Performance Efficiency Pillar

AWS Well-Architected Framework

November 2016

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

The focus of this paper is the Performance Efficiency pillar of the Amazon Web Services (AWS) Well-Architected Framework. It provides guidance to help customers apply best practices in the design, delivery, and maintenance of AWS environments.

Introduction

At Amazon Web Services, we understand the value of educating our customers about architectural best practices for designing and operating reliable, secure, efficient, and cost-effective systems in the cloud. As part of this effort, we developed the AWS Well-Architected Framework, which helps you understand the pros and cons of decisions you make while building systems on AWS. We believe well-architected systems greatly increase the likelihood of business success.

The framework is based on five pillars:

- Security
- Reliability
- Performance Efficiency
- Cost Optimization
- Operational Excellence

This paper focuses on the Performance Efficiency pillar and how to apply it to your solutions. In traditional, on-premises environments, achieving high and lasting performance can be challenging. By adopting the practices in this paper you will build architectures that are efficient and deliver sustained performance over time.

This paper is intended for those in technology roles, such as chief technology officers (CTOs), architects, developers, and operations team members. After reading this paper, you will understand AWS best practices and strategies to use when designing a performant cloud architecture. This paper doesn't provide implementation details or architectural patterns. However, it does include references to appropriate resources for this information.

Performance Efficiency

The Performance Efficiency pillar focuses on the efficient use of computing resources to meet requirements and the maintenance of that efficiency as demand changes and technologies evolve. This paper provides in-depth, best-practice guidance for architecting for performance efficiency on AWS.

Design Principles

In the cloud, there are a number of principles that can help you achieve performance efficiency.

- **Democratize advanced technologies:** Technologies that are difficult to implement can become easier to consume by pushing that knowledge and complexity into the cloud vendor's domain. Rather than having your IT team learn how to host and run a new technology, they can simply consume it as a service. For example, NoSQL databases, media transcoding, and machine learning are all technologies that require expertise that is not evenly dispersed across the technical community. In the cloud, these technologies become services that your team can consume while they focus on product development rather than resource provisioning and management.
- **Go global in minutes:** Easily deploy your system in multiple regions around the world with just a few clicks. This allows you to provide lower latency and a better experience for your customers at minimal cost.
- **Use *serverless* architectures:** In the cloud, server-less architectures remove the need for you to run and maintain servers to carry out traditional compute activities. For example, storage services can act as static websites, removing the need for web servers, and event services can host your code for you. This removes the operational burden of managing these servers, and also can lower transactional costs because these managed services operate at cloud scale.
- **Experiment more often:** With virtual and automatable resources, you can quickly carry out comparative testing using different types of instances, storage, or configurations.
- **Mechanical sympathy:** Use the technology approach that aligns best to what you are trying to achieve. For example considering data access patterns when selecting database or storage approaches.

Definition

Performance Efficiency in the cloud is composed of four areas:

- Selection

- Review
- Monitoring
- Trade-offs

Take a data-driven approach to selecting a high performance architecture. Gather data on all aspects of the architecture, from the high-level design to the selection and configuration of resource types. By reviewing your choices on a cyclical basis, you will ensure that you are taking advantage of the continually evolving AWS platform. Monitoring will ensure that you are aware of any deviance from expected performance and can take action on it. Finally, your architecture can make tradeoffs to improve performance, such as using compression or caching, or relaxing consistency requirements.

Selection

The optimal solution for a particular system will vary based on the kind of workload you have, and will often combine multiple approaches. Well-architected systems use multiple solutions and enable different features to improve performance.

In AWS, resources are virtualized and are available in a number of different types and configurations. This makes it easier to find an approach that closely matches with your needs, and you can also find options that are not easily achievable with on-premises infrastructure. For example, a managed service such as Amazon DynamoDB provides a fully managed NoSQL database with single-digit millisecond latency at any scale.

You can use AWS Solutions Architects, AWS Reference Architectures, and AWS Partners to help you choose an architecture based on what you have learned, but data obtained through benchmarking or load testing will be required to optimize your architecture. After you have identified your architectural approach, you should use a data-driven process to refine your selection of resource types and configuration options. You can obtain this data using benchmarking and load testing. For more information, see the Review section later in this whitepaper.

Your architecture will likely combine a number of different architectural approaches (e.g., event driven; extract, transform, load (ETL); or pipeline). The implementation of your architecture will use the AWS services that are specific to the optimization of your architecture's performance. In the following sections we look at the four main resource types that you should consider: compute, storage, database, and network.

Compute

The optimal compute solution for a particular system may vary based on application design, usage patterns, and configuration settings. Architectures may use different compute solutions for various components and enable different features to improve performance. Selecting the wrong compute solution for an architecture can lead to lower performance efficiency.

When architecting your use of compute you should take advantage of the elasticity mechanisms available to ensure you have sufficient capacity to sustain performance as demand changes. In AWS, compute is available in three forms: instances, containers, and functions. You should actively choose which form of compute to use for each component in your architecture. Instances are generally the default option; using containers can improve the utilization of instances; and functions are well suited to event-driven or highly parallelizable tasks.

Instances

Instances are virtualized servers and, therefore, you can change their capabilities with the click of a button or an API call. Because in the cloud resource decisions are no longer fixed, you should experiment with different server types.

In the AWS compute service, Amazon Elastic Compute Cloud (EC2), these virtual server *instances* come in different families and sizes, and they offer a wide variety of capabilities, including solid state drives (SSDs) and graphics processing units (GPUs). When you launch an EC2 instance, the instance type that you specify determines the hardware of the host computer used for your instance. Each instance type offers different compute, memory, and storage capabilities. Instance types are grouped in instance families based on these capabilities.

When selecting instance families and types also consider the configuration options that your workload needs:

- **Graphics Processing Units** (GPU). Using General-Purpose computing on Graphics Processing Units (GPGPU), you can build applications that benefit from the high degree of parallelism that GPUs provide by leveraging platforms such as CUDA in the development process. Also, if your application requires 3D rendering or video compression, GPUs enable hardware-accelerated computation and encoding, making your application more efficient.
- **Burstable instance families**. *Burstable instances* are designed to provide moderate baseline performance and the capability to burst to significantly higher performance when it's required by your workload. These instances are intended for workloads that don't use the full CPU often or consistently, but occasionally need to burst. They are well suited for general-purpose workloads, such as web servers, developer environments, and small databases. These instances provide CPU credits that can be consumed when the instance needs to provide performance. The credits will be accumulated when the instance does not need them.
- Amazon EC2 gives you access to **advanced computing features**, such as managing the C-state and P-state registers and controlling the turbo-boost of processors. Access to co-processors allows cryptography operations offloading through AES-NI, or advanced computation through AVX extensions.

You should use data to select the optimal EC2 instance type for your workload, ensure you have the correct networking and storage options, and also consider operating system settings that can improve the performance for your workload.

Containers

Containers are a method of operating system virtualization that allow you to run an application and its dependencies in resource-isolated processes.

Amazon EC2 Container Service (ECS) allows execution and management of containers on a cluster of Amazon EC2 instances in an automated way. You can create services in Amazon ECS that will be served by containers. By using application Auto Scaling, you can define automated, metrics-based scaling for your services so that the number of containers supporting your service grows as the service needs grow.

Amazon ECS leverages Auto Scaling groups, which enable you to scale the Amazon ECS cluster by adding EC2 instances. This way you ensure that you always have the appropriate underlying server capacity to host your containers.

Amazon ECS is integrated with Elastic Load Balancing (ELB) to support load balancing your services in the container fleet dynamically. By creating an Application Load Balancer instance, your containers will automatically register themselves to the load balancer when they scale. With these features, you can host multiple containers serving the same service on a single Amazon EC2 instance, increasing the scalability of your application.

When using containers, you should use data to select the optimal type for your workload as you use data to select your EC2 instance types. You should also consider container configuration options such as memory, CPU, and tenancy configuration.

Functions

Functions abstract the execution environment from the code you want to execute. The use of functions allows you to execute code or provide a service without running or managing an instance.

AWS Lambda lets you run code without provisioning or managing servers or containers. You just upload your code and AWS Lambda takes care of everything required to run and scale your code. You can set up your code to automatically trigger from other AWS services; you can call it directly; or you can use it with Amazon API Gateway.

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. You can create an API that acts as a “front door” to your AWS Lambda function. API Gateway handles all the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls, including traffic management, authorization and access control, monitoring, and API version management.

To deliver optimal performance with AWS Lambda, you should choose the amount of memory you want for your function, and then you are allocated proportional CPU power and other resources. For example, choosing 256 MB of memory allocates approximately twice as much CPU power to your Lambda

function as requesting 128 MB of memory. You can also control the amount of time each function is allowed to run (up to a maximum of 300 seconds).

Elasticity

Elasticity allows you to match the supply of resources you have against demand for them. Instances, containers, and functions all provide mechanisms for elasticity either in combination with Auto Scaling or as a feature of the service itself.

Optimally matching supply to demand delivers the lowest costs for a system, but you also need to plan for sufficient extra supply to allow for provisioning time and individual resource failures. Demand can be fixed or variable, requiring metrics and automation to ensure that management does not become a burdensome and disproportionately large cost in itself.

In AWS, you can use a number of different approaches to match supply with demand. The *Cost Optimization Pillar of the AWS Well-Architected Framework* whitepaper describes how to use each of these approaches:

- Demand-based approach
- Buffer-based approach
- Time-based approach

Key AWS Services

The key AWS service for elastic compute solutions is Auto Scaling because you use it to match the supply of your resources against demand for them. Instances, containers, and functions all provide mechanisms for elasticity either in combination with Auto Scaling or as a feature of the service itself.

Resources

Refer to the following resources to learn more about AWS best practices for compute.

Documentation

- Instances: Instance Types
- Containers: Amazon ECS Container Instances

- Functions: Compute Requirements – Lambda Function Configuration

Storage

The optimal storage solution for a particular system will vary based on the kind of access method (block, file, or object) you use, patterns of access (random or sequential), throughput required, frequency of access (online, offline, archival), frequency of update (WORM, dynamic), and availability and durability constraints.

In AWS storage is virtualized, and there are a number of different storage types. This makes it easier to match your storage methods more closely with your needs, and it also offers storage options that are not easily achievable with on-premises infrastructure. For example, Amazon Simple Storage Service (S3) is designed for 11 nines of durability. When you use cloud storage, you can change from using magnetic hard drives (HDDs) to solid state drives (SSDs), and easily move virtual drives from one instance to another in seconds.

Characteristics

When you select a storage solution, you should consider the different characteristics that you require, such as shareability, file size, cache size, latency, throughput, and persistence of data. Then match the requirements you want to the AWS service that best fits your needs: Amazon S3, Amazon Elastic Block Store (EBS), Amazon Elastic File System (EFS), or Amazon EC2 instance store.

Performance can be measured by looking at throughput, input/output operations per second (IOPS), and latency. Understanding the relationship between those measurements will help you select the most appropriate storage solution.

Storage	Services	Latency	Throughput	Shareable
Block	EBS, Instance Store	lowest, consistent	Single	Mounted on single instance, copies via snapshots
Filesystem	EFS	low, consistent	Multiple	Many clients
Object	S3	low-latency	Web scale	Many clients

From a latency perspective, if your data is only accessed by one instance then you should use block storage, such as Amazon EBS with Provisioned IOPS. Distributed filesystems such as Amazon EFS generally have a small latency overhead for each file operation, so they should be used where multiple instances need access.

Amazon S3 has features than can reduce latency and increase throughput. You can use cross-region replication (CRR) to provide lower-latency data access to different geographic regions.

From a throughput perspective, Amazon EFS supports highly parallelized workloads (for example, using concurrent operations from multiple threads and multiple Amazon EC2 instances), which enables high levels of aggregate throughput and operations per second. For Amazon EFS, use a benchmark or load test to select the appropriate right performance mode.

Key AWS Services

The key AWS service for storage is Amazon S3, which provides secure, durable, highly-scalable cloud storage. The following services and features are also important:

- **Amazon EBS** provides persistent block storage volumes for use with EC2 instances.
- **Amazon EFS** provides simple, scalable file storage for use with Amazon EC2 instances.
- **Amazon EC2 instance store** provides temporary block-level storage for use with EC2 instances.

Resources

Refer to the following resources to learn more about AWS best practices for storage.

Documentation

- Amazon S3: Request Rate and Performance Considerations
- Amazon EFS: Amazon EFS Performance

- Amazon EBS: I/O Characteristics

Videos

- Amazon EBS Design for Performance

Configuration Options

Storage solutions generally have configuration options that allow you to optimize for the type of workload.

Amazon EBS provides a range of options that allow you to optimize storage performance and cost for your workload. These options are divided into two major categories: SSD-backed storage for transactional workloads, such as databases and boot volumes (performance depends primarily on IOPS), and HDD-backed storage for throughput-intensive workloads such as MapReduce and log processing (performance depends primarily on MB/s).

SSD-backed volumes include the highest performance Provisioned IOPS SSD for latency-sensitive transactional workloads and General Purpose SSD that balance price and performance for a wide variety of transactional data.

Amazon S3 Transfer Acceleration enables fast transfer of files over long distances between your client and your Amazon S3 bucket. Transfer Acceleration leverages Amazon CloudFront globally distributed edge locations to route data over an optimized network path. For a workload in an Amazon S3 bucket that has intensive GET requests you should use Amazon S3 with Amazon CloudFront. When uploading large files you should use multi-part uploads with multiple parts uploading at once to help maximize network throughput.

Access Patterns

How you access data will affect how the storage solution performs. Select the storage solution that aligns best to your access patterns, or consider changing your access patterns to align with the storage solution to maximize performance.

Creating a RAID 0 (zero) array allows you to achieve a higher level of performance for a file system than you can provision on a single volume. Consider using RAID 0 when I/O performance is more important than fault

tolerance. For example, you could use it with a heavily used database where data replication is already set up separately.

Amazon EBS HDD-backed volumes include Throughput-Optimized HDD for frequently accessed, throughput-intensive workloads and the lowest cost Cold HDD for less frequently accessed data.

You can further optimize for how you use storage systems by using techniques covered in the *Partitioning or Sharding* section of the Trade-Offs topic in this whitepaper.

Key AWS Services

The key AWS services for storage solutions are Amazon EBS, Amazon S3, and Amazon EFS which provide block, object, and filesystem storage solutions. These services have configuration options that further allow you to optimize your storage solution. The access patterns your components have should influence the storage solution you select.

Resources

Refer to the following resources to learn more about AWS best practices for storage.

Documentation

- Storage

Database

The optimal database solution for a particular system can vary based on requirements for availability, consistency, partition tolerance, latency, durability, scalability, and query capability. Many systems use different database solutions for different subsystems and enable different features to improve performance. Selecting the wrong database solution and features for a system can lead to lower performance efficiency.

Although a workload's database approach (RDBMS, NoSQL, etc.) has significant impact on performance efficiency, it is often an area that is chosen according to organizational defaults rather than through using a data-driven approach. As with storage, it is critical to consider the access patterns of your workload, and also to consider if other non-database solutions could solve the problem more

efficiently (e.g., using an object store such as Amazon S3, a search engine, or a data warehouse).

Characteristics

Consider the different characteristics you require (e.g., availability, consistency, partition tolerance, latency, durability, scalability, and query capability) so that you can select the most performant database approach to use (relational, No-SQL, warehouse, in-memory).

In AWS there are services for each approach, you should consider using different services for different types of data.

- **Amazon Relational Database Service (RDS)** provides a fully managed relational database. For information that is highly related use Amazon RDS.
- **Amazon DynamoDB** is a fully managed NoSQL database that provides single-digit millisecond latency at any scale. For Internet-scale information, such as user profiles, use Amazon DynamoDB.
- **Amazon Redshift** is a managed petabyte-scale data warehouse that allows you to change the number or type of nodes as your performance or capacity needs change. Use Amazon Redshift when you need SQL operations that will scale.
- **Amazon ElastiCache** is a web service that makes it easy to deploy, operate, and scale an in-memory data store or cache in the cloud. Use Amazon ElastiCache when you want to improve the performance of web applications by retrieving information from fast, managed, in-memory data stores.
- **Amazon CloudSearch** is a managed service in the AWS Cloud that makes it simple and cost-effective to set up, manage, and scale a search solution for your website or application. Use Amazon CloudSearch when you want search or reporting with low latency and high throughput.

Configuration Options

Database solutions generally have configuration options that allow you to optimize for the type of workload. Consider the configuration options for you selected database approach such as storage optimization, database level settings, memory, and cache.

- **Amazon Relational Database Service (RDS)** provides read replicas (not available for Oracle or SQL Server) to scaleout read performance, SSD-backed storage options, and Provisioned IOPS, as well as database-level settings.
- **Amazon DynamoDB** provides throughput and storage scaling, and secondary indexes to allow you to efficiently query on any attribute (column). You can also *project* attributes (copied from the table into the index) to improve performance.
- **Amazon Redshift** allows you to change the number or type of nodes in your data warehouse and scale up all the way to a petabyte or more of compressed user data. Dense Compute (DC) nodes allow you to create very high performance data warehouses using fast CPUs, large amounts of RAM, and solid-state disks (SSDs).
- **Amazon ElastiCache** with Memcached supports sharding to scale in-memory cache with multiple nodes. Amazon ElastiCache for Redis includes clustering, with multiple shards forming a single in-memory key-value store that is tebibytes (TiBs) in size, plus read replicas per shard for increased data access performance.
- **Amazon CloudSearch** offers powerful auto scaling for all search domains. As your data or query volume changes, Amazon CloudSearch can scale your search domain's resources up or down as needed. You can control scaling if you know that you need more capacity for bulk uploads or are expecting a surge in search traffic.

Access Patterns

The way that you access data will affect how the database solution performs. Select the database solution that aligns best to your access patterns, or consider changing your access patterns to align with the storage solution to maximize performance.

Optimize how you use database systems based on your access pattern (e.g., indexes, key distribution, partition, or horizontal scaling).

- **Amazon Relational Database Service (RDS)** ensures that you have indexes for your critical searches, and materialized views where supported. You can also add indexes to read replicas to improve query performance.

- **Amazon DynamoDB** manages table partitioning for you automatically, adding new partitions if necessary and distributing provisioned throughput capacity evenly across them. To ensure that your load is evenly distributed across partitions, you should design for uniform data access across items in your tables.
- **Amazon Redshift** will provide the best performance if you specify sort keys, distribution keys, and column encodings as these can significantly improve storage, IO, and query performance.
- **Amazon ElastiCache** with the Memcache engine will use multiple ElastiCache nodes efficiently if you distribute your cache keys across the nodes. You should configure your clients to use consistent hashing so that when nodes are added or removed that this does not lead to excessive cache misses due to keys being moved.
- **Amazon CloudSearch** requests could be resource intensive to process, which can have an impact on the performance and cost of running your search domain. You should fine-tune your search requests to help reduce the processing overhead.

To further optimize your use of database systems, explore the techniques covered in the *Partitioning or Sharding* section of the Trade-Offs topic in this whitepaper.

Key AWS Services

The key AWS services for database solutions are Amazon RDS, Amazon DynamoDB, and Amazon RedShift which provide relational, NoSQL, and data warehouse solutions. These services have configuration options that further allow you to optimize your storage solution. The access patterns of your components should influence the storage solution you select.

Resources

Refer to the following resources to learn more about AWS best practices for databases.

Documentation

- Cloud Databases with AWS

Network

The optimal network solution for a particular system will vary based on latency, throughput requirements, and so on. Physical constraints such as user or on-premises resources will drive location options, which can be offset using edge techniques or resource placement.

In AWS, networking is virtualized and is available in a number of different types and configurations. This makes it easier to match your networking methods more closely with your needs. AWS offers product features (e.g., enhanced networking instance types, Amazon EBS optimized instances, Amazon S3 transfer acceleration, and dynamic Amazon CloudFront) to optimize network traffic. AWS also offers networking features (e.g., Amazon Route 53 latency routing, Amazon Virtual Private Cloud (VPC) endpoints, and AWS Direct Connect) to reduce network distance or jitter.

Location

The AWS Cloud infrastructure is built around Regions and Availability Zones. A Region is a physical location in the world having multiple Availability Zones. Availability Zones consist of one or more discrete data centers, each with redundant power, networking, and connectivity, housed in separate facilities. These Availability Zones offer you the ability to operate production applications and databases that are more highly available, fault tolerant, and scalable than would be possible from a single data center.

Choose the appropriate Region or Regions for your deployment based on some key elements:

- **Where your users are located**: Choosing a Region close to the users of the application ensures lower latency when they use the application.
- **Where your data is located**: For data-heavy applications, the major bottleneck in latency is when data is transferred to the computing part of the application. Application code should execute as close to the data as possible.
- **Other constraints**: Take into account constraints such as security and compliance.

Placement Groups

Amazon EC2 provides placement groups for networking. A placement group is a logical grouping of instances within a single Availability Zone. Using placement groups with supported instance types enables applications to participate in a low-latency, 10-gigabits-per-second (Gbps) network. Placement groups are recommended for applications that benefit from low network latency, high network throughput, or both. Using placement groups has the benefit of lowering jitter in network communications.

Edge Locations

Latency-sensitive services are delivered at *the edge* using a global network of edge locations. These edge locations commonly provide services such as Content Delivery Network (CDN) and Domain Name System (DNS). By having these services at the edge, they can respond with low latency to requests for content or DNS resolution. These services can also provide geographic services such as Geo Targeting of content (providing different content based on end-users location), or latency based routing to direct end-users to the nearest region (minimum latency).

Amazon CloudFront is a global CDN that can be used to accelerate both static content such as images, scripts, and videos, as well as dynamic content such as APIs or web applications. It relies on a global network of edge locations that will cache the content and provide high-performance network connectivity to our users. Amazon CloudFront also accelerates many other features such as content uploading and dynamic applications, making it a performance addition to all applications serving traffic over the Internet.

Amazon Route 53 is a highly available and scalable cloud DNS web service. It's designed to give developers and businesses an extremely reliable and cost-effective way to route end users to Internet applications by translating names, like www.example.com, into numeric IP addresses, like 192.0.2.1, that computers use to connect to each other. Amazon Route 53 is fully compliant with IPv6 as well.

You should use edge services to reduce latency and to enable caching of content. You need to ensure that you have configured cache control correctly for both DNS and HTTP/HTTPS to gain the most benefit from these approaches.

Product Features

In cloud computing networking is critical, and therefore services commonly offer features to optimize network performance. You should consider product features such as EC2 instance network capability, enhanced networking instance types, Amazon EBS optimized instances, Amazon S3 Transfer Acceleration, and Dynamic Amazon CloudFront to optimize network traffic.

Amazon S3 Content Acceleration is a feature that lets external users benefit from the networking optimizations of Amazon CloudFront to upload data to Amazon S3. This makes it easy to transfer large amounts of data from remote locations that don't have dedicated connectivity to the AWS Cloud.

Amazon EC2 instances can also leverage enhanced networking. Enhanced networking uses single root IO virtualization (SR-IOV) to provide high-performance networking capabilities on supported instance types. SR-IOV is a method of device virtualization that provides higher IO performance and lower CPU utilization than traditional virtualized network interfaces. Enhanced networking provides higher bandwidth, higher packet per second (PPS) performance, and consistently lower inter-instance latencies.

Amazon Elastic Network Adapters (ENA) provide further optimization by delivering 20 Gbps of network capacity for your instances within a single placement group.

Amazon EBS–optimized instances use an optimized configuration stack and provide additional, dedicated capacity for Amazon EBS IO. This optimization provides the best performance for your EBS volumes by minimizing contention between Amazon EBS IO and other traffic from your instance.

Networking Features

When architecting your solution in the cloud you should consider networking features to reduce network distance or jitter.

Latency-Based Routing (LBR) for Amazon Route 53 helps you improve your application's performance for a global audience. LBR works by routing your customers to the AWS endpoint (for EC2 instances, Elastic IP addresses, or ELB load balancers) that provides the fastest experience based on actual performance measurements of the different AWS Regions where your application is running.

AWS Direct Connect provides dedicated connectivity to the AWS environment, from 50 Mbps up to 10 Gbps. This gives you managed and controlled latency and provisioned bandwidth so your applications can connect easily and in a performant way to other environments. Using one of the AWS Direct Connect partners, you can have end-to-end connectivity from multiple environments, providing an extended network with consistent performance.

Amazon VPC endpoints provide reliable connectivity to AWS services (for example, Amazon S3) without requiring an Internet gateway or a Network Address Translation (NAT) instance. Using VPC endpoints, the data between your VPC and another AWS service is transferred within the Amazon network, helping protect your instances from Internet traffic.

Key AWS Services

The key AWS service for networking solutions is Amazon Route 53, which provides latency-based routing. In addition, the use of Amazon VPC endpoints and AWS Direct Connect can reduce network distance or jitter.

Resources

Refer to the following resources to learn more about AWS best practices for networking.

Documentation

- Networking Products with AWS

Review

When you first architect your solution, you have a fixed set of options to choose from. Then, over time new technologies and approaches become available that could improve the performance of your architecture. In the AWS Cloud because your infrastructure is code it is much easier to experiment with new features and services.

Using AWS, you can take advantage of our continual innovation, which is driven by customer need. We reduce prices and release new regions, edge locations, services, and features regularly. Any of these new releases could positively improve the performance efficiency of your architecture.

After you have identified your architectural approach, you should use data to drive your selection of resource types and configuration options. You can obtain this data using benchmarking and load testing.

To adopt a data-driven approach to architecture you should implement a performance review process that considerers the following:

- **Infrastructure as code:** Define your infrastructure as code using approaches such as AWS Cloud Formation templates. The use of templates allows you to place your infrastructure into source control alongside your application code and configurations. This enables you to apply the same software development practices to your infrastructure so you can iterate rapidly.
- **Deployment Pipeline:** Use a continuous integration/continuous deployment (CI/CD) pipeline (e.g., source code repository, build systems, deployment, and testing automation) to deploy your infrastructure. This enables you to deploy in a repeatable, consistent and low-cost fashion as you iterate.
- **Well defined metrics:** Set up your metrics and monitoring to capture key performance indicators (KPIs). We recommend that you use both technical and business metrics. For website or mobile apps key metrics are capturing time to first byte or rendering. Other generally applicable metrics include thread count, garbage collection rate, and wait states. Business metrics, such as the aggregate cumulative cost per request, can alert you to ways to drive down costs. Carefully consider how you plan to interpret metrics. For example, you could choose the maximum or 99th percentile instead of the average.
- **Performance test automatically:** As part of your deployment process automatically trigger performance tests after the quicker running tests have passed successfully. The automation should create a new environment, set up initial conditions such as test data, and then execute a series of benchmarks and load tests. Results from these tests should be tied back to the build so you can track performance changes over time. For long running tests you can make this part of the pipeline asynchronous from the rest of the build. Alternatively, you could execute performance tests overnight using Spot Instances.

- **Load generation:** You should create a series of test scripts that replicate synthetic or prerecorded user journeys. These scripts should be idempotent and not coupled, and you might need to include "pre-warming" scripts to yield valid results. As much as possible you want your test scripts to replicate the behavior of usage in production. You can use software or software as a service (SaaS) solutions to generate the load. Consider using AWS Marketplace solutions and Spot Instances; they can be cost-effective ways to generate the load.
- **Performance visibility:** Key metrics should be visible to your team, especially metrics against each build version. This allows you to see any significant positive or negative trend over time. You should also display metrics on the number of errors or exceptions to make sure you are testing a working system.
- **Visualization:** Use visualization techniques that make it clear where performance issues, hot spots, wait states, or low utilization is occurring. Overlay performance metrics over architecture diagrams-- call graphs or code can help identify issues more quickly.

This performance review process can be implemented as a simple extension of your existing deployment pipeline and then evolved over time as your testing requirements become more sophisticated. For future architectures you should be able generalize your approach and reuse the same process and artifacts.

When architectures perform badly this is normally because a performance review process has not been put into place or is broken. If your architecture is performing badly putting this performance review process in place will allow you to apply Deming's plan-do-check-act (PDCA) cycle to drive iterative improvement.

Benchmarking

Benchmarking uses synthetic tests to provide you with data on how components perform. In this section we discuss how to use benchmarking to drive improvements in your workloads. We don't cover the use of benchmarking to compare different vendor's products or implementations.

Benchmarking is generally quicker to set up than load testing and is used when you want to evaluate the technology for a particular component. Benchmarking

is often used at the start of a new project, when you don't yet have a whole solution that you could load test.

For benchmarking, you should follow the performance review process. However, your deployment pipeline will just consist of the benchmark tests themselves. You can either build your own custom benchmark tests, or you can use an industry standard test, such as TPC-DS (to benchmark your data warehousing workloads). Industry benchmarks are helpful when you are comparing different environments. Custom benchmarks are useful for targeting specific types of operations that you expect to make in your architecture.

With benchmarking, it is generally more important to pre-warm your test environment to ensure valid results. You should run the same benchmark multiple times to be sure you've captured any variance over time.

Since benchmarks are generally faster to run than load tests, they can be used earlier in the deployment pipeline to provide faster feedback on performance deviations to your team. When you evaluate a significant change in a component or service, a benchmark can be a quick way to see if you can justify the effort to make the change based on the benchmarked difference. Benchmarking should be used in conjunction with load testing because load testing will tell you how your **whole** workload will perform in production.

Key AWS Services

The key AWS services supporting benchmarking are AWS CodeDeploy and AWS Cloud Formation. Use these services to automate the testing of your infrastructure in a repeatable fashion.

Resources

Refer to the following resources to learn more about AWS best practices for benchmarking.

Videos

- Performance Channel
- Performance Benchmarking on AWS

Documentation

- Amazon S3 Performance Optimization

- Amazon EBS Volume Performance
- AWS CodeDeploy
- AWS CloudFormation

Load Testing

Load testing uses your *actual* workload so you can see how your whole solution performs in a production environment. Load tests should be done using synthetic or sanitized versions of production data (remove sensitive or identifying information). Use replayed or pre-programmed user journeys through your application at scale that exercise your entire architecture. As part of your delivery pipeline you can automatically carry out load tests and compare against pre-defined KPIs and thresholds to ensure that you continue to get the performance you require.

Amazon Cloud Watch can collect metrics across the resources in your architecture. You can also collect and publish custom metrics to surface business or derived metrics. Use CloudWatch to set alarms that indicate when thresholds are breached to signal that a test is outside of expected performance.

Using AWS services you can run production-scale environments to test your architecture aggressively. Since you only pay for the test environment when it is needed, you can carry out full-scale testing at a fraction of the costs of using an on-premises environment. You should take advantage of the AWS Cloud to test your workload to see where it fails to scale or scales in a non-linear way. You can use Spot Instances to generate loads at low cost and discover bottlenecks before they are experienced in production.

When you write critical user stories for your architecture you should include performance requirements, such as specifying how quickly each critical story should execute. For these critical stories you should implement additional scripted user journeys to ensure you have visibility into how these stories perform against your requirement.

Where load tests take considerable time to execute you should parallelize them using multiple copies of your test environment. Your costs will be similar, but your testing time will be reduced. (It costs the same to run one EC2 instance for 100 hours as it does to run 100 instances for one hour.) You can also lower the

costs of these tests by using Regions that have lower costs than the Regions you use for production.

The location of your load test clients should reflect the geographic spread of your end users. Also you will need to aware of any caching (DNS, Amazon CloudFront) to ensure that you replicate actual performance.

Key AWS Services

The key AWS service supporting load testing is Amazon CloudWatch, which allows you to collect metrics on how your whole architecture is performing during the load test. With CloudWatch you can also create custom and business metrics.

Resources

Refer to the following resources to learn more about AWS best practices for load testing.

Documentation

- Load Testing CloudFront

Monitoring

Once you have implemented your architecture you will need to monitor its performance so that you can remediate any issues before your customers are aware of them. Monitoring metrics should be used to raise alarms when thresholds are breached. The alarm can trigger automated action to work around any badly performing components.

When you use AWS, Amazon CloudWatch provides the ability to monitor and send notification alarms. You can use automation to work around performance issues by triggering actions through Amazon Kinesis, Amazon Simple Queue Service (SQS), and AWS Lambda.

Active and Passive

Monitoring solutions fall into two types: active monitoring (AM) and passive monitoring (PM). AM and PM complement each other to give you a full view of how your workload is performing.

Active monitoring simulates user activity in scripted user journeys across critical paths in your product. AM should be continuously performed in order to test the performance and availability of a workload. AM complements PM by being continuous, lightweight, and predictable. It can be run across all environments (especially pre-production environments) to identify problems or performance issues before they affect end users.

Passive monitoring is commonly used with web-based workloads. PM collects performance metrics from the browser (nonweb-based workloads can use a similar approach). You can collect metrics across all users (or a subset of users), geographies, browsers, and device types. You should use PM to understand these issues:

- **User experience performance**: PM provides you with metrics on what your users are experiencing, which gives you a continuous view into how production is working, as well as a view into the impact of changes over time.
- **Geographic performance variability**: If a workload has a global footprint and users access the application from all around the world, using PM can enable you to spot a performance problem affecting users in a specific geography.
- **The impact of API use**: Modern workloads use internal APIs and third-party APIs. PM provides the visibility into the use of APIs so you can identify performance bottlenecks that originate not only from internal APIs but also from third-party API providers.

Phases

Monitoring at AWS consists of five distinct phases which are explained in more detail in *The Reliability Pillar of the AWS Well-Architected Framework* whitepaper:

1. Generation – scope of monitoring, metrics and thresholds
2. Aggregation – creating a complete view from multiple source
3. Real-time processing and alarming - recognizing and responding
4. Storage – data management and retention policies
5. Analytics – dashboards, reporting, and insights

Amazon CloudWatch is a monitoring service for AWS Cloud resources and the workloads that run on AWS. You can use Amazon CloudWatch to collect and track metrics, collect and monitor log files, and set alarms. Amazon CloudWatch can monitor AWS resources such as Amazon EC2 instances and Amazon RDS DB instances, as well as custom metrics generated by your applications and services, and any log files your applications generate. You can use Amazon CloudWatch to gain system-wide visibility into resource utilization, application performance, and operational health. You can use these insights to react quickly and keep your application running smoothly. Amazon CloudWatch dashboards enable you to create re-usable graphs of AWS resources and custom metrics so you can monitor operational status and identify issues at a glance.

Ensuring that you do not see too many false positives or are overwhelmed with data is key to an effective monitoring solution. Automated triggers avoid human error and can reduce the time to fix problems. Plan for *game days* where simulations are conducted in the production environment, to test your alarming solution and ensure that it correctly recognizes issues.

Key AWS Services

The key AWS service that supports monitoring is **Amazon CloudWatch**, which allows easy creation of alarms that can trigger scaling actions. The following services and features are also important:

- **Amazon S3**: Acts as the storage layer, and allows for lifecycle policies and data management.
- **Amazon EMR**: Can analyze metrics to help you monitor performance.

Resources

Refer to the following resources to learn more about AWS best practices for monitoring to promote Performance Efficiency.

Video

- AWS re:Invent 2015 | (DVO315) Log, Monitor and Analyze your IT with Amazon CloudWatch
- AWS re:Invent 2015 | (BDT312) Application Monitoring: Why Data Context Is Critical

Documentation

- CloudWatch Documentation

Trade-Offs

When you architect solutions, think about trade-offs so you can ensure an optimal approach. Depending on your situation you could trade consistency, durability, and space versus time or latency, to deliver higher performance.

Using AWS, you can go global in minutes and deploy resources in multiple locations across the globe to be closer to your end users. You can also dynamically add read-only replicas to information stores, such as database systems to reduce the load on the primary database. AWS also offers caching solutions such as Amazon ElastiCache, which provides an in-memory data store or cache, and Amazon CloudFront, which caches copies of your static content closer to end users.

The following sections detail some of the trade-offs you can make and how you can implement them.

Technique	Applies To	Use	Gains
Caching	read-heavy	Space (Memory)	Time
Sharding or Partitioning	write-heavy	Size & Complexity	Time
Compression	large data	Time	Space
Buffering	many requests	Space & Time	Efficiency

Caching

Most workloads rely on a dependent component such as a service or database that offers a source of truth or a consolidated view of data. Generally these components in an architecture are harder to scale, and represent a significant proportion of the cost of the workload. You can improve performance efficiency by using caching to trade off against freshness or memory used. These techniques generally update on an asynchronous or periodic basis. The trade-off is that your data is not always fresh or therefore not always consistent with the source of truth.

Application Level

You can make this trade off at a code level by using application-level caches or memorization. At the architectural level you should us Amazon ElastiCache, which is an in-memory data store or cache supporting the Redis and Memcached engines. When requests are cached, execution time is reduced. This provides a way to scale horizontally through the caching layer, and reduces load on your most heavily used components.

Database level

Database replicas enhance performance databases by replicating all changes to the master databases to read replicas (not available for Oracle or SQL Server). This replication makes it possible to scale out beyond the capacity constraints of a single database for read-heavy database workloads.

Amazon RDS provides read replicas as a fully managed service. You can create one or more replicas of a given source database and serve high-volume application read traffic from multiple copies of your data, thereby increasing aggregate read throughput. You should also add additional indexes to the read replica, where the database engine supports it. For example you could add more indexes to the MySQL read replica. For latency-sensitive workloads you should use the Multi-AZ feature to specify which Availability Zone the read replica should be in to reduce cross-Availability Zone traffic.

Geographic Level

Another example of caching is the use of a Content Delivery Network (CDN), using a CDN is a good way to reduce latency for clients. CDNs should be used to store static content and to accelerate dynamic content. You should consider the use of a CDN for your APIs; even dynamic content can benefit through the use of network optimization methods.

Amazon CloudFront can be used to deliver your entire website, including dynamic, static, streaming, and interactive content using a global network of edge locations. Requests for your content are automatically routed to the nearest edge location, so content is delivered with the best possible performance. Amazon CloudFront is optimized to work with other Amazon Web Services, like Amazon S3, Amazon EC2, Elastic Load Balancing, and Amazon Route 53. Amazon CloudFront also works seamlessly with any non-AWS origin server, which stores the original, definitive versions of your files.

Key AWS Services

The key AWS services for caching solutions are Amazon ElastiCache, which provides a general-purpose application cache, and Amazon CloudFront, which allows you to cache information closer to your users.

Resources

Refer to the following resources to learn more about AWS best practices for caching.

Documentation

- Best Practices for Implementing Amazon ElastiCache
- AWS CloudFormation Best Practices

Videos

- Turbocharge your apps with Amazon ElastiCache

Partitioning or Sharding

When using technologies, such as relational databases, that require a single instance due to consistency constraints, you can only scale vertically (by using higher specification instances and storage features). When you hit the limits of vertical scaling, you can use a different approach called data partitioning or *sharding*. With this model, data is split across multiple database schemas, each running in its own autonomous primary DB instance.

Amazon RDS removes the operational overhead of running multiple instances, but sharding will still introduce complexity to the application. The application's data access layer will need to be modified to have awareness of how data is split so that it can direct queries to the right instance. (You can use a proxy or routing mechanism to remove caching code from the application, or implement it in a data access layer.) In addition, any schema changes will have to be performed across multiple database schemas, so it is worth investing some effort to automate this process.

NoSQL database engines will typically perform data partitioning and replication to scale both the reads and the writes in a horizontal fashion. They do this

transparently without the need of having the data partitioning logic implemented in the data access layer of your application. Amazon DynamoDB in particular manages table partitioning for you automatically, adding new partitions as your table grows in size or as read- and write-provisioned capacity changes.

Partitioning or sharding provides a way to scale write-heavy workloads, but requires that data is evenly distributed and evenly accessed across all partitions or shards. It can introduce complexity in relational database solutions, while NoSQL solutions generally trade consistency to deliver this.

Key AWS Services

The key AWS service for partitioning or sharding is Amazon DynamoDB, which manages table partitioning for you automatically.

Resources

Refer to the following resources to learn more about AWS best practices for partitioning and sharding.

Documentation

- Best Practices for DynamoDB

Videos

- AWS re:Invent 2015 | (DAT401) Amazon DynamoDB Deep Dive

Compression

Compressing data trades computing time against space, and can greatly reduce storage and networking requirements. Compression can apply to file systems, data files, and web resources such as stylesheets and images, but also to dynamic responses such as APIs.

Amazon CloudFront supports compression at the edge. The source system can serve resources in a standard fashion, and the CDN will automatically compress the resources if and only if the web clients can support it.

When transferring large quantities of information into or out of the cloud you should consider non-network based solutions. AWS Snowball is a petabyte-scale data transport solution that uses secure appliances to transfer large amounts of data into and out of the AWS Cloud. Using AWS Snowball addresses common challenges with large-scale data transfers, including high network costs, long transfer times, and security concerns.

Amazon Redshift uses compression with columnar data storage. Instead of storing data as a series of rows, Amazon Redshift organizes the data by column. Columnar data stores can be compressed much more than row-based data stores because similar data is stored sequentially on disk. Amazon Redshift employs multiple compression techniques and can often achieve significant compression relative to traditional relational data stores. When loading data into an empty table, Amazon Redshift automatically samples your data and selects the most appropriate compression scheme.

Key AWS Services

The key AWS service for compression is Amazon CloudFront, which supports compression at the edge.

Resources

Refer to the following resources to learn more about AWS best practices for compression.

Documentation

- Amazon CloudFront: Serving Compressed Files
- Amazon RedShift: Columnar Storage
- AWS Snowball: What Is AWS Snowball?

Buffering

Buffering uses a queue to accept messages (units of work) from producers. For resiliency the queue should use durable storage. A *buffer* is a mechanism to ensure that applications can communicate with each other when they are running at different rates over time. Messages can then be read by consumers, which allows the messages to run at the rate that meets the consumers' business

requirements. By using a buffer, you can decouple the throughput rate of producers from that of consumers. You don't have to worry about producers having to deal with data durability and *backpressure* (where producers slow down because their consumer is running slowly).

When you have a workload that generates significant write load that doesn't need to be processed immediately, you can use a buffer to smooth out demands on consumers.

On AWS, you can choose from multiple services to implement a buffering approach. Amazon Simple Queue Service (SQS) provides a queue that allows a single consumer to read individual messages. Amazon Kinesis provides a stream that allows many consumers to read the same messages.

An Amazon SQS *queue* is a reliable, scalable, and fully managed repository for messages that are awaiting processing. Amazon SQS enables applications to quickly and reliably queue messages that one component in the application generates to be consumed by another component. For this approach, a queue is created that producers post messages to, and which resides at a well-known address.

To reduce cost, producers post messages using Amazon SQS batch API actions. Consumers read messages from the well-known queue using a fixed-sized Auto Scaling group of EC2 instances to cope with instance failures. *Long polling* lets you retrieve messages from your Amazon SQS queue as soon as they become available. Use long polling to reduce the cost of retrieving messages. Amazon SQS uses the *message visibility* feature to hide messages that have been read. Message visibility reduces the risk that you might process the same message twice, though you should be aware that by default Amazon SQS *at-least-once delivery* means that you can receive a message more than once.

You can use Amazon SQS first in/first out (FIFO) queues if you need exactly-once processing. Message visibility allows for messages that have not been processed to reappear (for example, in the case of an instance failure). For long-running tasks you can extend the visibility time-out, and your application will need to delete messages after they have been processed.

An alternative approach is to use Amazon Kinesis to provide buffering. It differs from Amazon SQS in that it allows multiple *consumers* to read the same

message at any one time. However, a single message will be read using the Kinesis Client Library, or AWS Lambda, and will be delivered to one-and-only-one consumer for an *Application*, which is a name provided by the consumer when it connects to the Stream. Different *Applications* can all consume the same messages concurrently, providing a "Publish and Subscribe" model.

When architecting with a buffer keep in mind two key considerations: First, what is the acceptable delay between producing the work and consuming the work? Second, how do you plan to handle duplicate requests for work?

To optimize the speed with which work items are consumed by more consumers, you can use Amazon EC2 Spot Instances. Using Spot Instances, you can bid on spare Amazon EC2 computing capacity. To handle duplicate messages with Amazon SQS we recommend using Amazon SQS (FIFO) queues, which provide exactly-once processing.

For AWS Kinesis you should consider provisioning a table in Amazon DynamoDB that can track the work items that have already been successfully processed. An alternative to deduplication is to implement idempotent processing; this should be used where you require higher throughput processing. *Idempotence* is an application logic pattern that allows a consumer to process a message multiple times, but when a message is subsequently processed it has no effect on downstream systems or storage.

Key AWS Services

The key AWS service for buffering is Amazon SQS, which allows you to decouple producers from consumers using a queue.

Resources

Refer to the following resource to learn more about AWS best practices for buffering.

Documentation

- Best Practices for Amazon SQS

Conclusion

Achieving and maintaining performance efficiency requires a data-driven approach. You should actively consider access patterns and trade-offs that will allow you to optimize for higher performance. Using a review process based on benchmarks and load tests will allow you to select the appropriate resource types and configurations. Treating your infrastructure as code will enable you to rapidly and safely evolve your architecture, while you use data to make fact-based decisions about your architecture. Putting in place a combination of active and passive monitoring will ensure that the performance of your architecture does not degrade over time.

AWS strives to help you build architectures that have performance efficiency while delivering business value. To make your architectures truly perform, you should use the tools and techniques discussed in this paper.

Contributors

The following individuals and organizations contributed to this document:

- Philip Fitzsimons, Sr Manager Well-Architected, Amazon Web Services
- Julien Lépine, Solutions Architect, Amazon Web Services
- Ronnen Slasky, Solutions Architect, Amazon Web Services

Further Reading

For additional help, please consult the following sources:

- AWS Well-Architected Framework