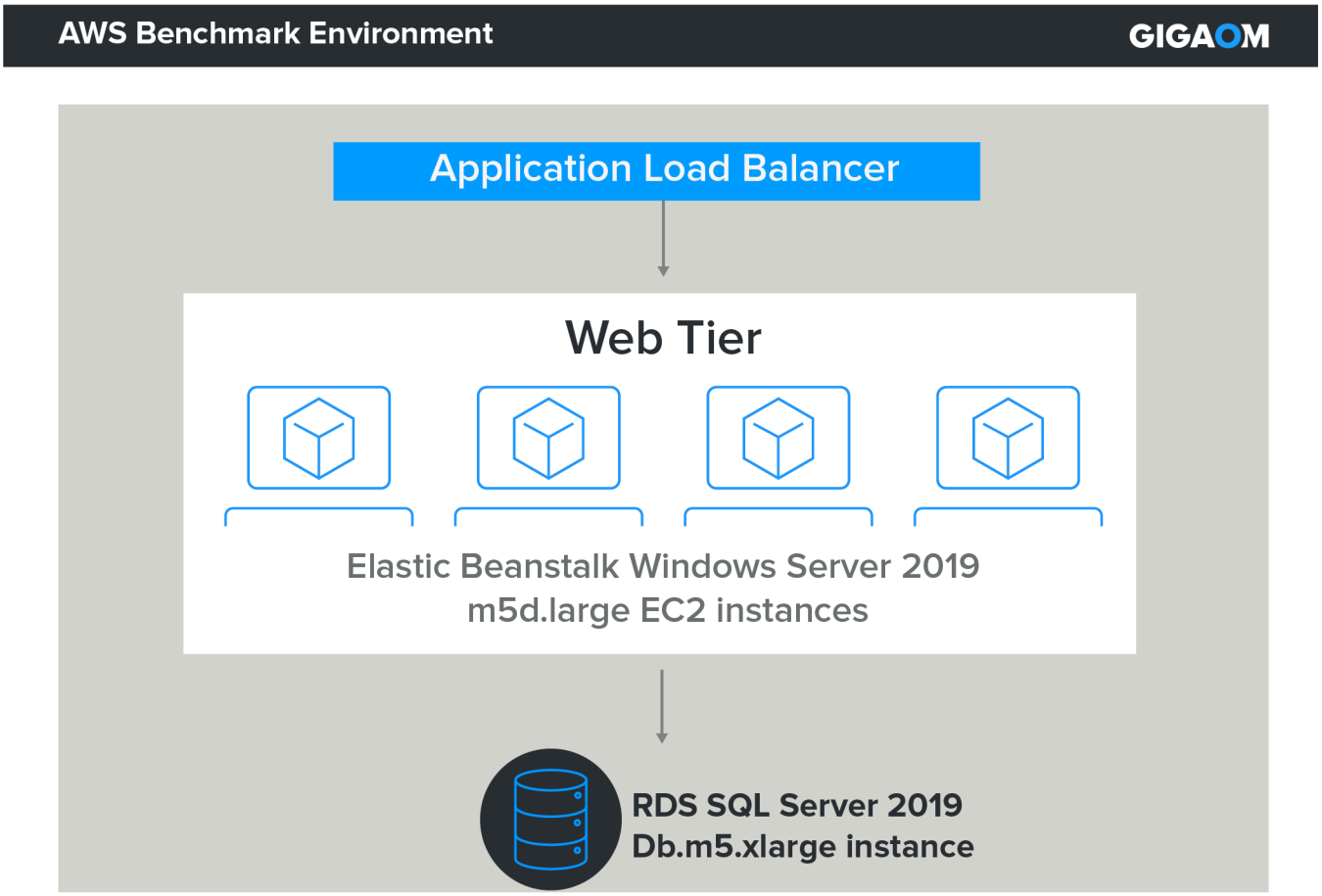
The Microsoft Azure environment shown in **Figure 6** used the Azure App Service with a Premium V3 App Service Plan, configured to run a minimum of four instances. Database services were provided by a Gen5 instance of Azure SQL Database with four vCores. Load balancing services are included as part of the Azure App Service.



Source: GigaOm 2022

Figure 6. The Microsoft Azure Benchmark Environment

The AWS environment shown in **Figure 7** 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 2019. Load balancing services were supported by an Application Load Balancer provisioned by Elastic Beanstalk.



Source: GigaOm 2022

Figure 7. 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 application's home page 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 presented below are the total number of successful sessions executed during each test and the average duration of each test. AWS led the way in successful sessions completed, especially on the Item search testing, as shown in **Figure 8**.

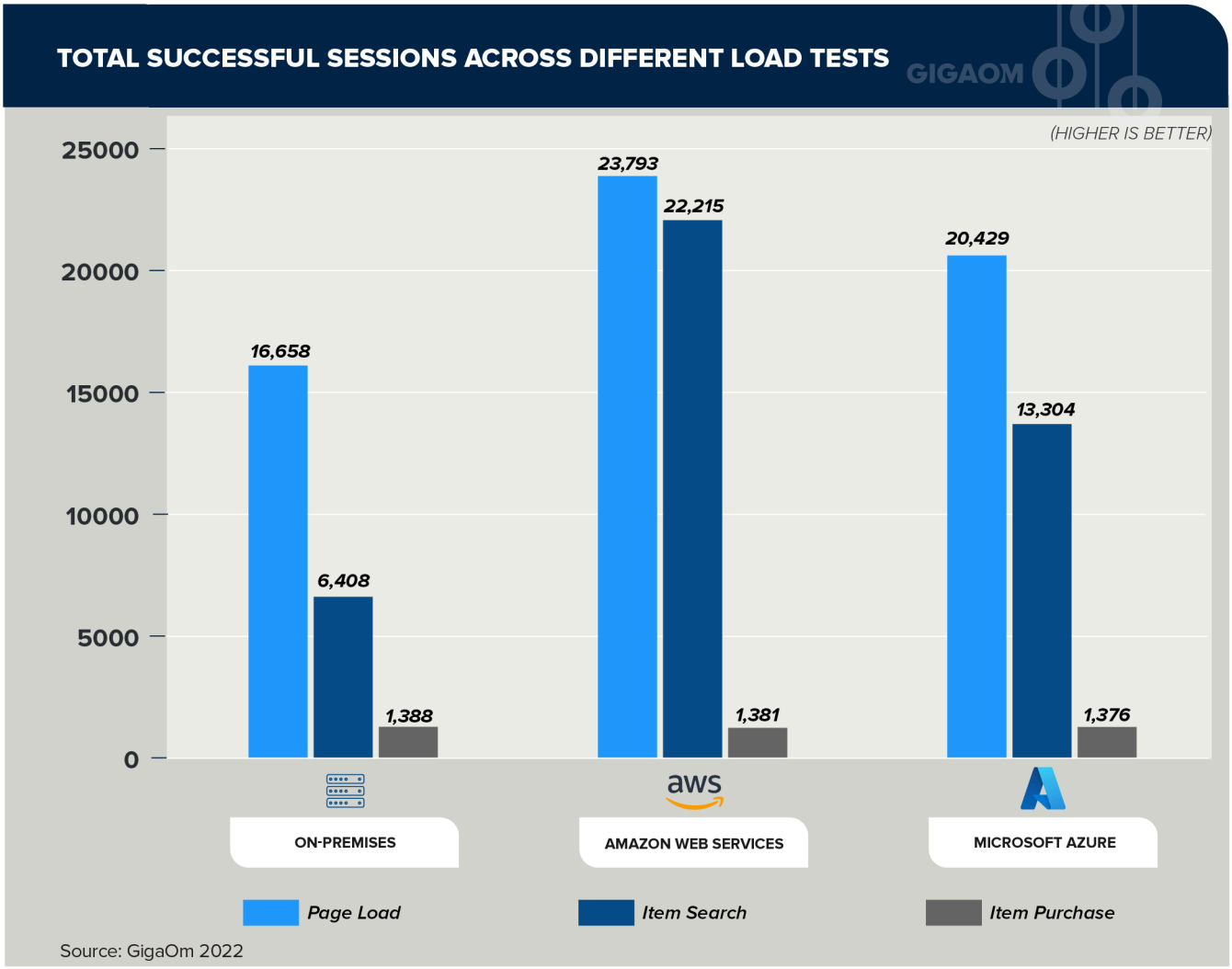


Figure 8. Total Successful Sessions Across Different Load Tests (higher is better)

The AWS environment had the lowest average duration for each test, with Azure demonstrating comparable numbers. **Figure 9** shows the results.

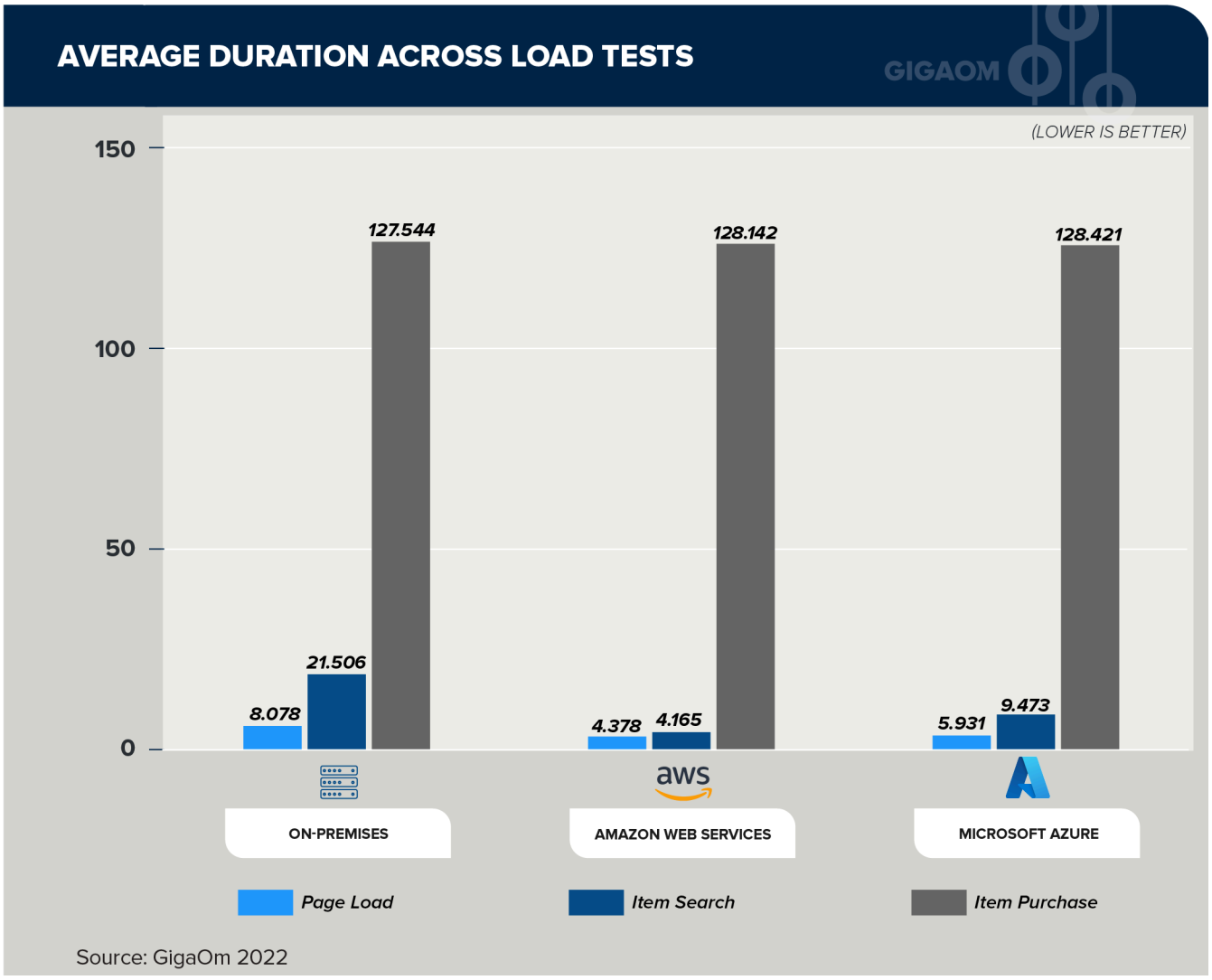


Figure 9. Average Duration Across Load Tests (lower is better)

Overall, the performance of all the environments was remarkably consistent, with both cloud services showing better performance over the on-premises environment for page load and item search. This makes sense because of the similar hardware and software used for each environment. However, on-premises had a better performance in terms of average duration than the cloud options for item purchase.

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 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 back end. 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 2017 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. The wizard handled changes to the database connection information.

The AWS Toolkit for Visual Studio 2017 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 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 created 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, Skylake (2015) is one of four architectures in play, as well as 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 server, 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

On-Premises Server Specifications GIGAOM

	QUANTITY	O/S	CPU	RAM (GB)	STORAGE (GB)
vCenter	1	N/A	2	10	250
Load Balancer	1	Ubuntu 20.04	2	8	40
Routing	1	Windows Server 2019	2	8	40
Web Server	4	Windows Server 2019	2	8	40
Database Server	1	Windows Server 2019	4	16	90

Source: GigaOm 2022

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 2019 Standard installed.

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

The load-balancer VM used Nginx 1.21.5 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 used 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 2019. 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 in the App Service Plan.

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 used 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 the Windows Server platform with IIS on Windows Server 2019.

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 four 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, equivalent to 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 Server and Database Components Across Environments GIGAOM

	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

*Remote Storage

Source: GigaOm 2022

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 2019. 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 ensured 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 the 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

CPU Cores Used in Each Environment GIGAOM

	WEB	DATABASE
On-Premises	Skylake	Skylake
Amazon Web Services	Skylake-SP or Cascade Lake	Skylake or Cascade Lake
Microsoft Azure	Cascade Lake	Skylake or Broadwell

Source: GigaOm 2022

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

Testing Configuration

Three different tests were performed across the three environments to check 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 worldwide. Each test is composed of multiple load injector nodes, each running one or more test instances. LoadView assesses each test to determine the number of test instances supported by a single node.

It's important to note that the tests run from within AWS, and one of the tested environments also runs in AWS. Specifically, the AWS environment runs in the US East (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 application's main page and evaluate how long it takes to load.
- Item search: Access the application's main page and search for an item using the search box. Click on the first result from the search.
- Item purchase: Access the application's main page 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, and the overall

time of the test.

Testing Results

Home Page Load

The *Home Page Load* 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

Home Page Load Test Specifications GIGAOM

	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 2022

Each environment was subjected to the same test with the same parameters. As noted above, the AWS environment runs 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

Home Page Load Test Results GIGAOM

	On-Premises	Amazon Web Services	Microsoft Azure
Success Sessions	16658	23793	20429
Failed Sessions	0	0	0
Average Duration	8.078	4.3784	5.9306
Max Duration	30.879	14.928	13.344
Std Deviation	1.2161	0.9074	1.3653

Source: GigaOm 2022

AWS was able to load more sessions within the one-hour time frame with Azure close behind. OnPrem had a much longer session duration for loading the page.

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 five users, increased by five 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

Item Search Test Specifications GIGAOM

	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 2022

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

Item Search Test Results GIGAOM

	On-Premises	Amazon Web Services	Microsoft Azure
Success Sessions	6408	22215	13304
Failed Sessions	0	0	0
Average Duration	21.5063	4.1652	9.4733
Max Duration	32.575	14.427	17.611
Std Deviation	1.8676	1.1989	3.8292

Source: GigaOm 2022

LoadView could complete more sessions in the AWS environment by a large margin. This is most likely due to the LoadView Agents being located in AWS itself.

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. At 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 five users, increased by five 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

Item Purchase Test Specifications GIGAOM

	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 2022

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

Item Purchase Test Results GIGAOM

	On-Premises	Amazon Web Services	Microsoft Azure
Success sessions	1388	1381	1376
Failed sessions	0	0	0
Average duration	127.5439	128.1419	128.4206
Max duration	147.323	146.868	148.328
Std deviation	6.3147	5.9667	6.1923

Source: GigaOm 2022

All three options performed equally well with the number of concurrent users. LoadView was set to have a “normal user” delay between tasks. This may have led to such similar timings. The servers were never taxed during this scenario.

Test Summary

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

Although the AWS and Azure environments were configured to autoscale the 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, reducing the number of web server instances to three or possibly two would be safe. 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 of one or three years. The total cost is an estimate of running the application for three years.

The cost calculations focus solely on running the applications, 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 significantly impact 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.

The savings can be profound. Our modeling determined that a deployment built on Azure with Azure Hybrid Benefit licensing can yield savings of 35% over the cost of an AWS deployment, and 54% over the cost of an on-premises deployment.

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 GIGAOM

On-Premises	Estimated Costs
Host Hardware	$20,000.00
Hosts SnS (15% per year)	$9,000.00
Power	$4,000.00
Vmware Licenses (4x CPU)	$10,500.00
Vmware SnS (30 % per year)	$10,500.00
Windows License (5x)	$5,000.00
Windows SQL Standard License	$10,800.00
Total	$69,800.00

Source: GigaOm 2022

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 four vCPUs. The resulting cost breakout is shown in **Table 14**.

Table 14. Microsoft Azure Estimated Costs

Microsoft Azure Estimated Costs GIGAOM

Microsoft Azure	Estimated Costs
App Service	$20,292
Azure SQL*	$18,842
Support	$3,600
Total	$42,734

*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.

Source: GigaOm 2022

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 paying for the SQL license, reducing the Azure SQL Database cost to $8,334 for the three years. Both the Azure SQL Database and App Service Plan are applying the reserved instance pricing option 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 $42,735, compared to $32,226 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 cost $895 for Azure with AHB in **Table 3** would cost $1,187 without AHB. The $613 cost of a 2 VM Azure deployment with Azure AHB goes up to $905 without AHB.

The impact on session costs is comparable. At 4 VM, the cost goes from $0.81 per 10K sessions with

Azure using AHB to $1.08 per 10K sessions with Azure without AHB. At 2 VM, those figures are $0.56 and $0.82, 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 GIGAOM

On-Premises	Estimated Costs
EC2 Instances	$15,516
RDS Instance	$22,693
Network Traffic	$720
Application LB	$720
Support	$10,224
Total	$49,873

Source: GigaOm 2022

8. About Michael Delzer

Michael Delzer is a global leader with extensive and varied experience in technology. He spent 15 years as American Airlines' Chief Infrastructure Architecture Engineer, and delivers competitive advantages to companies ranging from start-ups to Fortune 100 corporations by leveraging market insights and accurate trend projections. He excels in identifying technology trends and providing holistic solutions, which results in passionate support of vision objectives by business stakeholders and IT staff. Michael has received a gold medal from the American Institute of Architects.

Michael has deep industry experience and wide-ranging knowledge of what's needed to build IT solutions that optimize for value and speed while enabling innovation. He has been building and operating data centers for over 20 years, and completed audits in over 1,000 data centers in North America and Europe.. He currently advises startups in green data center technologies.

9. About KK Verma

KK Verma is a career Chief Information Officer (CIO) with over three decades of experience, specializing in building and growing organizations by leveraging technology to drive revenue, profitability, and customer excellence. He has led technology departments for some of the well-known firms during his career – Conduent, Citigroup, IBM, American Airlines, Hewlett Packard.

KK is a strategic advisor and partner, who has led several large-scale digital transformations for organizations, with proven results. He brings a unique blend of leadership skills, organizational development, and relationship-building skills to create new business models to thrive in the new economy.

Mr. Verma received his Master of Business Administration (MBA) from University of Texas and Bachelor of Computer Engineering from Pune University. He is currently residing in Dallas, Texas for over 25 years with his family.

10. About Evan Chisholm

Evan Chisholm is a seasoned IT engineer focused on process automation and up-skilling operations teams to adopt DevOps practices. Evan has worked with organizations of all sizes (from startups to Fortune 500) across many verticals, including fintech, manufacturing, and service delivery. He aims to eliminate “toil” and empower engineers to do more than they imagined.

11. 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.

12. Copyright