Best Practices for Deploying Microsoft SQL Server on AWS

September 2018

Notices

This document is provided for informational purposes only. It represents AWS's current product offerings and practices as of the date of issue of this document, which are subject to change without notice. Customers are responsible for making their own independent assessment of the information in this document and any use of AWS's products or services, each of which is provided "as is" without warranty of any kind, whether express or implied. This document does not create any warranties, representations, contractual commitments, conditions or assurances from AWS, its affiliates, suppliers or licensors. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

Abstract

This whitepaper focuses on best practices to attain the most value for the least cost when running Microsoft SQL Server on the AWS platform. Although for many general-purpose use cases, Amazon Relational Database Service (Amazon RDS) for Microsoft SQL Server provides an easy and quick solution, in this paper we focus on scenarios where you need to push the limits to satisfy your special requirements.

In particular, this whitepaper explains how you can minimize your costs, maximize availability of your SQL Server databases, and optimize your infrastructure for maximum performance. The flexibility of AWS services, combined with the power of Microsoft SQL Server, can provide expanded capabilities for those who seek innovative approaches to optimize their applications and transform their businesses.

The main focus of this paper is on the capabilities available in Microsoft SQL Server 2017, which is the most current version at the time of publication. Existing databases that are running on previous versions (i.e., 2008, 2012, 2014, and 2016) can be migrated to SQL Server 2017 and run in compatibility mode.

Mainstream and extended support for SQL Server 2000 and 2005 has been discontinued by Microsoft[1]. Any database running on those versions of SQL Server must be upgraded to a supported version first. Although it is possible to run those versions of SQL Server on AWS, that discussion is outside the scope of this whitepaper.

[1] https://support.microsoft.com/en-us/lifecycle/search

Introduction

Microsoft SQL Server offers several High Availability/Disaster Recovery (HA/DR) solutions, each suitable for specific requirements. These include:

- Log Shipping
- Mirroring (Deprecated, use Availability Groups instead)
- Always On Availability Groups (Enterprise Edition)
- Always On Basic Availability Groups (Standard Edition)
- Always On Failover Cluster Instances
- Distributed Availability Groups

These solutions rely on one or more secondary servers with SQL Server running as active or passive standby. Based on the specific HA/DR requirements, these servers can be located in close proximity to each other or far apart.

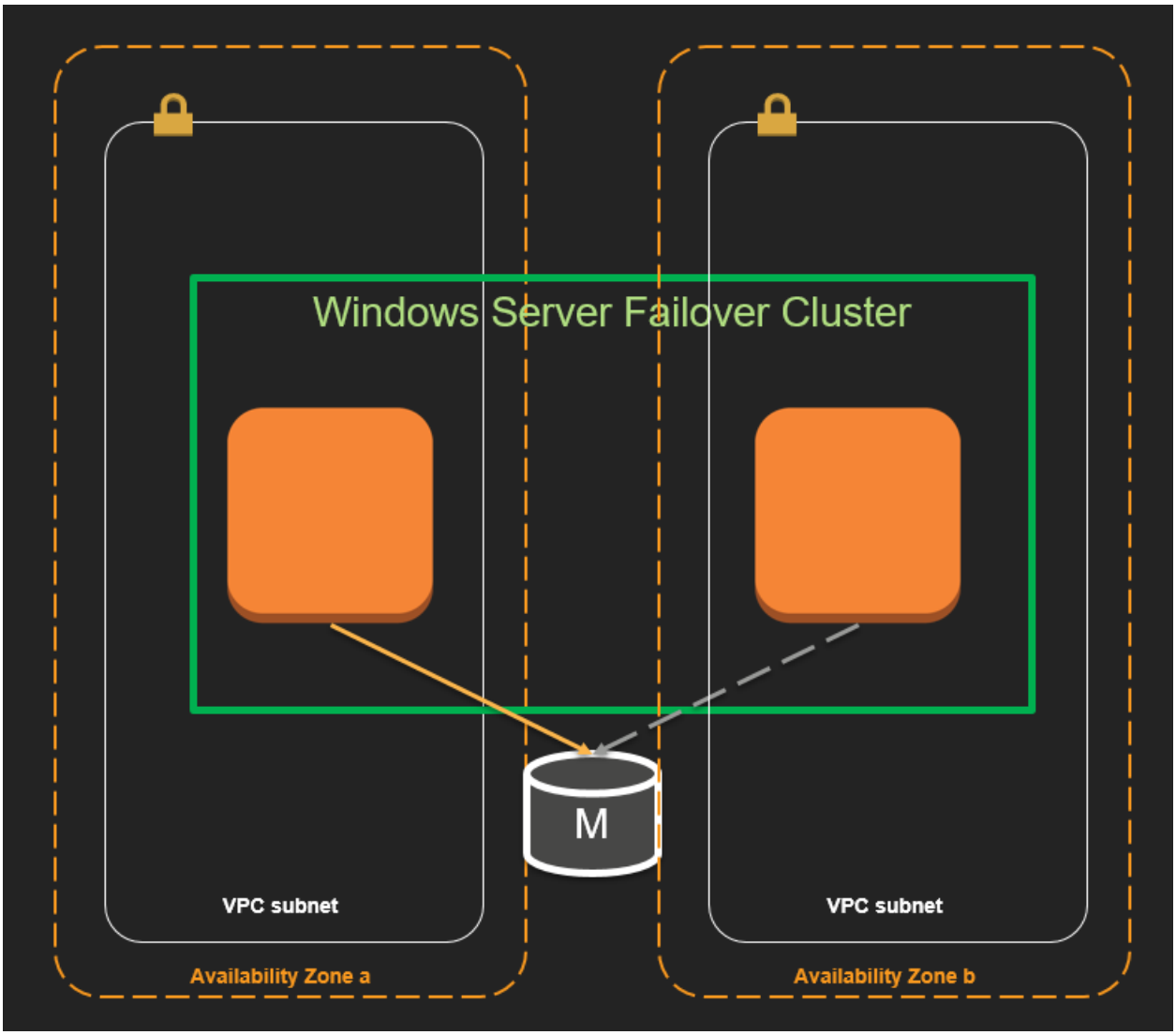
In AWS, you can choose between very low latency or an extremely low probability of failure. Or you can combine these options to create the solution most suitable to your use case. In this paper, we look at these options and how they can be used with SQL Server workloads.

Availability Zones and Multi-AZ Deployment

AWS Availability Zones (AZs) are designed to provide separate failure domains, while keeping workloads in relatively close proximity for low latency inter-communications. AZs are a good solution for synchronous replication of your databases using Mirroring, Always On Availability Groups, or Basic Availability Groups. SQL Server provides zero data-loss and, when combined with the low-latency infrastructure of AWS Availability Zones, provides high performance.

This is one of the main differences between most on-premises deployments and AWS. For example, Always On Failover Cluster Instances (FCI) is often used inside a single datacenter. This is because all nodes in an FCI cluster must have access to the same shared storage. Locating these nodes in different datacenters could adversely impact performance. However, with AWS, FCI nodes can be located in separate AZs and still provide good performance because of the low-latency network link between all AZs within a region. This feature enables a

higher level of availability and could eliminate the need for a third node, which is often coupled with an FCI cluster for disaster recovery purposes.



SQL Server Always On Failover Cluster Instances in AWS

Cluster Placement Groups and Enhanced Networking

Amazon EC2 enables you to deploy a number of EC2 instances inside a cluster placement group. This means those EC2 instances are not only inside a single AZ, but also, to ensure minimum network latency, within close physical proximity in the same datacenter. To gain the highest bandwidth on AWS, you

can leverage enhanced networking and Elastic Network Adapter (ENA)[2]. To minimize latency, you can deploy Always On Failover Cluster Instances, or Always On Availability Groups on instances that run inside an EC2 cluster placement group.

At first glance, this may appear to conflict with HA requirements because the close proximity of servers increases the likelihood of simultaneous failures. However, this approach can be combined with Multi-AZ, to provide higher levels of availability. For example, you can create an Always On FCI inside a cluster placement group and add that cluster as a node to a Multi-AZ Always On Availability Group using asynchronous replication. This arrangement gives you maximum performance while providing failover to another local instance if a process or instance fails.

Additionally, in the case of an AZ failure, you can manually failover your database to the second AZ. Relatively low latency between AZs provides near-zero data-loss, even with asynchronous replication in place. This is an example of HA with close to zero performance penalty for applications that are very sensitive to latency, combined with a DR solution within the same AWS region to minimize the recovery point objective (RPO).

As noted previously, although using a cluster placement group reduces network latency to a minimum, it increases the likelihood of simultaneous failures for instances running inside it. For example, if you have two EC2 instances, both running on the same physical host, or both sharing the same power source and network switch, a failure of any of these underlying components results in the failure of both of your EC2 instances. Instead, you may prefer to sacrifice the lower-latency of cluster placement groups in exchange for lowering the probability of simultaneous failures of your EC2 instances. In such cases, you may leverage spread placement groups to ensure that your instances are placed far enough apart to minimize chances of a partial or full failure at the same time.

[2] https://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/enhanced-networking.html

Multi-Region Deployments

For those workloads that require even more resilience against unplanned events, you can leverage the global scale of AWS to ensure availability under any circumstances.

Since Amazon Virtual Private Cloud (Amazon VPC) is confined within a single region, for a multi-region deployment, you would need to establish connectivity between your VPCs in different regions. Although there are a number of ways to do this, in most cases, the best approach is using inter-region VPC peering. This approach provides security, optimized performance, and enhanced throughput by ensuring that all traffic between your VPCs is encrypted, stays on the AWS global network backbone, and never traverses the Internet.[3]

If you have applications or users that are deployed in remote regions which need to connect to your SQL Server instances, you can leverage the AWS Direct Connect feature that provides connectivity from any Direct Connect connection to all AWS regions.[4]

Although it is possible to have synchronous replication in a multi-region SQL Server deployment, the farther apart your selected regions are, the more severe the performance penalty is for a synchronous replication. Often the best practice for multi-region deployments is to establish an asynchronous replication, especially for regions that are geographically distant. For those workloads that come with aggressive RPO requirements, asynchronous multi-region deployment can be combined with a multi-AZ or single-AZ synchronous replication. You can also combine all three methods into a single solution. However, these combinations would impose significant number of additional SQL Server license costs, which need to be considered as part of your in-advance planning.

In cases involving several replicas across two or more regions, distributed availability groups[5] may be the most suitable option. This feature allows you to combine availability groups deployed in each region into a larger distributed

[3] https://do.awsstatic.com/aws-answers/AWS_Multiple_Region_Multi_VPC_Connectivity.pdf

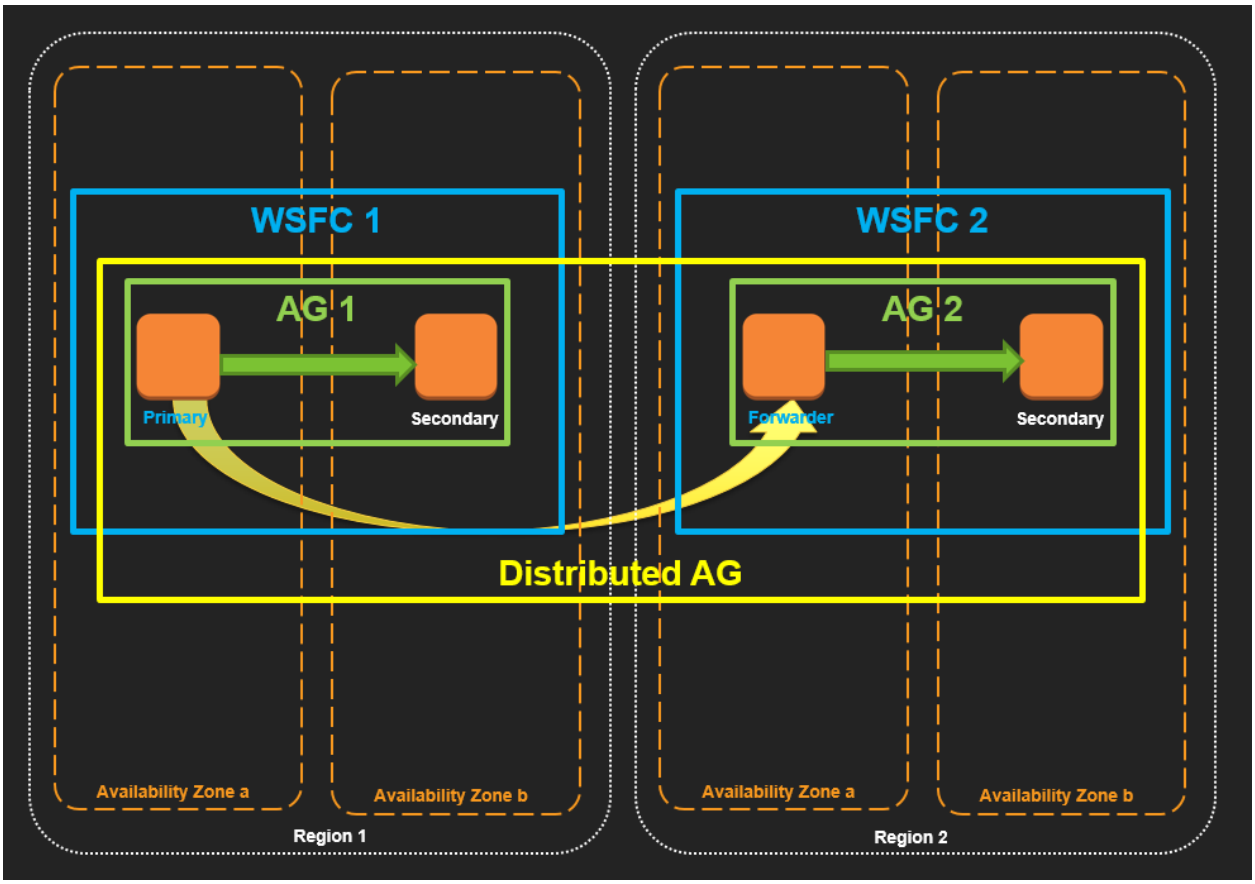
[4] https://docs.aws.amazon.com/directconnect/latest/UserGuide/remote_regions.html

[5] https://docs.microsoft.com/en-us/sql/database-engine/availability-groups/windows/distributed-availability-groups

availability group. In each distributed availability group, one of the availability groups acts as the primary and the others behave as secondaries. Even though each availability group has a primary instance (or cluster) and a number of secondaries of its own, the primary instance of a secondary availability group participating in a distributed availability group acts as a secondary for the primary instance of the primary availability group. These primaries are referred to as "forwarder" instances.

Distributed availability groups can also be used to increase the number of read replicas. A traditional availability group allows up to 8 read replicas. This means that you can have a total of 9 replicas, including the primary. Using a distributed availability group, a second availability group can be added to the first, increasing the total number of replicas to 18. This process can be repeated with a third availability group and a second distributed availability group. The second distributed availability group can be configured to include either the first or second availability groups as its primary. Distributed availability groups is the means through which SQL Server Always On can achieve virtually unlimited scale.

Another use case of a distributed availability group is for zero downtime database migrations. The independence of SQL Server Distributed Availability Group from Active Directory and Windows Server Failover Cluster (WSFC) is the main benefactor for these cases. It allows you to keep both sides of the migration synchronized without having to worry about the complexities of Active Directory or WSFC. You can refer to this blog for more details.



SQL Server Distributed Availability Group in AWS

Performance optimization

In some cases, maximizing performance may be your utmost priority. Both SQL Server and AWS have several options to substantially increase performance of your workloads.

The first and most effective way to improve the performance of your workloads is by optimizing your applications and database schemas. You may also be able to significantly improve application performance by changing your application to use NoSQL instead of a standard relational database. Both AWS and SQL Server provide NoSQL capabilities. Amazon DynamoDB is a managed NoSQL database service that offers provisioned throughput. You may also use the capabilities of Microsoft SQL Server, such as memory-optimized tables, XML, and JSON capabilities. Microsoft SQL Server allows you to mix relational and NoSQL queries and tailor high-performing solutions for complex problems.

Some customers go one step further and evaluate moving their relational databases to Amazon Aurora. Aurora is a fully managed, open-source and enterprise-grade RDBMS service, based on MySQL. It offers high performance and very high levels of elasticity and flexibility. You can also benefit from Aurora's serverless capability to achieve highest degrees of cost-optimization coupled with steady performance, without having to manage your servers. Although technically it may be possible to design a similarly flexible architecture based on SQL Server, but the license constraints on number of active and running servers, leads to such high cost-overheads that render the benefits of automated dynamic horizontal scaling useless.

Another area to consider for performance tuning is your physical database schema. Careful physical design, definition of non-clustered indexes and horizontal partitioning of tables and indexes, may drastically increase your database performance. For analytic processing of transactional data, you may leverage columnstore indexes, a feature of Microsoft SQL Server which can provide up to 10x performance and data compression improvements.

Sometimes, you can solve your scale problems by logically slicing your data into smaller chunks and then host each chunk on a separate server. This technique is called sharding and is particularly useful for scaling write-heavy databases.

However, you may already have gone through application and DB level optimization, or your situation may not lend itself to such practices (e.g., having to run legacy LOB applications, or projecting impending load increase, etc.) In these cases, the next place to focus on is your infrastructure and assess how it can be reconfigured to solve or alleviate your problem. The next sections focus on infrastructure design options for maximizing performance of Microsoft SQL Server on AWS cloud.

Using Amazon Elastic Block Store (Amazon EBS)

Amazon EBS is a single-AZ block storage service with various flexible options, catering for diverse requirements. When it comes to maximizing performance, using a Provisioned IOPS volume type (io1) is the best choice. You can provision up to 32,000 IOPS per io1 EBS volume (based on 16 KiB I/O size), along with 500 MB/s throughput.

If you need more IOPS and throughput than provided by a single EBS volume, you can create multiple volumes and stripe them in your Windows or Linux instance (Microsoft SQL Server 2017 may be installed on both Windows and Linux). Striping allows you to further increase available IOPS per instance up to 80,000, and throughput per instance up to 1750 MB/s.[6]

One point to remember is to use EBS-optimized EC2 instance types. This means a dedicated network connection will be allocated to serve between your EC2 instance and EBS volumes attached to it.

One of the major benefits of using EBS volumes, is ability to create point-in-time and instantaneous EBS snapshots. This feature copies the EBS snapshot to Amazon S3 infrastructure, an AWS service that comes with 99.999999999% durability. Despite EBS volumes being confined to a single AZ, EBS snapshots may be restored to any AZ within the same region.

Although each EBS volume can be as large as 16 TB, it could take a long time to transfer all of its data to S3, but EBS snapshots are always point-in-time. This means that SQL Server and other applications can continue reading and writing to and from EBS volume, while data is being transferred in the background.

You can use AWS Systems Manager Run Command to take application-consistent EBS snapshots of your online SQL Server files at any time, with no need to bring your database offline or in read-only mode. The snapshot process uses Windows Volume Shadow Copy Service (VSS) to take image-level backups of VSS-aware applications.[7] Microsoft SQL Server has been VSS-aware for more than a decade and is perfectly compatible with this technique. The process works with regular EBS volumes, as well as striped volumes (i.e., single file-system on top of several volumes). It is also possible to take VSS snapshots of Linux instances, however, that process requires some manual steps, because Linux does not natively support VSS.

EBS volumes are simple and convenient to use, and in most cases effective, too. However, there may be circumstances where you need even higher IOPS and throughput than what is offered by Amazon EBS.

[6] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSVolumeTypes.html

[7] https://docs.aws.amazon.com/systems-manager/latest/userguide/integration-vss.html

Instance Storage

Storage optimized EC2 instance types[8] use fixed-size local disks and a variety of different storage technologies are available. Among these, Non-Volatile Memory express (NVMe) is the fastest technology with the highest IOPS and throughput. The i3 class of instance types provides NVMe SSD drives, e.g., i3.16xlarge, comes with 8 disks, each with 1.9 TB of storage. When selecting storage optimized EC2 instance types for maximum performance, it is essential to understand that some of the smaller instance types provide instance storage that is shared with other instances. These are virtual disks that reside on a physical disk attached to the physical host. By selecting a bigger instance type, such as i3.2xlarge, you ensure that there is a 1:1 correspondence between your instance store disk and the underlying physical disk. This ensures consistent disk performance and eliminates the noisy-neighbor problem.

Instance disks are ephemeral and only live as long as their associated EC2 instance lives. If the EC2 instance fails or is terminated, all of its instance storage disks are wiped out and the data stored on them is irrecoverably lost. Unlike EBS volumes, instance storage disks cannot be backed up using a snapshot. Therefore, if you choose to use EC2 instance storage for your permanent data, you need to provide a way to increase its durability.

One suitable use for instance storage may be the tempdb system database files because those files are recreated each time the SQL Server service is restarted. SQL Server drops all tempdb temporary tables and stored procedures during shutdown. As a best practice, the tempdb files should be stored on a fast volume, separate from user databases. For the best performance, ensure that the tempdb data files within the same filegroup are the same size and stored on striped volumes.[9]

Another use for EC2 instance storage is the buffer pool extension[10]. Buffer pool extension is available on both the Enterprise and Standard editions of Microsoft SQL Server. This feature utilizes fast random access disks (SSD) as a secondary

[8] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/storage-optimized-instances.html

[9] https://docs.microsoft.com/en-us/sql/relational-databases/databases/tempdb-database

[10] https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/buffer-pool-extension

cache between RAM memory and persistent disk storage, thus, striking a balance between cost and performance when running workloads on SQL Server.

Although instance storage disks are the fastest available to EC2 instances, their performance is capped at the speed of the physical disk. You can go beyond the single disk maximum by striping across several disks.

You could also use instance storage disks as the cache layer in a Storage Spaces Direct storage pool.

Scale-Out File Server

Windows Server 2016 introduced a new service called Storage Spaces Direct (S2D). S2D allows all or selected disks attached to instances of a Windows Server Failover Cluster (WSFC) to be clustered into a storage pool and made available to all members of the cluster. S2D has flexible options to maximize performance, storage space, and durability. It removes the complexities of managing different disk technologies and creating a RAID cluster spread across several servers in a network.[11]

Using Amazon EC2 Windows instances along with S2D solves both problems of durability and scale. You can deploy any number of Windows EC2 instances, join them to the same Windows AD domain, and create a WSFC. (Windows Server 2016, and later, allow multi-domain clusters, as well as a WSFC of nodes in a workgroup, i.e., nodes not joined to any AD domain.) Then you can enable S2D in your WSFC and add all attached NVMe disks to your pool. Next, you can create an SMB 3.0 share drive on top of this pool. SQL Server software can be installed on the same WSFC, or it may be on a separate instance (S2D converged model). The latter option effectively decouples your SQL Server storage from its software, allowing you to choose other instance types that may be more appropriate for your SQL Server software. For example, you may use compute optimized instance types to increase the number of cores available for processing, or you may choose memory optimized instance types to maximize available memory (particularly useful with memory-optimized tables in SQL

[11] https://docs.microsoft.com/en-us/windows-server/storage/storage-spaces/storage-spaces-direct-overview

Server.) These optimizations can reduce the number of vCPU cores needed and reduce your SQL Server license costs.

To prevent the network from becoming a bottleneck between the storage and SQL Server instances, you can launch your EC2 instances in a cluster placement group. Selecting instance types that support enhanced networking using the Elastic Network Adapter (ENA) is another option to consider.[12] ENA is natively available on Windows and can be installed on Linux instances. Many AWS Linux images come with the ENA driver installed by default. You can have your SQL Server software installed on either Windows or Linux, and Scale-Out File Server (SOFS) on your Windows Server Failover Cluster. The SOFS share drive is available to mount on your SQL Server instance using the SMB 3.0 protocol. SMB 3.0 is natively supported on Windows and has recently become available on some popular Linux distributions.

When tuning for performance, it is important to remember the difference between latency and bandwidth. You should find a balance between network latency and availability. Using ENA drivers maximizes bandwidth, but it has no effect on latency. Network latency changes in direct correlation with the distance between interconnecting nodes. Clustering nodes is a way to increase availability, but placing cluster nodes too close to each other increases the probability of simultaneous failure, therefore, reducing availability. Putting them too far apart yields the highest availability, but at the expense of higher latency.

AWS Availability Zones within each AWS region are engineered to provide a balance that fits most practical cases. Each AZ is engineered to be physically separated from other AZs, while keeping in close geographic proximity to provide very low network latency. Therefore, in the overwhelming number of cases, the best practice is to spread cluster nodes across multiple AZs.

Read Replicas

You may determine that many of your DB transactions are read-only queries, and that the sheer number of incoming connections is flooding your database. Read replicas are a known solution for this situation. You can offload your read-only transactions from your primary SQL Server instance to one or more read

[12] https://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/enhanced-networking.html

replica instances. Read replicas can also be used to perform backup operations, relieving primary instance from performance hits during backup windows. When using availability group (AG) listeners, if you mark your connection strings as read-only, SQL Server routes incoming connections to any available read replicas and only sends read-write transactions to the primary instance.

Always On Availability Groups, introduced in SQL Server 2012, supports up to four secondary replicas. In more recent versions of SQL Server (i.e., 2014, 2016, and 2017), Always On Availability Groups support one set of primary databases and one to eight sets of corresponding secondary databases.[13]

There may be cases where you have users or applications connecting to your databases from geographically dispersed locations. If latency is a concern, you can locate read replicas in close proximity to your users and applications.

Note that when you use a secondary database for read-only transactions or backup operations, you must ensure that the server software is properly licensed.

Security optimization

Security is the first priority at AWS, and there are many AWS security features available to you. These features can be combined with the built-in security features of Microsoft SQL Server to satisfy even the most stringent requirements and expectations.

Amazon EBS Encryption

If you are using EBS volumes to store your SQL Server database files, you have the option to enable block-level encryption. Amazon EBS transparently handles encryption and decryption for you. This is available through a simple checkbox, with no further action necessary.

[13] https://docs.microsoft.com/en-us/sql/database-engine/availability-groups/windows/overview-of-always-on-availability-groups-sql-server

AWS Key Management Service (KMS)

AWS KMS is a fully managed service to create and store encryption keys. You can use KMS-generated keys or bring your own keys. In either case, keys never leave KMS and are protected from any unauthorized access.

You can use KMS keys to encrypt your SQL Server backup files when you store them on Amazon S3, Amazon Glacier, or any other storage service. Amazon EBS encryption also integrates with AWS KMS.

Transparent Data Encryption (TDE)

TDE is a feature available in Microsoft SQL Server that provides transparent encryption of your data at rest. TDE is available on Amazon RDS for SQL Server, and you can also enable it on your SQL Server workloads on EC2 instances.[14]

This feature is only available on SQL Server Enterprise Edition. However, if you want to have encryption-at-rest for your database files on Standard Edition, you can use EBS encryption instead.

If you consider using EBS encryption instead of SQL Server TDE, it is important to understand their differences. EBS encryption is done at the block level, i.e., data is encrypted when it is stored and decrypted when it is retrieved. However, with TDE, the encryption is done at the file level, i.e., database files are encrypted and can only be decrypted using the corresponding certificate. For example, this means that if you use EBS encryption without TDE and copy your database data or log files from your EC2 instance to an S3 bucket, the files will not be encrypted. Furthermore, if someone gains access to your EC2 instance, database files will be exposed instantly.

Always Encrypted

Always Encrypted is a feature that allows separation between data owners and data managers. Sensitive data that is stored in Microsoft SQL Server using Always

[14] https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/transparent-data-encryption

Encrypted, stays encrypted even during query processing. Encryption keys remain with the data owners and are not revealed to the database engine.[15]

Row-Level Security

Row-Level Security (RLS) in SQL Server enables you to control database access at the row level. This feature reduces your attack surface by filtering out all unauthorized access attempts, originating from any layer of your application, directly from the database. It could potentially simplify your applications, but you need to design your applications in a way that differentiates users at the database level. For example, if you have a web application that shares the same connection string for all database operations, this feature would not be applicable. This situation needs to be considered at application design time.

Dynamic Data Masking

Dynamic Data Masking (DDM) is another feature that can simplify design and implementation of security requirements in applications. You can use DDM for partially or fully masking certain fields when they are returned as part of query results. You can leverage central policies to apply DDM on sensitive data.[16]

Amazon VPC

There are many features in Amazon VPC that help you to secure your data in transit. You can use security groups to restrict access to your EC2 instances and only allow whitelisted endpoints and protocols. You can also use network access control lists to blacklist known sources of threats.

A best practice is to deploy your SQL Server instances in private subnets inside a VPC, and only allow access to the Internet through a VPC NAT gateway, or a custom NAT instance.

[15] https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-database-engine

[16] https://docs.microsoft.com/en-us/sql/relational-databases/security/dynamic-data-masking

Whitelisting

You can leverage Windows Server Group Policies to whitelist your SQL Server software, and possibly other known applications, on your EC2 Windows instances. This ensures that nothing but your whitelisted applications can run on those servers. This is one of the most effective ways to eliminate the possibility of having malware infect your instances. This way, anything other than legitimate applications are filtered at the Windows kernel level.

Linux does not natively support a mechanism for whitelisting applications. However, there are third-party tools that provide a similar function on Linux.

Cost optimization

SQL Server can be hosted on AWS through License Included (LI), as well as Bring Your Own License (BYOL) licensing models. With LI, you run SQL Server on AWS and pay for the licenses as a component of your AWS hourly usage bill. The advantage of this model is that you do not need to have any long-term commitments and can stop using the product at any time and stop paying for its usage.

However, many businesses already have considerable investments in SQL Server licenses and may want to reuse their existing licenses on AWS. This is possible using BYOL:

1. If you have Software Assurance (SA), one of the benefits is license mobility. License mobility allows you to use your licenses on server instances running anywhere, including on Amazon EC2 instances.
2. If you do not have SA, you can still use your own licenses on AWS. However, there are specific requirements for running SQL Server instances on dedicated hardware. Therefore, you would need to use Amazon EC2 Dedicated Hosts[17] to ensure license compliance. AWS is the only major public cloud that provides this option.

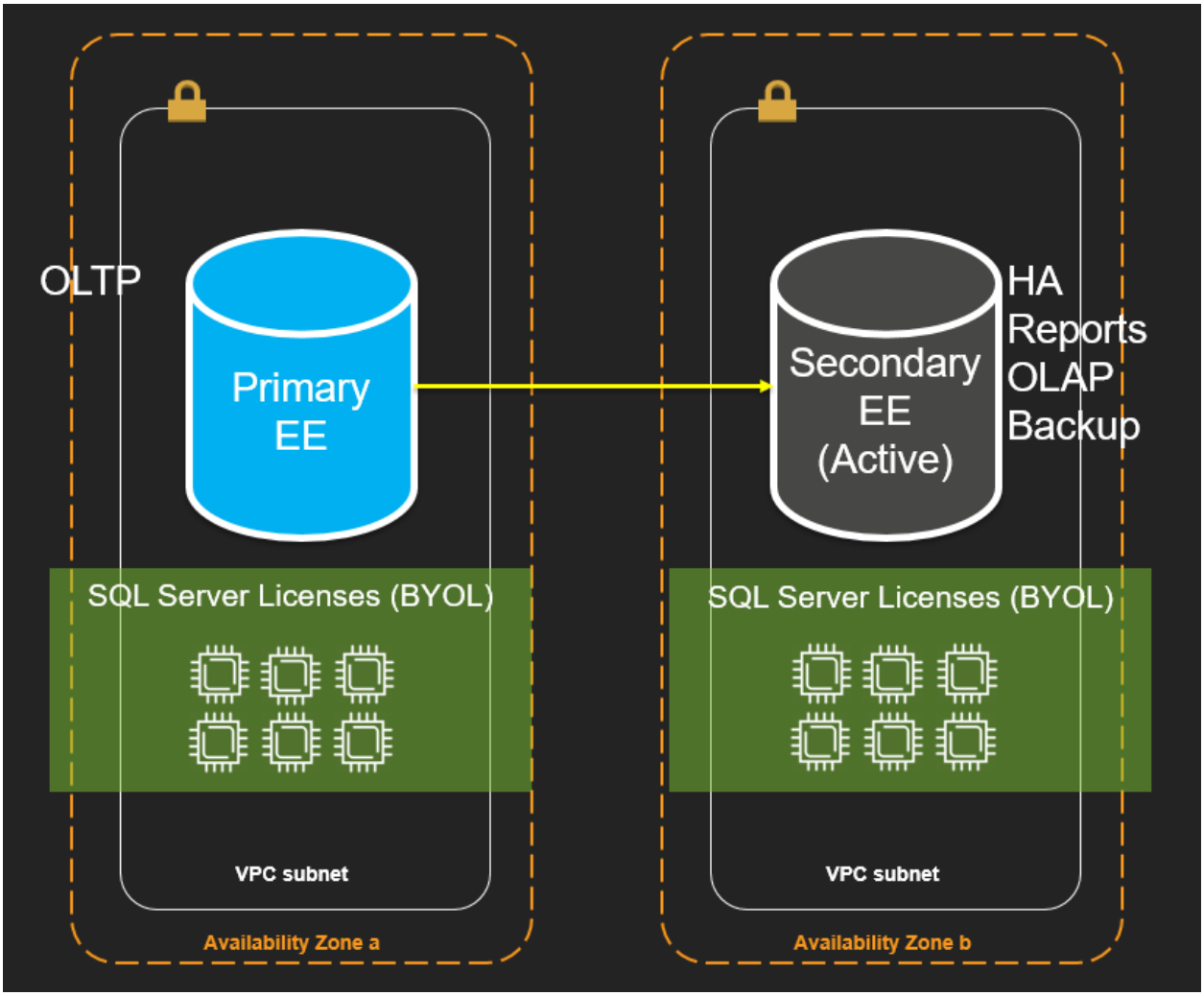
One of the best opportunities for cost optimization in the cloud is through applying a combination of BYOL and LI models. A common use case is SQL

[17] https://aws.amazon.com/ec2/dedicated-hosts/

Server Always On Availability Groups with active replicas. Active replicas are used primarily for:

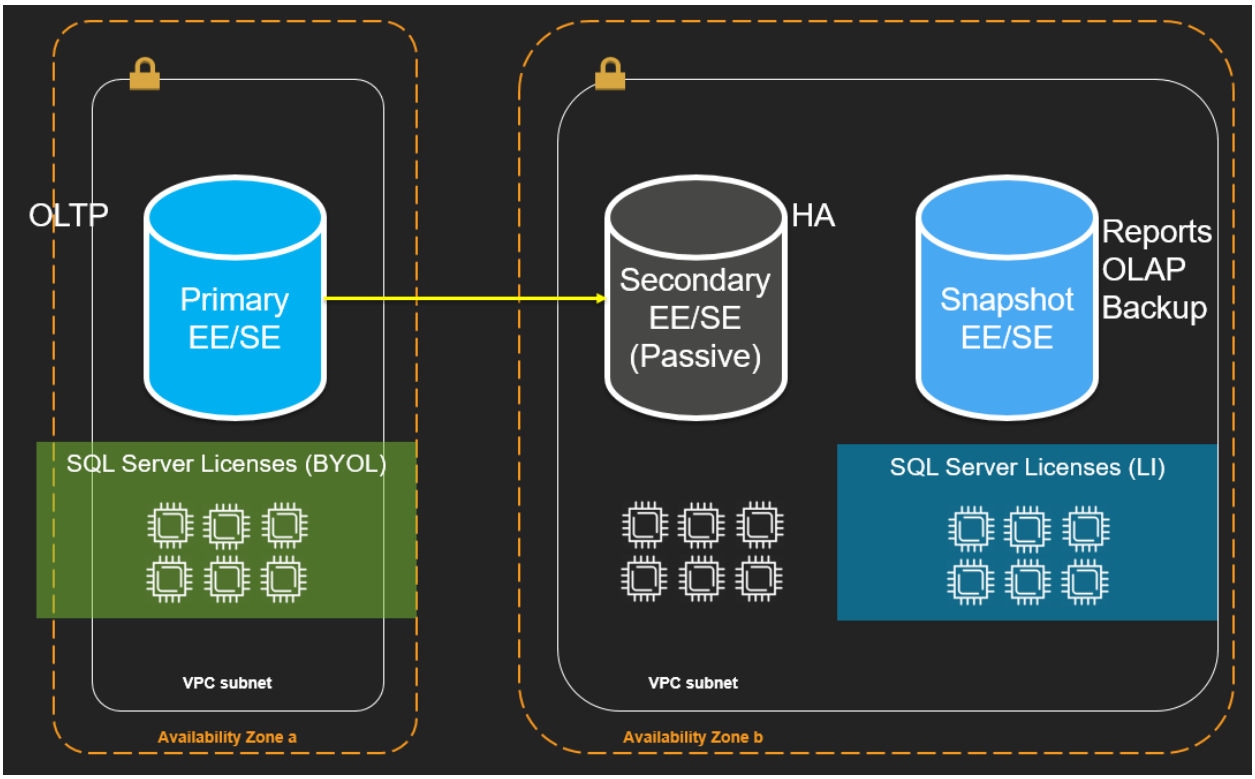
1. Reporting
2. Backup
3. OLAP Batch jobs
4. HA

Out of the above four operations, the first three are often performed intermittently. This means that you would not need an instance continuously up and dedicated to running those operations. In a traditional on-premises environment, you would have to create an active replica that is continuously synchronized with the primary instance. This means you need to obtain an additional license for the active replica.



SQL Server active replication on-premises

In AWS, there is an opportunity to optimize this architecture by replacing the active replica with a passive replica, therefore relegating its role solely for the purpose of HA. Other operations can be performed on a separate instance using License Included which could run for a few hours and then be shut down or terminated. The data can be restored through an EBS snapshot of the primary instance. This snapshot can be taken using VSS-enabled EBS snapshots, thus ensuring no performance impact or downtime on the primary.



Eliminating active replica licenses in AWS

Please refer to our Microsoft licensing page[18] for more details.

SQL Server inside Docker Container

> "SQL Server 2017 offers use rights for virtual machines and containers, to provide flexibility for customers' deployments. There are two primary licensing options for virtual machines and containers in SQL Server 2017 – the ability to license individual virtual machines and containers and the ability to license for maximum densities in highly virtualized or high-density container environments." (SQL Server 2017 Licensing Datasheet)[19]

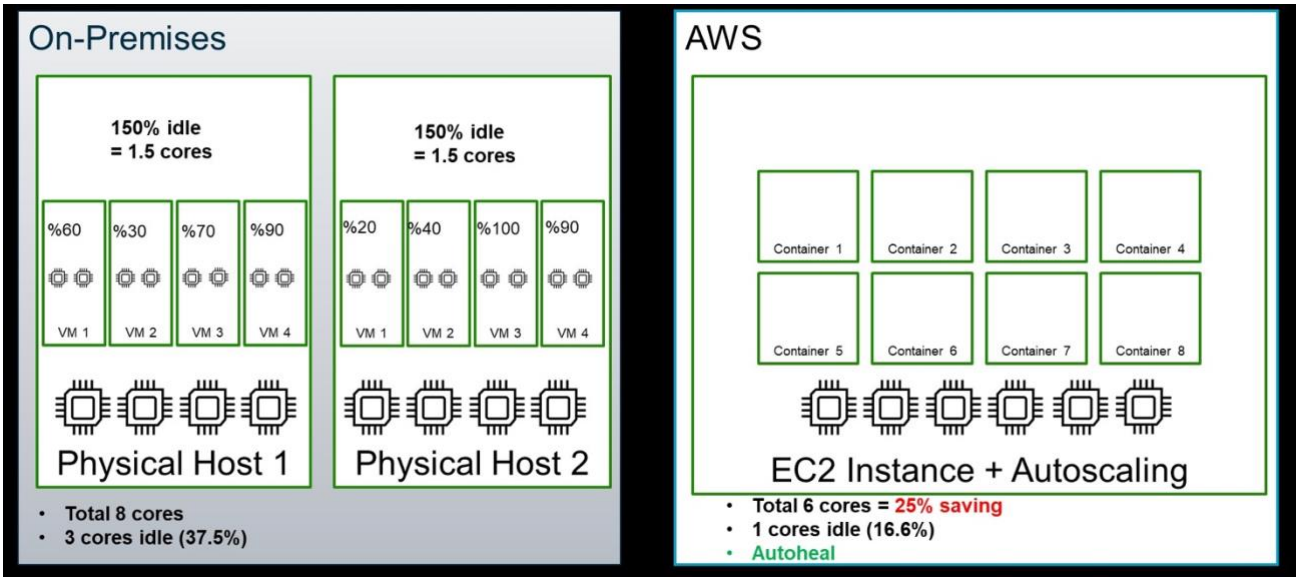
[18] https://aws.amazon.com/windows/resources/licensing/

[19] https://download.microsoft.com/download/B/C/0/BC0B2EA7-D99D-42FB-9439-2C56880CAFF4/SQL_Server_2017_Licensing_Datasheet.pdf

Although containerization is a new feature introduced in SQL Server 2017, but instances of SQL Server 2017 can be used to host databases with compatibility levels of earlier versions. Businesses who wish to use BYOL with SQL Server licenses on AWS, often have Software Assurance available. SA gives them the benefit of license mobility, as well as upgrade to SQL Server 2017.

Some of the benefits of running SQL Server inside containers, include higher flexibility than VM deployments, and it does so with much less overhead. Unlike VMs, containers do not need a guest OS running on top of a hypervisor and host OS. Instead, all containers share the same OS kernel of the host. This means it is possible to run a far greater number of containers on the same server, with almost no extra overhead. Although, hyperthreading on AWS is always done with a fixed ratio of physical cores to vCPUs (1:2), containers allow any arbitrary ratio. Each core of a container host is translated to 1024 ECU units for the container. Based on your compute capacity requirements, you can assign any number of cores, or a fraction of a core, to each container. This flexibility is the key to eliminate problems mentioned in previous sections.

Another advantage of this solution is the elasticity of SQL Server containers running on the same EC2 instance or cluster. Resources allocated to a VM are fixed and increasing or decreasing resources always requires downtime. However, containers share resources available on the same host, therefore, they can use more or less resources as needed over time. This is a common advantage that comes with leveraging high-density solutions. If a SQL Server instance requires more resources that are not available on its current host, it can be moved to a new host in a few seconds. Boot time for SQL Server containers is a matter of seconds and the speed and agility are major benefits of this solution.



Containers increasing efficiency as well as efficacy of SQL Server infrastructure

In this example, there are two physical hosts with 8 CPU cores. As shown in the picture, a VM that runs at 100% capacity cannot utilize any of the idle resources available on the same host. Despite the availability of 3 idle CPU cores, VM 3 is running in a resource constrained state.

The situation is significantly improved by moving to containers, as shown on the right side of the picture. Not only are no containers resource constrained, but also the overall number of physical cores is reduced from 8 to 6. The same principle can be applied to available memory. Containerization can improve the SQL Server infrastructure to be both more cost-efficient as well as capable of delivering better performance.

Currently, only SQL Server in Linux containers are available for production workloads.

Generally speaking, any SQL Server workload (or any other DB for that matter) is composed of two major components:

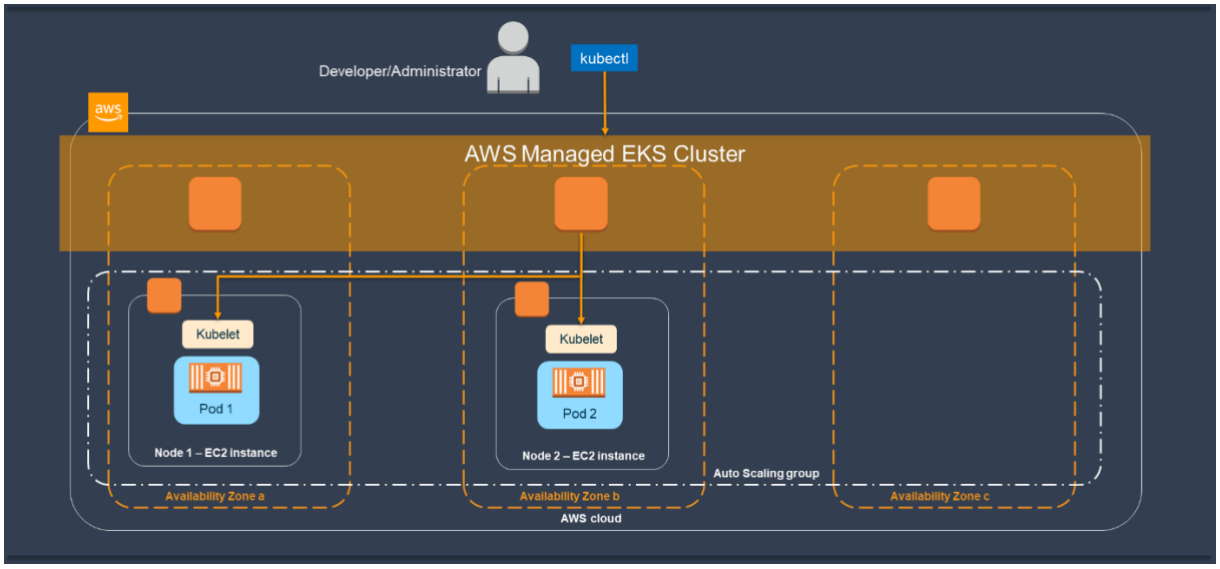
- The software, i.e., the executing process
- The storage, i.e., the files on the file system

In our case, the SQL Server software runs inside a container, which can be deployed through the orchestration engine (i.e., Amazon EKS) on any node (i.e., EC2 instances) of its cluster.

For storage, the options are:

1. Storage Classes backed by EBS volumes attached to host EC2 instances.
2. Storage Classes based on a clustered storage solution, such as Portworx
3. Storage Class for NFS file share, deployed on a Linux clustering solution, such as GlusterFS.

Option 1 is the simplest and most cost-effective solution. It can use a single EBS volume of type GP2 (general purpose) or IO1 (provisioned IOPS). Options 2 and 3 provide higher availability, as the cluster can be multi-AZ, and potentially higher performance, as several EBS volumes can be striped to form a storage pool. But these advantages come at the expense of higher storage costs, as all blocks must be redundantly stored in a second or third mirror volume.



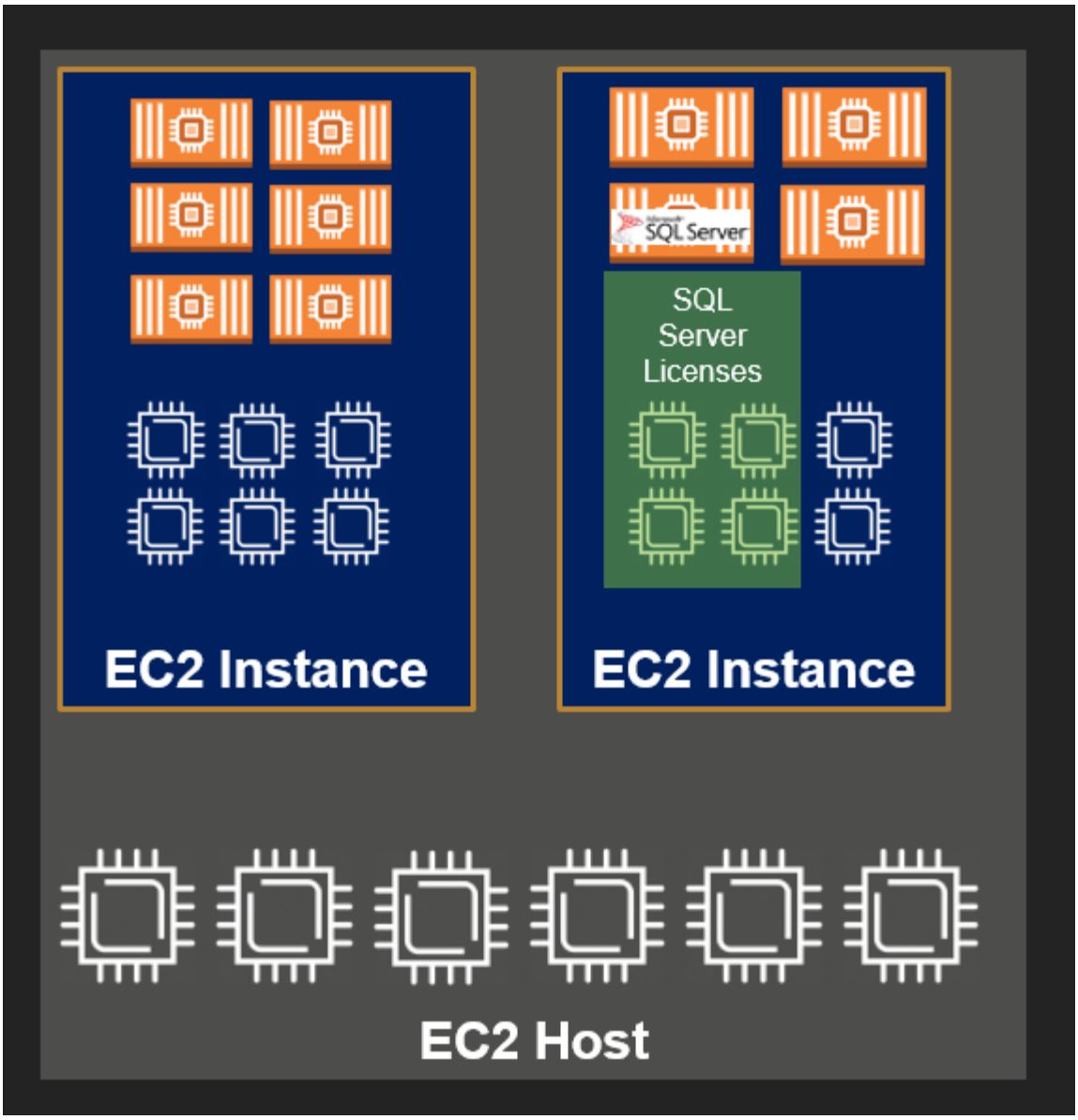
SQL Server inside Docker container, using Amazon EKS and Amazon EFS

Licensing Processors Available to SQL Server

There are cases in which you may need to run SQL Server on an instance that comes with too many processors but need to keep the number of applicable core licenses to a minimum. Consider the following cases:

- You have selected an instance type based on your memory or networking requirements, and the target instance type comes with too many vCPU cores, therefore increasing the number of core licenses required and shooting up costs.
- You are migrating a legacy application that needs to run on the same server as SQL Server (e.g., because of using some shared libraries).

In these cases, you can leverage SQL Server containers to solve your problem. When you run SQL Server on a server with multiple cores, you need to license all cores available in the instance, regardless of how many are actually used by SQL Server. However, if you run SQL Server inside a container, you can limit the number of cores accessible to your container, thus reducing the number of licenses required to only what is available to the container. In such cases, the best practice is to limit the number of cores accessed by a container to no less than 4 cores. That is because, similar to other virtual OSE options, the minimum number of per-core licenses allowed by SQL Server is 4, with additional licensing costs based on increasing increments of 2 (i.e., 4, 6, 8, 10, etc.)



Applying SQL Server licenses to each container

Conclusion

In this whitepaper, we described a number of best practices for deploying SQL Server workloads on AWS. We saw how AWS services can be used to compliment Microsoft SQL Server features to address different requirements.

Each solution and approach may be embraced according to particular business requirements. SQL Server containers may be used along with EC2 Dedicated Hosts, in order to tackle over-subscription licensing issue. AWS Availability

Zones, regions, EC2 cluster and spread placement groups, VPC peering, and inter-region AWS Direct Connect can be used to implement various solutions covering almost any type of HA requirements. Different Microsoft Windows, Linux, and SQL Server features can be used in conjunction with AWS services to create scalable infrastructures, satisfying high performance requirements.

Contributors

The following individuals and organizations contributed to this document:

- Sepehr Samiei, Solutions Architect, Amazon Web Services

Document Revisions

Date	Description
September 2018	First publication