Options for Migrating Web Apps to Azure

Motivations

Before we can really start talking about any kind of migration, we need to take some time and consider the motivations for moving to the cloud. There are a wide array of motivations that fall into various categories, but the important part is to have a clear understanding of what problems we are trying to solve, what the desired outcomes are, and how to measure success. I'm not going to dig into cloud adoption strategy, but if that is something you are interested in or need, Microsoft has put together an extensive set of documentation called the Cloud Adoption Framework. Now, it is some dense stuff, and RBA offers this type of strategy, and we would be happy to guide you through it if needed.

The Biggest Cloud Myth

I think it is prudent to start with addressing one of the biggest, and most persistent, cloud myths out there. That is: the cloud is inherently cheaper and that an organization will save money by moving to the cloud. That is just plain wrong. Now, everything is situational, and it is possible that an organization can realize cost savings, but I feel comfortable saying you probably will not save money by moving to the cloud.

Obviously, cost is important, and to be fair, most businesses will not support a move to the cloud if the costs are significantly higher, so it is something we must keep in mind. It is just super complex and hard to get right up front.

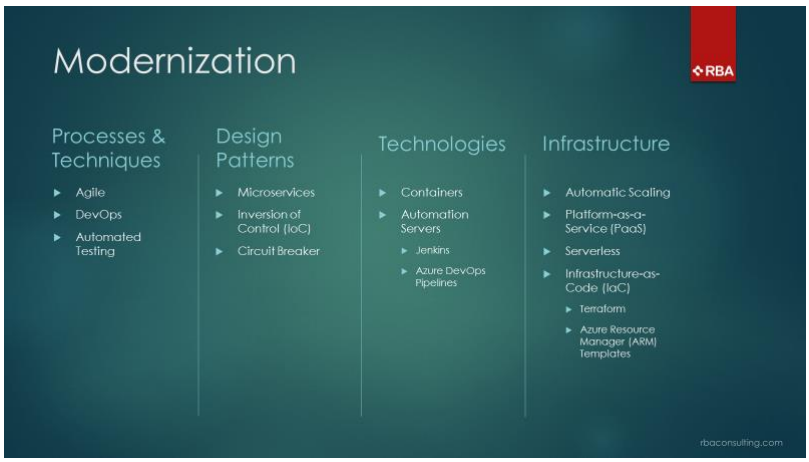
Here is a recent example. Last year (2019), NASA selected AWS to run their Earthdata Cloud where they expect to be hosting around 250 petabytes of data by 2025. One of the main use cases for this data is for people to download large quantities it. What do you think NASA forgot to factor into their cost estimates? Downloading the data, or egress cost. That is data leaving the cloud platform. Azure and Google Cloud have the same type of thing. It is free to get data onto the cloud platform, but there is a charge for getting out. Considering that downloading is one of the main use cases for this data, it seems like an obvious mistake, but the moral of the story, to me, is that anyone can get it wrong.

So, if we are probably not going to save money, why move to the cloud? Really, modernization should be one of the main objectives an organization is trying to achieve nowadays, and cloud services are one of the tools in the toolbox.

Modernization

Let me take a minute here and talk about what modernization is and why it is important.

My definition of modernization is: Using current processes and techniques, design patterns, technologies, and infrastructure.



Modernization Examples

But, what does that get us? This is my marketing pitch for modernization, "It allows IT to be more nimble to business needs and strategy which increases our value to the business." That is really why IT exists. To satisfy business needs and strategy.

And then for the business:
Modernization means reducing time to market, and enabling an organization to be more effective, efficient, and agile.

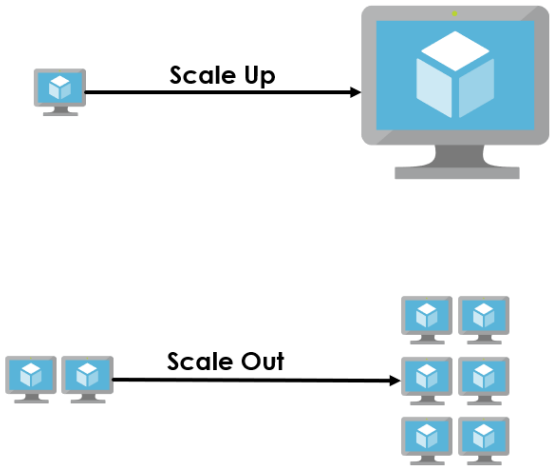
Azure

So, where does Azure fit into that? It is primarily the physical infrastructure, but it brings other features that facilitate the modernization effort as well as physical security. This is an area where there could be a big difference between having hardware on-premises or at a collocation (colo). Most colo's probably have the physical security thing buttoned down but doing it on your own can be hard. Especially if your organization needs to adhere to some certification like SOX, PCI, HIPPA, or a variety of the ISO standards. Azure satisfies all those standards and a ton more.

Rather than looking at absolute cost saving, it is typically better to look at the return on investment (ROI) and business value, including any of those value added, modernization features you get when moving to Azure.

Autoscaling

For me, one of the best features in Azure is automatic scaling. There are two types of scaling. Scaling up and scaling out, sometimes also referred to as vertical and horizontal scaling. Scaling up is adding more resources to an existing system. For example, adding more RAM or more CPU to an existing virtual machine. Scaling out is adding more of a resource, like adding another virtual machine. This is an important feature for usability, but especially important regarding cost management. Provision only as much as needed, when needed, and deprovision when not.



Scale Up vs Scale Out

The general guideline, for a while now, has been to scale out whenever possible, and the cloud was built around that philosophy. So, typically it is much more expensive to scale up in the cloud than it is to scale out. So, how does the elastic scaling nature of Azure help with cost management? When you own your own hardware, you pretty much have two options. Either buy enough servers, or big enough servers, to be able to handle your peak capacity, or accept running with degraded performance at peak load, and that doesn't even consider whatever redundancy model you have to have in place.

Most companies have predictable, but variable, usage pattern. For example, one company I worked for had a US based, workday usage pattern. Every morning traffic would pick up to something relatively steady throughout the day. Then in the evenings the traffic would slowly decrease and be minimal overnight. Weekends were even slower than overnight, but there was almost always some traffic. There were not any huge spikes, but definitely enough variance to warrant less resources overnight and weekends which, and if really you think about it, accounted for about 75% of the time. We had to have enough horsepower to handle that regular daily load, and that left us over-provisioned overnight and weekends.

That was a daily usage pattern, but what about something like tax preparation services? Something more seasonal? It is easy to see where most of the year usage is low but starting in January through the middle of April there is consistently higher traffic.

Ticketmaster is another good example? They have semi-unpredictable traffic spikes whenever venues open sales for whatever the new concert is nowadays, and then the rest of time they must maintain some, relatively minimal, baseline.

How about a bonus Star Wars example? My family took a trip to Disneyland earlier this year (2020), and, being a big Star Wars fan, I was super excited to hit up Galaxy's Edge (Star Wars land). What I didn't know is, they had just opened a new ride, Rise of the Resistance, a week before our trip. This isn't set up like a normal amusement park ride where you just get in line and wait. Instead you sign up for a Boarding Group through the Disneyland app starting at 8

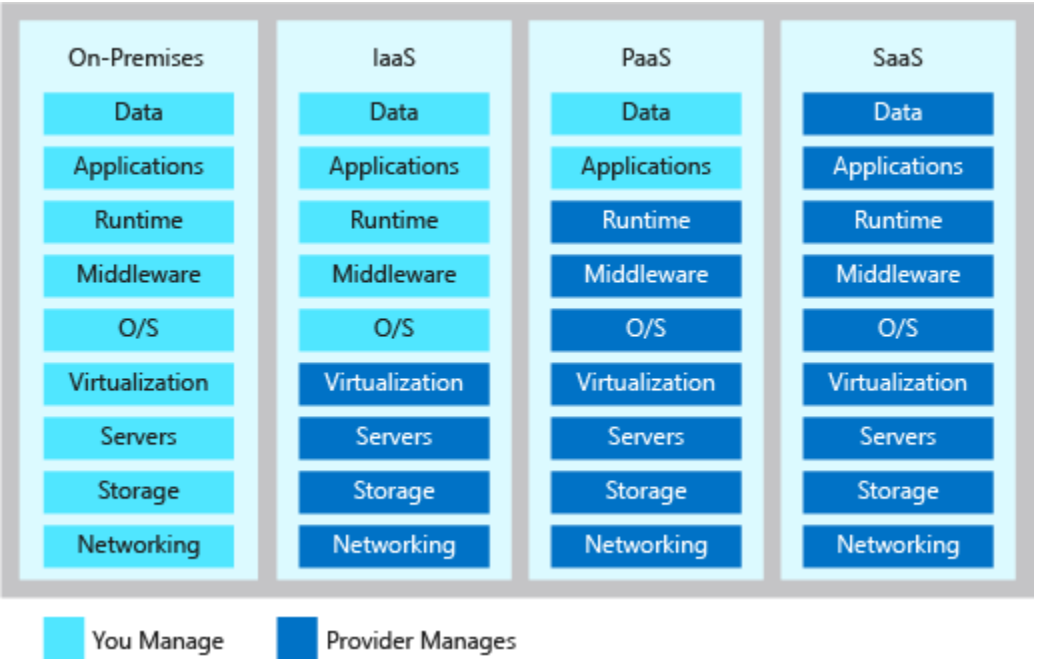
AM. There is a limited number of people that are able to get onto the ride in a day, so what you have is thousands of people getting to the park early to try and click a button right at 8 AM, and all the Boarding Groups, and the standby groups, are filled within minutes so they close down the Boarding Group request processes until the next morning. Then the rest of the day you can track which Boarding Groups are currently boarding through the app. The usage here is a bit more predictable with a huge spike in the morning, and then people checking throughout the day to see what Boarding Groups are currently boarding and if they are going to make it onto the ride or not[1].

Back to scaling, with Azure and automatic scaling, you can create rules to scale out to be able to handle peak capacity when needed, and scale in when the resources are not needed. Those rules can be specific date/times or metric based, for example CPU or RAM percentage. Remember, in the cloud, you pay for what you are using when you are using it. That means different things depending on the resource, but the goal here is to run only what you need to run when you need to run it. That is how you get optimal usability during peak loads but can also minimize cost. Compare that again to running your own hardware where you need to purchase enough resources to handle peak loads or be willing to compromise performance. That generally means that either your hardware is underutilized most of the time, or your users are affected, and a lot of them since we are talking about when the system is under load. Unfortunately, a lot of companies opt for accepting the degraded performance during peak loads.

[1] This is actually a perfect use-case for serverless.

Reduced System Administration

Let's take a look at Microsoft's Cloud Responsibility Model.



(https://docs.microsoft.com/en-us/learn/modules/pillars-of-a-great-azure-architecture/media/cloud-responsibility-model.png)

Another main benefit to moving to the cloud is a reduction of overall system administration. At a minimum, you no longer need to deal with the physical aspects of the infrastructure, and in some cases, Azure will take care of the underlying software maintenance. Ideally this means you can focus more of your effort on your organization's core business, and less on what it runs on.

As can be seen, as we move from On-Prem, to Infrastructure-as-a-Service (IaaS), to Platform-as-a-Service (PaaS), an organization has less responsibility over the infrastructure. Some things I like to keep in mind here, the items on the bottom of the diagram are the physical infrastructure. "Networking", "Storage", and "Servers" on the diagram are the physical hardware and networking cables connecting them. Microsoft takes care of the physical networking, but you would still need to manage the virtual network, subnets, firewalls, and things of that nature. Then once we get to "Virtualization" that is the physical VM hosts as well as the virtualization software, something like Hyper-V.

As we move to the right from Infrastructure-as-a-Service to Platform-as-a-Service, the provider responsibility gets more software based, think about it as Microsoft now keeps the OS up to date and, in some cases, the underlying networking. For example, App Services has a built-in load balancer, and you don't have to do anything to set that up. It's just there and works. Now, it's a pretty simple load balancer and is just for routing traffic to a pool of servers, not for advanced functionality like SSL termination or Web Application Firewalls. It is pretty easy to think that less system administration translates to direct cost reduction, but it's more nuanced

than that and isn't always the case. A lot of that cost is shifted to the cloud provider and is bundled into the pricing. The cloud provider should be able to provide those services for a lower cost due to economies of scale.

Compute Migration Options

When it comes to migrating an existing application, we really have two main options if we aren't going to completely rebuild it. Infrastructure-as-a-Service, or Platform-as-a-Service.

Infrastructure-as-a-Service

Let's start with Infrastructure-as-a-Service. Here we have two main options as well. Virtual Machines, and Virtual Machine Scale Sets (VMSS).

Virtual Machines

The first is regular old virtual machines. Straight up lift and shift. What I'm talking about here is replicating your on-prem VMs or physical servers into the cloud. From a change perspective, this may be the easiest option. If an organization wants to go down this route, Microsoft has a tool, Azure Migrate, specifically designed for assessing your environment, making recommendations, testing, and then performing the migration into Azure.

In general, I don't recommend this option, but it can make sense when an organization is running on old hardware and there is some urgency to the migration. I've been part of these, and the biggest piece of advice I have in this scenario is to concentrate on right-sizing the servers. Don't just copy whatever you have on-prem.

Virtual Machine Scale Sets

The other option in IaaS is Virtual Machine Scale Sets. This option is a little more appealing to me because now you get automated scaling, but it does change the deployment. To be fair, just about anything you do when migrating to the cloud is going to change the deployment process, but with VMSS it feels a little more extreme to me.

Let's consider what needs to be done to automatically scale a set of virtual machines. To add an instance, a whole new VM needs to be spun up. There are a couple ways to do this, but one that is built into Azure DevOps is creating virtual hard disks (VHD) for the deployment. That is a VM disk with everything already installed and ready to go, including the app. It's not like using MSDeploy, Octopus, xcopy or even Copy and Paste where the deployment artifact is your app. The deployment artifact is a full VHD. For Azure DevOps to automate this, it creates a new virtual machine with whatever OS you select. Installs all the prerequisites and the application onto the VM, from a deployment script that you write. Creates a virtual hard disk and stores it in an Azure storage account, and then deletes the virtual machine used in the creation process. The other way is to start with a virtual machine template (i.e. ARM template), and during the scaling operation itself to install any dependencies and the app from a deployment script. I'm not saying either of these are bad, it's just very different than what most of us are used to. Like a lot of things when moving to the cloud though, if you need it, you need it.

IaaS Summary

With these Infrastructure-as-a-Service options, you no longer have to manage and maintain the physical aspects of the infrastructure or the virtualization software. To me, the more you can offload to the provider the better, so what are some other reasons an organization might choose this option?

I already mentioned if an enterprise is at end of life of owned hardware and has some urgency for getting off it. I ran into this once where some relatively expensive hardware appliances were reaching the end of support, so we had a short timeline to decide whether to replace them or make a move to the cloud.

Another reason you may need to go the Infrastructure-as-a-Service route is if you require full access to the VM or specific OS customizations. Once we move to PaaS, Microsoft starts managing the OS, and that means that we lose some control. Another aspect of this is if your software relies on installed 3rd party software. I don't know if anyone does this anymore, but I remember working on some software that needed to perform some image manipulation. Rather than write the code ourselves, we called off to some other software installed on the machine.

Another reason you may want to stick with VMs is you have more flexibility in the hardware configuration. Specifically, they can get Much. Larger. Remember, the cloud is built around scaling out, if you need more resources add more servers. So, when would scaling up make more sense? At RBA, we had a client moving their high-volume CMS site to Azure. Initially we were planning on using a PaaS option, but once we started looking at the CMS licensing costs, we had to reconsider. It was way more cost effective to pay for larger VMs than it was to pay for more CMS licenses, so we ended up going with Virtual Machine Scale Sets.

Another reason an organization might stick with VMs is isolation. Once we start moving towards PaaS, the underlying resources are shared, so, if you need isolation, VMs are an option. App Services has an isolation option too, so it isn't a hard line like OS customization is, but App Services isolation can get expensive.

Platform-as-a-Service

Moving on to Platform-as-a-Service. This is where things start to get interesting to me. In PaaS, we have a further reduction of infrastructure responsibilities covering the OS, middleware, and runtime. Again, this basically means Microsoft takes care of patching of the VMs. That leaves just the applications and data to the organization.

The main options for migrating a web app to Azure are Azure App Services and Containers.

Azure App Services

To me, App Services is the first stage of evolution of Platform-as-a-Service, and at its core, App Services is just another layer of abstraction on top of the VM. In addition to Microsoft managing the OS and runtime, you get autoscaling and a built-in load balancer out of the box. Assuming you select a production level workload, you also get one of my other favorite features, deployment slots, which gives you the ability to do Blue-Green deployments.

Blue-Green Deployments

With Blue-Green deployments, you have two identical production environments. One is servicing the live production traffic, and the other is a pre-production environment. Now, when the next release is ready for production, it gets deployed to the pre-prod environment and any final smoke testing or user acceptance testing can be performed there. Then, when everything is considered ready, prod and pre-prod are swapped with the push of a button. This makes the deployment to production, nearly instantaneous with the added benefit of having rapid rollback capability built in. If something goes sideways with the new release, simply swap back to the old prod environment, and figure out what went wrong.

The primary requirement for Blue/Green deployments is having two identical, production level environments. That alone makes Blue-Green deployments out of reach for many businesses because of both cost and management. That is no longer a barrier with App Services.

App Services Limitations

From a CI/CD perspective, App Services follows a more traditional approach, but there are limitations when moving to App Services though. Remember, this is an abstraction on top of the OS, so it limits access to the VM's underlying resources. Some of the common limitations are:

- No access to assemblies in the Global Assembly Cache or GAC
- Saving files to disk is not allowed
- Log files can't be written to disk
- Emails can't be sent from within the app

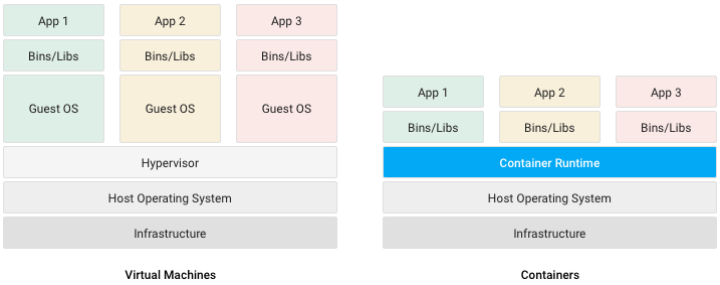
For the most part, these have some relatively easy workarounds:

- GAC: typically, one gets around this by packaging the assemblies with your application
- Saving files to disk: refactor the app to send them to blob storage
- Logs: stream them off somewhere.
- Emails: send them off to a queue and have something else pick them up to process and send

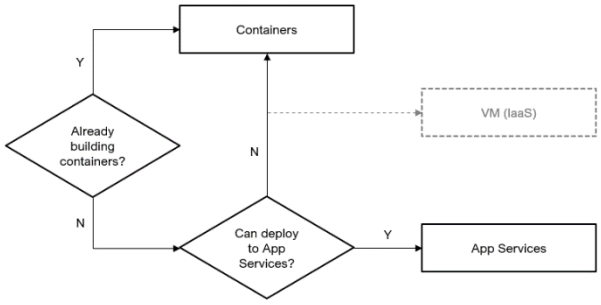
One additional thing to note about deploying applications directly to App Services is that all applications within an App Service are running under a single App Pool. Typically, when I've deployed applications on-prem, each application had its own App Pool to isolate them from each other, and that's just not something that can be done in App Services.

Containers

If deploying a traditional web application to App Services is the first stage of evolution, I consider Containers the next stage.



I'm not going to dig deep into Containers here, but the basic idea is instead of virtualizing the whole hardware stack like with virtual machines, Containers virtualize at the operating system level with multiple containers running on top of the OS. So, the Container images are much smaller, they use resources more efficiently because multiple Containers share the same OS, and because they share the OS, they can be spun up much quicker than a virtual machine. The other main benefits are portability, you can run them just about anywhere, and a consistent environment because the entire runtime is packaged with the application along with all its dependencies, so the application is running in the exact same environment from dev to prod.

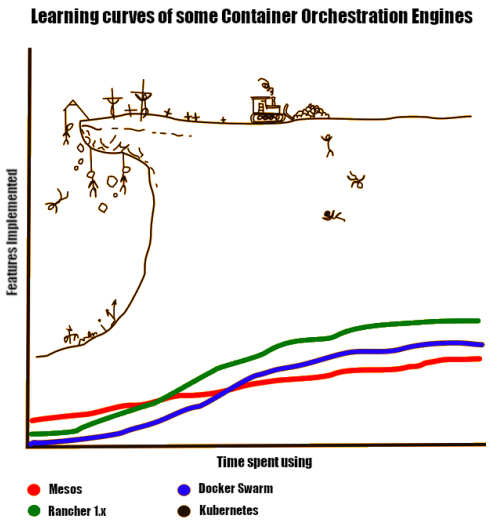


That all sounds great, but there are nuances to creating container images and complexity with the orchestrators they run on. I tend to take a pragmatic approach, so for me it really comes down to whether an organization has already started creating containers or not. If so, that means there should already be some expertise in house and containerization is a viable option for the migration. If the organization hasn't already started creating containers elsewhere, then it comes down to whether the app can be directly deployed to App Services. If so, go with App Services. If not, let's say because it requires OS customization, the options become either IaaS and a VM, or containers. From a modernization standpoint, I'd pick containers whenever possible.

There are technically four options for hosting containers on Azure, but two of them really aren't worth considering. There is Service Fabric, which is pretty much dead[2] at this point, and Azure Container Instances (ACI), which isn't meant for always on containers. It isn't cost effective for that. It's good for things that are spun up and shut down over a short period of time. Things like container image development, quick POCs, batch jobs, or for elastic bursting for CI/CD.

[2] Service fabric is not really dead as much of Azure runs on it, but the industry is moving away from Service Fabric in favor of AKS for container orchestration.

That leaves us with App Services for Containers and Azure Kubernetes Service or AKS. This mostly boils down to one thing. Kubernetes. Is. Complex. So, if your organization has expertise in Kubernetes, jump right in to AKS. Otherwise, deploying it to App Service for Containers is a very good first step while working to build that Kubernetes expertise, because, for what it is worth, when organizations start containerizing, it usually doesn't take very long before they start looking at Kubernetes.



When it comes to containers in App Services, you get the same benefits as when deploying an app directly. With AKS, just like with all the other Platform-as-a-Service offerings, Microsoft handles maintaining the hosts as well as Kubernetes, and you only pay for the nodes your application is deployed and running on (and storage and networking), but not cluster management, including the master node.

Compute Migration Options Summary

Really, those are what I would consider your main compute options when looking at migrating an existing web application to Azure without completely re-architecting or rebuilding it.

For IaaS you have straight up Virtual Machines or Virtual Machine Scale Sets. Virtual Machines are typically the least amount of change from the existing environment and the quickest to migrate, but you don't get any bells and whistles. Virtual Machine Scale Sets add the ability to automatically scale out the Virtual Machines, but you'll have some work to get the autoscaling up and running. With both cases you have full responsibility over the virtual machines and the underlying virtual networking, but you get flexibility in customizing both the size and operating system of the virtual machines as well as no longer needing to maintain the virtualization hosts or software.

For Platform-as-a-Service you have the options of deploying directly to App Services or containerizing the application and deploying to either App Services or AKS. Deploying directly to App Services is most likely closer to the existing environment than containerizing, so should have a lower barrier of entry. Whether you containerize your app or not, running on App Services gives you auto scalability options as well as things like deployment slots. When containerizing your application, deploying to App Services is a good entry point due to the complexity of Kubernetes, but there is a good chance your organization will move towards AKS shortly thereafter.

Database Migration Options

Most applications have some kind of relational database behind them, so what are our options there? Unsurprisingly, the database options are pretty similar to the compute. I'm going to focus on SQL Server here, but there are options if you use something else like MySQL or Postgres.

On the Infrastructure-as-a-Service side you have SQL Server on an Azure Virtual Machine. Then for Platform-as-a-Service, you have a few options. There are Managed Instances, Single Database, and Elastic Pools. Technically you could also import SQL server to CosmosDB, but CosmosDB is a NoSQL database, and that is more than just migrating to Azure, so it's not something I would consider a viable migration option. I mean, it is a viable option if you are looking to refactor your relational database to a NoSQL database, but that's not the discussion here.

SQL Server on an Azure VM

SQL Server on an Azure VM, the IaaS model, is very similar to deploying applications to a VM. You have complete control over the operating system and the SQL Server install, but you also have responsibility over the configuration, patching, backups, and logs. All that good stuff. Again, this option is the closest to your existing environment so it would be the least amount of changes. So, why choose this option? First, if you need that OS customization like before. Also, most of the Platform-as-a-Service options have some limitations. One of the most noticeable is SQL Agent isn't available in the standard PaaS database offering. So, if you rely on SQL Agent and don't want to refactor its functionality into something like Azure Functions, you may need to stick with SQL Server on a VM.

PaaS Database Options

So, the Platform-as-a-Service options. Here you have Managed Instances and Azure SQL. Again, they follow the basic cloud responsibility model where Microsoft maintains the OS and, in this case, also SQL Server. What you gain from moving to the PaaS offerings is built in high availability at 4 9's (99.99%) and automatic backups using read-access geo-redundant storage (RA-GRS), they put it somewhere else, which means if even an entire Azure region goes down, you won't lose your data.

Managed Instances

Technically Managed Instances are Azure SQL, but I tend to call them out specifically because they are a bit different. Managed Instances provide nearly 100% compatibility and full SQL Server access whereas the "normal" SQL Azure has more limitations. For example, SQL Agent is available in Managed Instances. Sounds great, but the downside is that they tend to be expensive, complex, and hard to set up.

Single Database & Elastic Pool

Then we have "normal" Azure SQL. This comes down to the single database and elastic pool options.

Single Databases are best used for relatively predictable usage patterns. They get their own set of dedicated resources that are isolated from each other and can be scaled manually or programmatically. Programmatically here does not mean automatically, it just means through a script. With databases in the cloud, you should always have some kind of retry logic in place, but if you anticipate scaling with any frequency with a single database it is even more important because there is a short switchover time.

The other option is elastic pools. They are best used when you have a collection of databases with varying and unpredictable usage. Elastic pools allow multiple databases to share resources. You add a database to the pool, set the minimum and maximum resources for the database. Then, within the pool the individual databases can auto-scale within the pool's designated resources.

A couple final bits about Azure SQL. A database can be moved in and out of elastic pools, so make your best guess and change it later if the other model is more appropriate. And then, Azure SQL databases get auto-tuning. This is nothing super complex, it's not going to replace your DBAs or writing good stored procedures, it primarily adding and dropping indexes and comparing the current execution plan to the previous, but it's something that helps. Also, Managed Instances only get the execution plan comparison part.

Database Migration Options Summary

When looking to migrate databases to Azure, I'd start with the Data Migration Assistant from Microsoft. It will run compatibility checks looking for migration blocking issues, partially or unsupported features and assist in the actual migration.

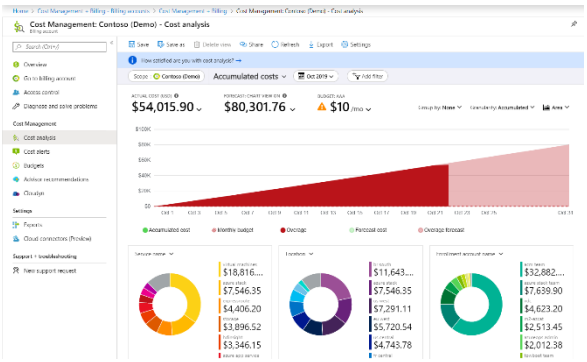
Those are your basic options when it comes to migrating your database to Azure. Generally, I start by seeing if "normal" Azure SQL will work, and then work back to Managed Instances or SQL Server on an Azure VM from there if I have to.

Cost Management

So, cost has come up throughout the discussion, but here are the things I think are most important when looking at cost management. For the most part these are general guidelines, but they most certainly apply to migration scenarios.

- When planning, the Azure Cost Calculator is a good resource, but understand that assessing cost up front is an extremely complex task and hard to actually get right. It works well to be able to compare interchangeable services, for example, you could see that Azure Container Instances would be way more expensive than running your container in App Services, but it's hard to really figure out where the final cost will shake out.

- Azure Cost Management in the portal gives a full overview of cost across your subscriptions, as well as detailed analysis and forecasting based on current spend. You can also set limits on spending, and generate alerts based on those limits.



Azure Cost Management

- There are ways to save compared to pay-as-you-go like Microsoft Enterprise Agreements, and if you go with VMs make sure to look into Azure Hybrid Benefit and Reserved Instances.

- Cost management is a process, and consumption optimization is not a one-time exercise. You need to regularly reassess costs as both Azure and application requirements change over time.

- The goal should be cost optimization, not cost minimization. For example, if a basic storage queue will work within the solution, there is no reason for a full-blown Service Bus, but if it **needs** a Service Bus, don't use a storage queue just because it is lower cost. It's common to be pressured to deliver a low-cost solution, but if it fails to satisfy the business needs and requirements, it is not a good solution.

- If the decision to move the cloud is primarily driven by cost, be prepared for failure. I mean, yes, cost savings are possible, but rather than look at absolute price, look at the return on investment and business value which includes the additional features and capabilities that add robustness and agility to your solutions.

- Probably most important, always look to "right size" things. Just recreating what you are already running on is not the best way to migrate to Azure. When you run your own hardware, you, by necessity, need to over provision to handle peak load and growth.

- Always remember that you pay for what you are using in Azure and use automatic scaling, whenever possible, to "right size" when you have variable traffic patterns.

And lastly, with Azure, or any cloud solution for that matter: The architecture is the cost, and the cost is the architecture. If you don't like the cost, then you don't like the architecture. That is just as true when migrating applications to the cloud as it is designing cloud native solutions.