

Microsoft Azure Service Bus, Web App, and Amazon Aurora

Using Microsoft Azure and Amazon Web Services in Concert

Chris Dusyk
200268705

Abstract

Cloud services provide users and technology professionals an easy and intuitive way to create and develop services and applications. These services are hosted on virtualized servers in large datacentres, existing all over the world. Generally, cloud services help reduce cost to companies by removing the hardware upfront and maintenance costs. Instead, customers are charged a (generally lower) hosting fee based on what they use. All of this infrastructure is split across many virtual resource managers in these datacentres.

The two largest cloud service providers are Microsoft Azure and Amazon Web Services. Both providers offer very similar services, however their approaches tend to be somewhat different. It's important to evaluate what each provider offers, and how their service operates and is managed, and compare against your needs as a consumer of these services. If necessary, it is easy to combine services from both to create a powerful application and infrastructure that meets these specific needs. This paper will explore this approach, using Amazon's Aurora database engine in concert with Microsoft's Service Bus, App Service, and Cloud Service Worker services to create a basic order processing application.

1. Introduction

Cloud services are rapidly becoming a major part of how we build software systems, particularly in the business domain. Cloud services offer powerful infrastructure with minimal effort and cost, making them an attractive platform for businesses to migrate their infrastructure and applications to. There are many different cloud providers with a variety of services, however the three main ones are: Microsoft Azure, Amazon Web Services (AWS), and Google Cloud Engine. Each of these three providers offers a large variety of different services, however this will primarily focus on a few of the services available in Azure and AWS.

The project application makes use of the Amazon Aurora database engine, hosted in AWS. Aurora is Amazon's custom database engine, designed to take advantage of the architecture available in a large datacenter, as opposed to a traditional database server. Aurora will be discussed in more detail further in this paper.

The core of the application is composed of three components in Microsoft Azure. First, the front-end application is hosted as an ASP.NET 4.6.1 "web app" in Azure. The web application

service hosts a website in a self-contained web server. There are different hosting packages that decide the resources allocated to the virtual server host, meaning you pay for the server resources you need. Next, there is an Azure Service Bus, Microsoft's version of a message queue, that accepts database write commands from the order creation part of the web application. The Service Bus queues the message, in a first-in-first-out manner (FIFO), and relays the messages in order to the worker service. The final component of the application is the worker service. This is an Azure Cloud Service worker service. The worker service is an API service that is triggered by a message signal from the Service Bus. The worker service unpackages the data from the Service Bus and inserts it into the database. There are many advantages to this architecture, which will be explored in more detail later.

Using cloud services together provides a strong and scalable system architecture, with typically high cost savings. It reduces the management and maintenance overhead, while making it easy to scale resources based on usage.

2. Amazon Web Services

Amazon Web Services (AWS) is Amazon's cloud services collection. There are many different options and services available, and everything is tied to a regular Amazon account. AWS' biggest strength is their infrastructure, as AWS gives users very granular control over every aspect of their cloud services.

AWS has many of the same services that Azure does, including a message queue service, an application host service, and many more. While there are implementation differences between the two, they provide very similar services. AWS began in 2006, when Amazon officially launched their Elastic Compute Cloud (EC2) service, providing a host for companies and individuals to rent computer space in the early days of the cloud (Mohammed, 2009). AWS was the first major cloud services provider to break into the market, and has consistently held a strong lead over all other providers when it comes to the breadth and strength of the products offered. AWS has datacentres all over the world, including in Montreal, Canada. This gives them a huge advantage with government contracts, as most government systems need to keep their data in Canada.

In this project, the only AWS product used was the Amazon Aurora database. Aurora is Amazon's proprietary database engine, designed to make use of the distributed nature of a large datacentre. It is very high performance and easy to migrate to.

2.1 Amazon Aurora

Aurora is Amazon Web Services' (AWS) custom relational database engine. It uses the MySQL structure, attached to AWS' new database engine. The database engine makes use of Amazon's distributed cloud services platform as opposed to a traditional database server. The engine itself is sharded across many virtual processors, allowing for very high performance and throughput.

Under-the-hood, Aurora is very different than any current database engine. It's designed around the fact that it's hosted in large datacenters where the database workload can be distributed across many virtual processors. Unfortunately, specific details are hard to come by as this is Amazon's proprietary technology, and is directly competing with other products from Microsoft Azure and Google Cloud.

2.1.1 Performance

On the largest instance container, Aurora can achieve up to 500,000 reads and 100,000 writes per second, as per Amazon's documentation. This is all done at very low latency, particularly if the application and infrastructure accessing the Aurora database is hosted in AWS as well. This is achieved by distributing all calculations across many SSD-backed virtual hosts, using quorums to maintain data atomicity. Quorum is a distributed system technique in which each atomic operation is given a vote on whether an operation is a commit or abort, $V_a + V_c > V$, where $0 < V_c, V_a \leq V$. Before a transaction completes, all operations involved must vote. To commit the transaction, it must obtain a commit quorum V_c . To abort the transaction, it must obtain an abort quorum V_a .

Another powerful tool to help with performance is the "push-button compute scaling". This refers to AWS' method of scaling up resources available to an RDS instance by pushing a button in the AWS management console. This provides a quick and easy way to provide more resources to the Aurora instance, giving it more power and throughput to process a heavy transaction workload. There are also options for setting up auto-scaling, where AWS will scale up the resources for an instance automatically given certain parameters around network and processor load on the instance. This makes managing an RDS instance very simple and helps eliminate a lot of maintenance time and cost.

2.1.2 Security

RDS instances run in total network isolation. Amazon achieves this by requiring users to set up Virtual Private Cloud (VPC). Various instances can run in a single VPC, and only applications and services inside that VPC can access each other. A Virtual Private Network (VPN) can be set up to establish a connection to a VPC, making it easy to control who and what can access the VPC and contained applications and services.

Amazon also uses end-to-end AES-256 encryption for all data transit, provided a very strong layer of security between the database instance and the outside world.

2.1.3 Advantages and Disadvantages

Aurora has many advantages as a database engine. To start with, it's very powerful and quick. Independent tests suggest around 5x the throughput of a MySQL server, with potential gains of up to 10x the throughput. This makes it a very desirable engine for systems with many users and much data. Another advantage is the self-replication features. By default, Aurora replicates across at least 3 availability zones. If anything happens to the primary instance, it will immediately failover to another availability zone. As well, Aurora is designed to be self-healing. This means that if any data corruption is detected in the main zone, it will attempt to correct it by combining the remaining availability zones' data and cleaning up the main zone's. The potential issue with this is that the other zones could potentially be corrupted as well and it may overwrite good data in the main zone. However, this is very unlikely to occur across many zones at once.

One of the biggest advantages, however, is the cost. Amazon prices Aurora very low in comparison to most other database services. For example, running a basic Aurora database (db.r3.large instance) will cost around \$0.290 per hour. Running a comparable MS SQL Server instance (db.r3.large instance) will cost \$1.020 per hour (AWS | Amazon RDS | Pricing, n.d.).

Aurora's other biggest advantage is both an advantage and a disadvantage. The top-level structure is a fork of MySQL, making it very easy for DBAs and programmers to transition to using it. As well, all existing MySQL management tools and drivers work with Aurora. Amazon offers a tool in AWS to help migrate an existing MySQL setup into Aurora, making it a very easy transition. However, it inherits MySQL's biggest weakness, in that it has very barebones organization and tooling. MySQL, especially in comparison to engines like MS SQL Server and Oracle, has minimal organization control. For example, Oracle packages are a powerful and convenient way to organize related stored procedures. MS SQL Server offers the ability to organize stored procedures and tables in virtual folders and subfolders, making it easy to work with. MySQL is unable to offer these sort of convenience.

Businesses may also be unwilling to trust their data to a black box server setup, however this is a potential disadvantage to any cloud solution. A traditional database server installation is easy to control and administrate at a low level, something that a cloud system will manage itself. This attitude is disappearing; however, it can be a barrier to entry for some.

3. Microsoft Azure

Azure is Microsoft's cloud services suite. It offers a wide range of products and services, primarily centered around the Microsoft stack. Azure products are managed through the Azure Web Portal. Azure, like AWS, has datacentres all over the world, however it does not have any in Canada. This does give it a slight disadvantage against AWS.

While Azure is generally targeted towards the Microsoft technology stack, it does support several other common frameworks. For example, the web application engine also supports

Python, NodeJS, Java, and PHP, as well as the Microsoft ASP.NET framework. There are also VM images for various Linux servers, as well as Windows Server. So while there is a clear targeting towards the Microsoft technology stack, you can still use all the same frameworks and options as you can in AWS.

Several Azure services are used in this project. To start with, the front-end web application is an ASP.NET 4.6.1 web application, hosted as a Web Application Service in Azure. The front-end application connects to a Service Bus and sends database write messages down through it. On the other end of the Service Bus, a Worker Role web service receives the message, parses the data, and inserts it into the database. All of these services connect intuitively as they are all built on the same stack and are well documented.

3.1 Web Application Service

Azure App Service is a managed platform for hosting application code (known as a Platform-as-a-Service, or PaaS), removing the need for managed servers and infrastructure. Web applications are a sub-component of the App Service. Web applications can be directly published to and hosted in App Service.

Web applications in the App Service have several major advantages. To start with, most popular languages and frameworks are supported, including:

- ASP.NET
- NodeJS
- Java
- PHP, with an environment template for Laravel
- Python
- Golang (beta)

Microsoft also has direct publish tooling for all major Integrated Development Environments (IDE), making it easy to write and publish applications to Azure (Espinosa, 2016).

Another major advantage to the App Service is its autoscaling functionality. There are several options to set up parameters for resource usage. As soon as a threshold is passed for resource requests, the autoscaling system can clone the App Service environment and automatically load balance it. This makes it very easy to handle unpredictable load bursts, which is a major issue in planning for resources for an application server.

The App Service offers a feature called “deployment slots”. Deployment slots are actually live web apps with their own hostnames. Web app content and configurations elements can be swapped between two deployment slots, including the production slot. Deploying your application to a deployment slot has the following benefits:

- You can validate web app changes in a staging deployment slot before swapping it with the production slot.

- Deploying a web app to a slot first and swapping it into production ensures that all instances of the slot are warmed up before being swapped into production. This eliminates downtime when you deploy your web app. The traffic redirection is seamless, and no requests are dropped as a result of swap operations. This entire workflow can be automated by configuring Auto Swap when pre-swap validation is not needed.
- After a swap, the slot with previously staged web app now has the previous production web app. If the changes swapped into the production slot are not as you expected, you can perform the same swap immediately to get your "last known good site" back.

(Lin, 2016)

Finally, the App Service's biggest advantage, especially for businesses, is the hybrid connection. Hybrid Connections are a feature of Azure BizTalk Services. Hybrid Connections provide an easy and convenient way to connect the Web Apps feature in Azure App Service (formerly Websites) and the Mobile Apps feature in Azure App Service (formerly Mobile Services) to on-premises resources behind your firewall (Ohlinger, 2016). This makes it very easy for companies to either partially migrate to Azure, or use a staged migration to move to Azure.

3.2 Service Bus

The Service Bus relays messages through a message queue, allowing for framework-agnostic collection of data that is more resilient to a network interruption on the part of a client device.

Service Bus is a multi-tenant cloud service, which means that the service is shared by multiple users. Each user, such as an application developer, creates a namespace, then defines the communication mechanisms she needs within that namespace (Manheim, Azure Service Bus, 2016). Figure 1 describes this architecture.

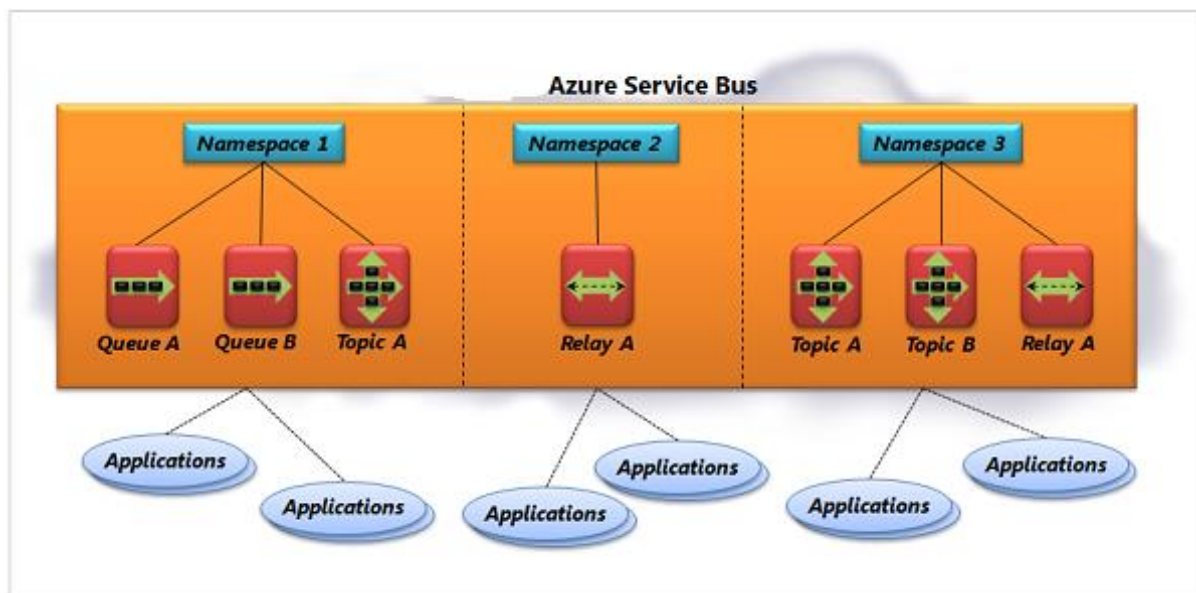


Figure 1. Image source: <https://azure.microsoft.com/en-gb/documentation/articles/service-bus-fundamentals-hybrid-solutions>

Within a namespace, there are four types of communication mechanisms, each of which connects applications in a different way. The types are:

- **Queues:** allow one-directional communication. Each queue acts as a broker that stores messages until they are received on the other end. Each message is received by a single recipient.
- **Topics:** provide one-directional communication using subscriptions. A single topic can have multiple subscriptions. Like a queue, a topic acts as a broker, but each subscription can optionally use a filter to receive only messages that match specific criteria (Manheim, Azure Service Bus, 2016).
- **Relays:** provide bi-directional communication. Unlike queues and topics, a relay doesn't store in-flight messages (i.e. it is not a broker). Instead, it just passes them directly to the destination application.
- **Event Hubs:** provide event and telemetry ingress to the cloud at massive scale, with low latency and high reliability (Manheim, Azure Service Bus, 2016).

This project uses a basic Service Bus queue, which will be described in greater detail.

3.2.1 Queues

The queue follows a simple workflow. A sender sends a message to a Service Bus queue, which holds the message until a receiver can pick it up and process it. The basic architecture is shown in Figure 2 below.

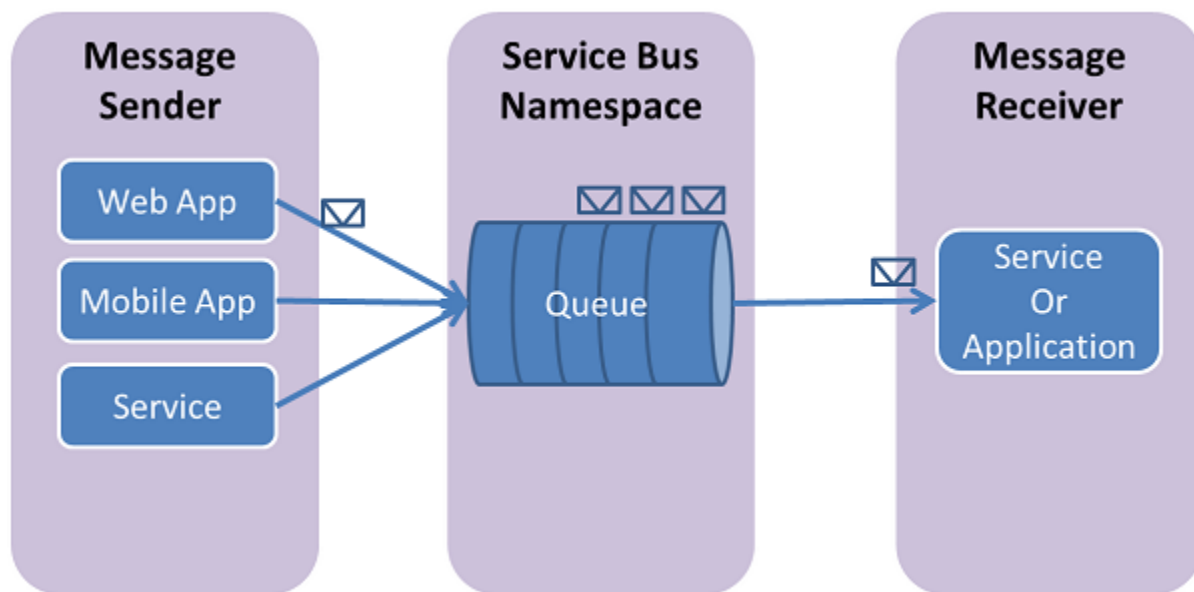


Figure 2. Image source: <https://azure.microsoft.com/en-us/documentation/articles/service-bus-dotnet-how-to-use-queues/>

Each message has two pieces: a set of properties (key/value pairs), and a binary message body. A receiver can read a message from a queue in two different ways. The first option, called “ReceiveAndDelete”, removes a message from the queue and immediately deletes it. This is a lightweight approach, however it has the disadvantage that if the receiver crashes while

processing, the message is lost. Since it's already been marked as received and removed from the queue, there is no way for another receiver to consume it. To combat this, the second option, called "PeekLock", removes the message from the queue when it processes it, but does not delete it. It instead locks the message, making other receivers unable to view it, until one of three events occurs:

1. If the receiver processes the message successfully, it calls the Complete function, and the queue deletes the message.
2. If the receiver decides that it can't process the message successfully, it calls the Abandon function. The queue removes the lock from the message and makes it available to other receivers.
3. If the receiver calls neither of these within a configurable period of time, the queue assumes the receiver has failed. In this case, it behaves as if the receiver called Abandon, making the message available to other receivers.

3.3 Cloud Services

Cloud Services is a Platform-as-a-Service (PaaS), much like the App Service. These are designed to support applications that are scalable, reliable, and cheap to operate (De George, 2015). Both App Service and Cloud Services are hosted on Virtual Machines (VM). However, users have more control over the VMs in Cloud Services. In the App Service, users see the application itself and do not have an option to connect to the underlying VM directly, either to manage it or install software. All of this is managed by Azure. With Cloud Services, users can directly connect to the underlying VM and modify most aspects of it. This ultimately means that it's quicker and easier to get a web application up and running in the App Service, at the cost of granular control. Figure 3 shows the general architecture of a Cloud Services application setup.

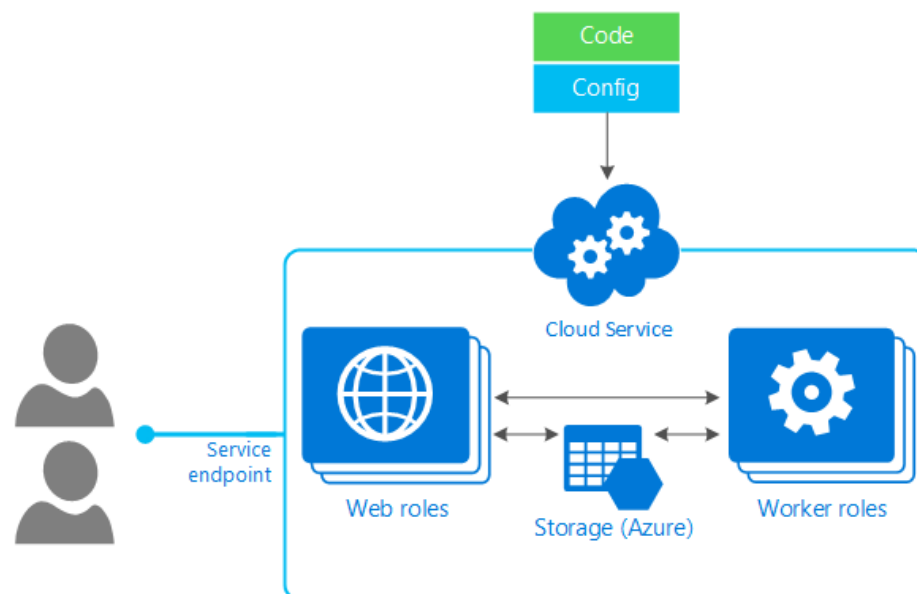


Figure 3. Image source: <https://azure.microsoft.com/en-us/documentation/articles/cloud-services-choose-me/>

Within Cloud Services, there are two types of roles: web roles, and worker roles. A VM created for a web role runs Windows Server with the app automatically deployed to Internet Information Services (IIS), Microsoft's web server. A VM created for a worker role runs Windows Server without IIS installed, as it is not needed. The worker role behaves similar to a Windows Service installed on a server.

With Cloud Services, users do not directly create VMs. Instead, users provide configuration options telling Azure how many of each type of VM they need. Users also choose what size (allocated resources) the VMs are. Like the App Services, Cloud Services support autoscaling, enabling roles to scale up and add resources as needed to support the application.

4. Azure vs AWS

Azure and AWS offer very similar products and services; however, they tend to approach each service slightly differently. As a result, both have their strengths and weaknesses. Despite these, Azure and AWS both offer a very wide range of products and services, with powerful management tools and interfaces that make it easy to administer and manage their services.

Amazon breaks AWS down into four main categories: Compute, Storage and Content Delivery, Databases, and Networking. All of these operate under Amazon's extensive admin controls, which include identity management, auditing, encryption key creation/control/storage, monitoring and logging, and more (Van Winkle, 2015). AWS also has powerful analytics services, such as Amazon EMR (AWS' Hadoop framework), and many application and deployment services.

Like AWS, Azure is broken down into four categories: build infrastructure, develop modern applications, gain insights from data, and manage identity and access. Azure also has its own Hadoop implementation, HDInsight, and Apache Storm for real-time stream processing (Van Winkle, 2015). Azure's comparable services all fit into these categories, and compare to AWS' offerings.

One of Azure's advantages over AWS is the Hybrid Cloud Approach. A hybrid cloud allows companies and users to use both cloud and on-premise resources. This can make a huge difference in terms of application hosting and data governance. Many companies have contractual or legal requirements that prevent them from moving to a cloud provider, and the hybrid cloud helps resolve such issues. Azure provides a simple VPN tunnel to connect on-prem and cloud systems, as well as tools to combine on-prem and cloud resources. For example, Azure has a tool to load balance a web application across a cloud host and an on-prem host.

4.1 Government Cloud

A major issue for government agencies when moving to the cloud is data governance. Most government agencies are legally unable to move their data up into the cloud due to the data location. For example, Canadian government agencies with sensitive national data cannot move their data off Canadian soil. Amazon has an advantage here, as they are currently opening up a

datacentre in Montreal, giving Canadian agencies an option to keep data in Canada. Microsoft is in early plans to open a Vancouver-area datacentre, however it's still in early planning.

Another issue for government systems is security. To address this, Amazon has implemented the AWS GovCloud. GovCloud is “an isolated AWS region designed to allow U.S government agencies and customers to move sensitive workloads into the cloud by addressing their specific regulatory and compliance requirements” (Amazon Web Services Inc, 2016). GovCloud is certified as being compliant with the International Traffic in Arms Regulations (ITAR), which regulates security measures for defence-related data. GovCloud is also certified for regulations such as HIPAA, SOC 1-3, ISO, 27001, etc.

Amazon has a lead in the government cloud space, but Microsoft has joined in as well. Azure just brought its government cloud service out of preview. While the full list of compliances is not available yet, Azure Government is already attracting government agencies who already use the Microsoft stack in their existing systems.

4.2 Enterprise Services

Azure offers a strong enterprise suite of services, especially for companies already invested in the Microsoft stack for their day-to-day operations. A major component to this is Azure Active Directory. Many companies run on Windows, Office, and Active Directory. Azure Active Directory is designed to work in a hybrid mode with companies' on-prem Active Directory servers, making it very easy to integrate with. This also offers strong single-sign-on (SSO) options for all existing on-prem applications and Azure-hosted applications. This is also designed to integrate directly with Office 365, thus making it easy to have one central identity management system. Therefore, for a company already using Microsoft's infrastructure, Azure is an easy migration and integration.

For companies more invested in a Linux-based backend, AWS offers a stronger and easier integration infrastructure. While Azure offers Linux-based VMs and service backends, typically based on Ubuntu, AWS is more heavily Linux-based for its services and virtualization. AWS' biggest advantage here is its support for Red Hat Enterprise Linux (RHEL). Red Hat is the largest provider of enterprise-level Linux servers and systems, making them an important cloud partner for any provider. Microsoft and Red Hat have a disagreeable past, and therefore Azure does not have support for Red Hat servers. This makes AWS a better choice for companies with Linux environments (Van Winkle, 2015).

5. The Project

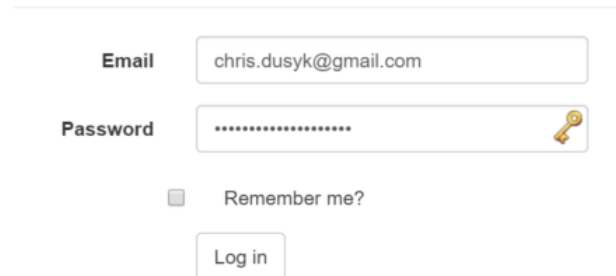
This project focuses on combining different services offered by the two largest cloud providers, namely Microsoft Azure and AWS. The project uses a web application hosted in Azure's App Service, a message queue hosted in Azure Service Bus, a worker service hosted in Azure's Cloud Service, and an Aurora database hosted in Amazon's Relational Database Service (RDS). All

source code for the project can be found on GitHub at <https://github.com/LordCheese/cs401-chrisdusyk>.

The project simulates a basic ordering system. Users are required to log into the application:

Log in.

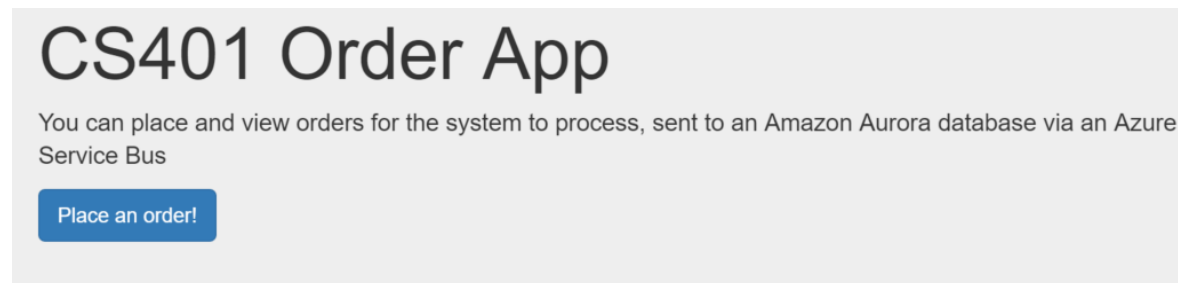
Use a local account to log in.



A login form with two input fields: 'Email' containing 'chris.dusyk@gmail.com' and 'Password' containing a masked password '*****'. To the right of the password field is a key icon. Below the password field is a checkbox labeled 'Remember me?'. At the bottom is a 'Log in' button.

Figure 4

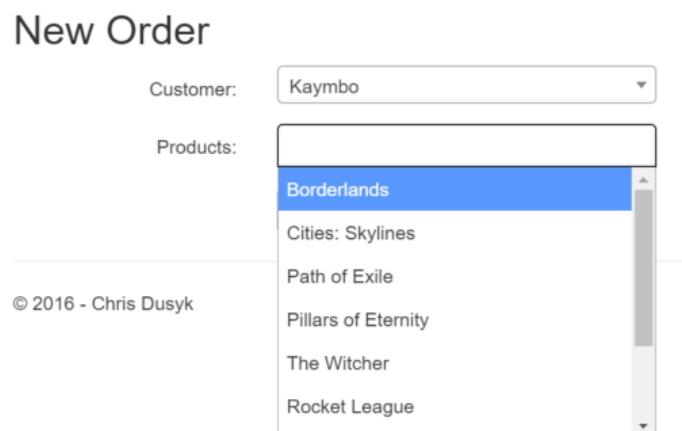
Once authenticated, they can create an order by clicking on the “Place an order!” button:



A landing page for the 'CS401 Order App'. It features a large title 'CS401 Order App' and a subtitle 'You can place and view orders for the system to process, sent to an Amazon Aurora database via an Azure Service Bus'. Below the subtitle is a blue button labeled 'Place an order!'.

Figure 5

This will take the user to the order screen. From here, they can select the customer and products:

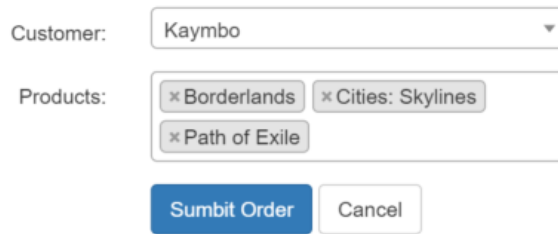


A 'New Order' form. It has a 'Customer:' label and a dropdown menu showing 'Kaymbo'. Below it is a 'Products:' label and a list box containing several game titles: 'Borderlands', 'Cities: Skylines', 'Path of Exile', 'Pillars of Eternity', 'The Witcher', and 'Rocket League'. The 'Borderlands' item is currently selected and highlighted in blue. At the bottom left of the form is the copyright notice '© 2016 - Chris Dusyk'.

Figure 6

Once the products are chosen, the user clicks on “Submit Order” to submit the order:

New Order



Customer: Kaymbo

Products: x Borderlands x Cities: Skylines x Path of Exile

Submit Order Cancel

Figure 7

This creates the order object, serializes it to a JSON dictionary, and sends it to the Service Bus queue for processing. The order sits in the queue until the Cloud Services worker picks it up for processing. The Cloud Services worker picks up the message from the queue, marks it as received, and then deserializes it from the JSON dictionary. The deserialized object is then parsed and inserted into the Aurora database.

5.1 Design

The application is designed to follow service-tiered architecture principles. That is, a front-end system accepts user input and pushes that down a service layer to a background handler. The background handler accepts the input, processes it as necessary, and then puts it in storage (i.e. a database). In this case, the web application is the front-end that accepts user input, the Azure Service Bus is the service layer, and the Cloud Services worker is the backend. An Amazon Aurora database, hosted in Amazon RDS.

This architecture was chosen as it is well-suited to building cloud applications. While cloud applications can be easily built using a more common 3-tier application structure, the service-tier architecture does a good job of taking advantage of the technological advantages native to cloud systems.

In my workplace, this is how we have been designing our current API services layer in Azure, and it presented itself as a good fit for this project. Using the Azure Service Bus queue helps mitigate potential database load and data loss, which is a huge advantage for data-driven business applications. The entire architecture is based on Microsoft’s recommended cloud approach, as is taught in their architect certifications.

Figure 8 below outlines the architecture used as a diagram. Data is entered into the CS401WebAppComponent, reachable at <http://cs401webappcomponent.azurewebsites.net/>. Once the user submits an order, it flows from the web application, into the Service Bus queue, and then to the CS401ServiceBusService worker, which processes it and inserts it into the Aurora database in Amazon RDS.

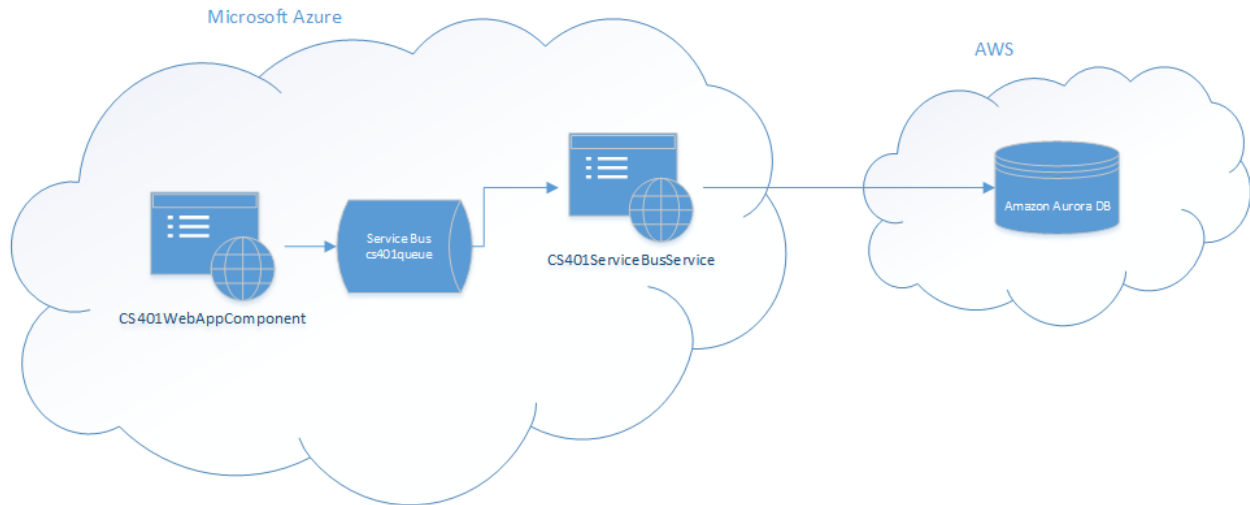


Figure 8

5.2 Data

The data model is a simple relational model, as seen in Figure 9. Figure 9 is an ERD generated by Entity Framework in Visual Studio 2015, showing the data layout and relationships.

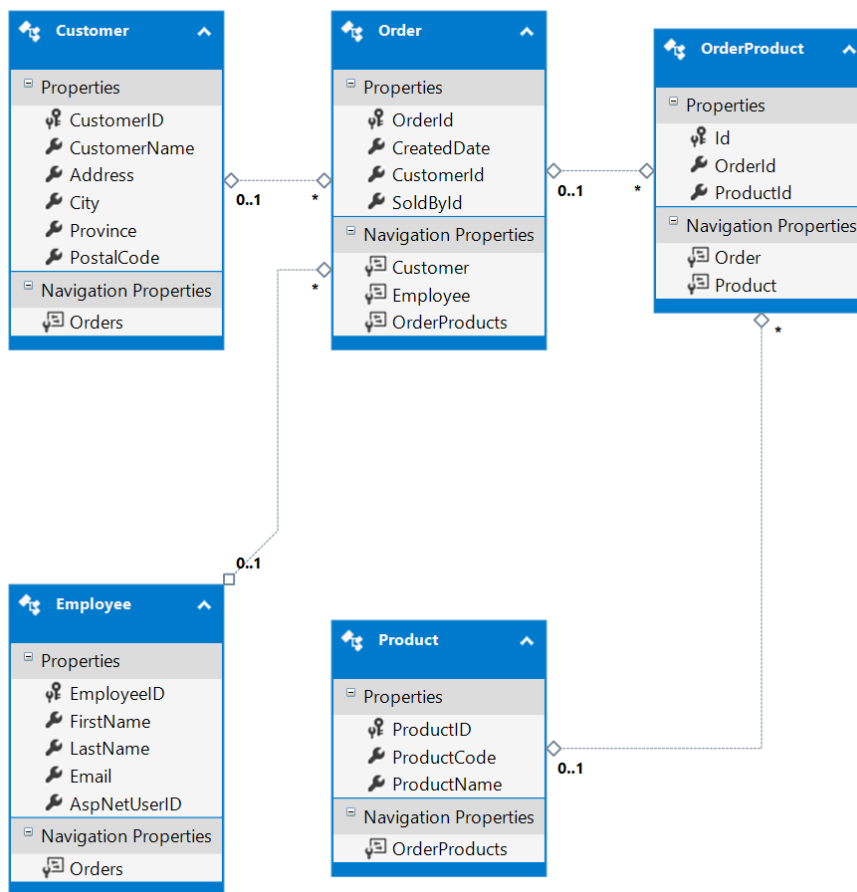


Figure 9. Image generated from Visual Studio 2015

An Order is the core table that describes an order item. And Order can have many OrderProducts, describing the products chosen in an order. Each OrderProduct relates back to the Products table for the product information. Each Order relates to a Customer and an Employee, through the CustomerId and SoldById respectively.

5.2.1 Data Access

Data access is handled through Microsoft's Entity Framework ORM. Entity Framework directly maps class objects in C# code to database tables, views, and procedures. It directly handles the selection, insertion, updating, and deletion of data, removing the need to write SQL code. For example, in the Cloud Services worker, the Order and OrderProducts are extracted from the Service Bus queue object and inserted into the database as per Figure 10.

```
// Create the database connection and context
// Since it's in a using-block, it will automatically dispose of the connection when it exits the block
using (CS401_DBEntities1 db = new CS401_DBEntities1())
{
    var order = packagedOrder.Order;

    // Add the order to the db context
    db.Orders.Add(order);

    // Insert order to get an OrderID
    db.SaveChanges();

    foreach (var orderProduct in packagedOrder.OrderProducts)
    {
        orderProduct.OrderId = order.OrderId;
        db.OrderProducts.Add(orderProduct);
    }

    // Insert OrderProducts into database now
    db.SaveChanges();
}
```

Figure 10

Amazon Aurora uses a fork of MySQL as its top-level structure, so it's necessary to use the MySQL Connector/.NET to connect Entity Framework to the MySQL variant structure. The MySQL connector can be obtained from the MySQL official developer site, at <https://dev.mysql.com/downloads/connector/net/6.9.html>. To make use of the connector, the MySql.Data and MySql.Data.EF6 DLLs must be added to the .NET project.

5.3 Web Application

The web application component, hosted as part of the Azure App Service, is an ASP.NET 4.6.1 web application. The programming language is C# and the framework is ASP.NET, both Microsoft technologies. Figure 11 shows the basic setup in Azure:

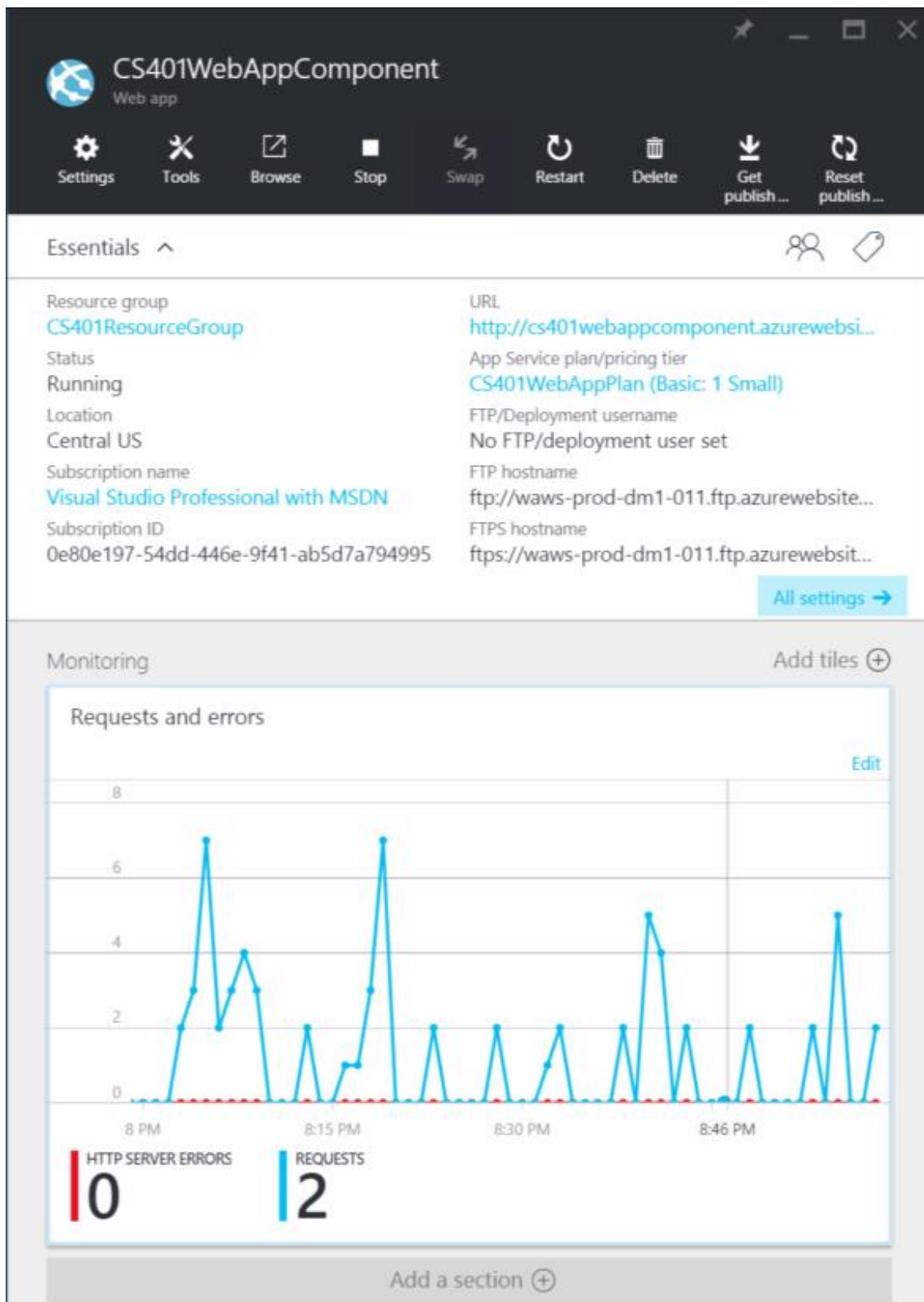


Figure 11

5.4 Cloud Services Worker

The Cloud Services worker role is a .NET 4.5 worker service hosted in Azure's Cloud Services feature. The worker service is hosted on a small VM running Windows Server 2012 R2. The VM can be connected to and managed using Telnet, PowerShell, or Remote Desktop, as it is a fully functional server that has been imaged with the software needed for the worker service to run. Figure 12 shows the basic setup in Azure:

The screenshot displays the Azure portal interface for a Cloud Service (Classic). The top navigation bar includes icons for Settings, Production slot, Update, Stop, Swap, and Delete. The main content area is divided into two sections: 'Essentials' and 'Roles and instances'.

Essentials

Property	Value
Resource group	cs401servicebusservice
Status	Running
Location	Central US
Subscription name	Visual Studio Professional with MSDN
Subscription ID	0e80e197-54dd-446e-9f41-ab5d7a794995
Site URL	http://cs401servicebusservice.cloudapp.net/
Public IP addresses	40.86.88.35
Deployment name	a16184f5c1ba4303931f418873edb31b
Deployment label	AzureApiServiceBus - 2016-04-10 11:53:46 ...
Deployment ID	5bc264ca48024583b5865fa67d228e60

[All settings →](#)

Roles and instances

NAME	STATUS	SIZE	UPDATE	FAULT
▼ OrderServiceBus				
OrderServiceBus_IN_0	✓ Running	Small	0	0

Figure 12

5.5 Testing

To enable the marker to test this application, I have set up a test user to place orders with:

- User: gerhard@cs.uregina.ca
- Pass: #6oqL?ws?3okLrf3AOM,

5.6 Successes and Failures

There were a few missteps and reworks through this project. To start with, I attempted to put both the web application and the worker service together in the same Cloud Services, however I found this difficult to work with. I had some deployment issues with the MySQL Connector when doing this that I was able to solve much later when I moved on from putting the web application and the worker service in the same container.

Once I separated the web application and the worker service, I found the separated apps easier to work with. It did require adding the MySql.Data and MySql.Data.EF6 DLLs to both projects, but this was minimal effort.

An advantage to ASP.NET MVC framework is that it uses Bootstrap templates for its UI styling by default, making it very easy to style and build the UI. This part went very quickly and easily, which is very convenient as I am not a strong UI/UX designer.

Once I got into really building the pieces, it was very easy and painless. Microsoft's Azure Documentation site was extremely helpful and I was able to put together working code very quickly. This was helped by how ASP.NET MVC scaffolds its controllers and views for you, based on a ViewModel you can define, which saved a lot of time and effort.

I did run into some issues early on with the MySQL Connector. After some effort I discovered the App Service and Cloud Service had a version of the MySQL DLLs installed on their hosting VMs, which was having compatibility problems with the version of the DLLs I was using. However, I was able to get around this by adding some code to the config files for the two projects, as seen in Figure 13.

```
<system.data>
  <DbProviderFactories>
    <!-- Remove installed MySql.Data library from Azure container -->
    <remove invariant="MySql.Data.MySqlClient" />
    <!-- Replace with MySql.Data DLL from app bin folder -->
    <add name="MySQL" description="ADO.Net driver for MySQL" invariant="MySql.Data.MySqlClient" type="MySql.Data.MySqlClient.MySqlClientFactory, MySql.Data" />
  </DbProviderFactories>
</system.data>
```

Figure 13

After solving that issue, everything else flowed very easily. Modern versions of ASP.NET (especially MVC), Visual Studio, and Azure work very efficiently together. It is very easy to publish code from Visual Studio straight to Azure. Visual Studio also offers excellent tooling to manage Azure products from Visual Studio, though Azure's web portal also works very well. My only issue with the Azure Portal is that Microsoft still hasn't moved everything from the Legacy

portal into the new portal, namely the Service Bus pieces. The Service Bus queues and namespaces still need to be managed in the Legacy portal, which is inconvenient.

Aside from the above however, getting the code running and deployed was incredibly easy and painless. I'm very glad I went with Azure and AWS for this project as they were easy to work with, incredibly powerful to use, and painless to deploy. As well, both Azure and AWS are heavily used in many technology sectors across the world, so there are many resources available to answer any questions or issues I ran into. Both providers have very active communities and excellent support documentation. I would consider working with either of these service providers and frameworks to be a path to success for any project.

6. Conclusion

This project makes use of several services offered by Microsoft Azure and Amazon Web Services, exploring some of the features and functionality these cloud providers can offer. Both proved to be very effective and powerful, while being easy to work with. Cloud providers can provide a huge cost savings in terms of hardware and maintenance to businesses, while giving them effective tools to build good software and infrastructure. There are many reasons to choose either provider over the other, and it ultimately comes down to the technology stack being used and the design needs in the project.

In this project, I found both providers to be extremely easy to work with and deploy to. Given that I am already a .NET developer in my workplace, I had a natural bias towards the ASP.NET framework and Microsoft Azure, however AWS' comparable services are very well respected and would have worked for my needs as well.

One of the biggest advantages to using these providers is how well documented and used they are. Both providers have excellent documentation on their dedicated documentation websites. There are also large communities of IT professionals who use both providers' services, and can help with any issues encountered by others.

The main issue with this project is its performance when going from the web application to the Aurora database, however this could have been mitigated by setting up a tunnel between the database virtual container and the Azure resource group. This was not done due to time and cost constraints for the scope of the project.

In conclusion, I would strongly recommend using either Azure or AWS (or potentially both depending on needs and costs). Cloud technology is remarkably powerful, and is becoming more mature every day. It is the perfect direction to go for many companies, particularly small to medium businesses who can save on the costs of running their own servers and infrastructure.

Works Cited

- Amazon Aurora – MySQL Compatible Relational Database*. (n.d.). Retrieved from Amazon Web Services Inc: <https://aws.amazon.com/rds/aurora/>
- Amazon Web Services Inc. (2016, March 24). *What is AWS GovCloud?* Retrieved from Amazon Web Services : <http://docs.aws.amazon.com/govcloud-us/latest/UserGuide/whatis.html>
- AWS | Amazon RDS | Pricing*. (n.d.). Retrieved from Amazon Web Services Inc: <https://aws.amazon.com/rds/aurora/pricing/>
- Chappell, D. (2014). *Adopting Microsoft Azure: A Guide for IT Leaders*. Retrieved from Microsoft Azure Documentation: <https://azureinfo.microsoft.com/rs/microsoftdemandcenter/images/EN-CNTNT-AdoptingMicrosoftAzure.pdf>
- De George, A. (2015, January 9). *Should I Choose Cloud Services?* Retrieved from Microsoft Azure Documentation: <https://azure.microsoft.com/en-us/documentation/articles/cloud-services-choose-me/>
- Dykstra, T. (2014, October 27). *Azure Cloud Service Multi-Tier Application with Tables, Queues, and Blobs*. Retrieved from MSDN Code: <https://code.msdn.microsoft.com/Windows-Azure-Multi-Tier-eadceb36>
- Espinosa, J. (2016, January 9). *Web Apps overview*. Retrieved from Microsoft Azure Documentation: <https://azure.microsoft.com/en-us/documentation/articles/app-service-web-overview/>
- Everything You Need to Know About Azure Service Bus Brokered Messaging (Part 1)*. (2015, January). Retrieved from Lock Me Down - Security for the Everyday Developer: <http://lockmedown.com/be-sure-with-azure-net-azure-service-bus-part-1/>
- Lin, C. (2016, March 9). *Set up staging environments for web apps in Azure App Service*. Retrieved from Microsoft Azure Documentation: <https://azure.microsoft.com/en-us/documentation/articles/web-sites-staged-publishing/>
- Manheim, S. (2015, October 07). *.NET multi-tier application using Azure Service Bus queues*. Retrieved from <https://azure.microsoft.com/en-us/documentation/articles/service-bus-dotnet-multi-tier-app-using-service-bus-queues/>
- Manheim, S. (2016, March 9). *Azure Service Bus*. Retrieved from Microsoft Azure Documentation: <https://azure.microsoft.com/en-gb/documentation/articles/service-bus-fundamentals-hybrid-solutions/>
- Manheim, S. (2016, January 26). *How to use Service Bus queues*. Retrieved from Microsoft Azure Documentation: <https://azure.microsoft.com/en-us/documentation/articles/service-bus-dotnet-how-to-use-queues/>

- Mohammed, A. (2009, March). *A History of Cloud Computing*. Retrieved April 11, 2015, from Computer Weekly: <http://www.computerweekly.com/feature/A-history-of-cloud-computing>
- MySQL Team. (n.d.). *Download Connector/Net*. Retrieved from MySQL Developer Site: <https://dev.mysql.com/downloads/connector/net/6.9.html>
- Narasayya, V., Das, S., Syamala, M., Chaudhuri, S., Li, F., & Park, H. (2013). A demonstration of SQLVM. *Proceedings of the 2013 international conference on Management of data - SIGMOD '13*. Retrieved from http://www.cidrdb.org/cidr2013/Papers/CIDR13_Paper25.pdf
- Ohlinger, M. (2016, February 29). *Hybrid Connections overview*. Retrieved from Microsoft Azure Documentation: <https://azure.microsoft.com/en-us/documentation/articles/integration-hybrid-connection-overview/>
- Van Winkle, W. (2015, January 31). *Microsoft Azure vs. Amazon Web Services: Cloud Comparison - Azure and AWS Compared*. Retrieved from Tom's IT Pro: <http://www.tomsitpro.com/articles/azure-vs-aws-cloud-comparison,2-870.html>