AWS Whitepaper

Financial Services Grid Computing on AWS

Financial Services Grid Computing on AWS: AWS Whitepaper

Table of Contents

This whitepaper is for historical reference only. Some content might be outdated and some links might not be available.

Financial Services Grid Computing on AWS

Publication date: **August 9, 2024** (*Document history*)

Financial services organizations rely on high performance computing (HPC) infrastructure grids to calculate risk, value portfolios, and provide reports to their internal control functions and external regulators. The scale, cost, and complexity of this infrastructure is an increasing challenge. Amazon Web Services (AWS) provides a number of services that enable these customers to surpass their current capabilities by delivering results more quickly and at a lower cost than with on-premises resources.

The intended audience for this paper includes grid computing managers, architects, and engineers within financial services organizations who want to improve their service. It describes the key AWS services to consider, some best practices, and includes relevant reference architecture diagrams.

Overview

High performance computing (HPC) in the financial services industry is an ongoing challenge because of the pressures from ever-increasing computational demand across retail, commercial, and investment groups, combined with growing cost and capital constraints. The traditional, on-premises approaches to solving these problems have evolved from centralized, monolithic solutions, to business-aligned clusters of commodity hardware, to modern, multi-tenant grid architectures with centralized schedulers that manage disparate compute capacity.

Cloud concepts such as *capacity on demand* and *pay as you go* pricing models offer new opportunities to teams who run HPC platforms to leverage previously unheard-of flexibility and scale to their businesses. This might take the form of a bursting model, where existing capacity is augmented by the cloud, or it might be through a *lift and shift* approach with new elastic clusters deployed into the cloud.

Historically, the challenge has been to manage a fixed set of on-premises resources, while maximizing utilization and minimizing queuing times. In a cloud-based model with capacity that is effectively unconstrained, the focus shifts away from managing and throttling demand, and towards optimizing supply. With this model, decisions become more granular and tailored to each customer, and focus on *how fast* and at *what cost*, with the ability to make adjustments as required by the business. With this relatively unconstrained capacity, concepts such as queuing and prioritization become irrelevant, as clients are able to submit calculation requests and have

them serviced immediately. This also results in upstream consumers increasingly expecting and demanding near instantaneous processing of their workloads at any scale.

Initial cloud migrations of HPC platforms are often seen as extensions or evolutions of on-premises grid implementations. However, forward-looking institutions are experimenting with the ever-expanding ecosystem of capabilities enabled by AWS. Some emerging themes include refreshing financial models to run on open-source Linux based operating systems, and exploring the performance benefits of the latest Arm-based central processing units (CPUs) through AWS Graviton. Amazon SageMaker AI increasingly democratizes the use of artificial intelligence/machine learning (AI/ML) techniques, and customers are looking to these tools to enable accelerated development of predictive risk models. Serverless approaches to both scheduling and orchestration are emerging trends with customers looking at solutions including the open-source HTC-Grid project.

For data-heavy calculations, Amazon EMR offers a fully managed, industry-leading cloud big data platform based on standard tooling using directed acyclic graph structures. This topic is explored further in the blog post How to improve FRTB's Internal Model Approach implementation using Apache Spark and Amazon EMR.

As HPC environments move to the cloud, the applications that are associated with them start to migrate too. Risk management systems which drive compute grids quickly become a bottleneck when the downstream HPC platform is unconstrained. By migrating these applications alongside the compute grid, the applications benefit from the elasticity that the cloud provides. In turn, data sources such as market and static data are sourced natively from within the cloud, from the same providers that customers work with today through services such as AWS Data Exchange.

Many of the building blocks required for fully serverless risk management and reporting solutions already exist today within AWS, with services like AWS Lambda for serverless compute and AWS Step Functions to coordinate them. As financial institutions become increasingly familiar and comfortable with these services, it's likely that serverless patterns will become the predominant HPC architectures of the future.

Are you Well-Architected?

The AWS Well-Architected Framework helps you understand the pros and cons of the decisions you make when building systems in the cloud. The six pillars of the Framework allow you to learn architectural best practices for designing and operating reliable, secure, efficient, cost-effective, and sustainable systems. Using the AWS Well-Architected Tool, available at no charge in the AWS

Management Console, you can review your workloads against these best practices by answering a set of questions for each pillar.

In the Financial Services Industry Lens, we focus on best practices for architecting your Financial Services Industry workloads on AWS.

In the HPC Lens, we focus on best practices for architecting your High Performance Computing (HPC) workloads on AWS.

For more expert guidance and best practices for your cloud architecture—reference architecture deployments, diagrams, and whitepapers—refer to the AWS Architecture Center.

Introduction

In general, traditional HPC systems are used to solve complex mathematical problems that require thousands or even millions of CPU hours. These systems are commonly used in academic institutions, biotech, and engineering firms. In banking organizations, HPC systems are used to quantify the risk of given products, trades, or portfolios, which enables traders to develop effective hedging strategies, price trades, and report positions to their internal control functions and ultimately to external regulators. Insurance companies leverage HPC systems in a similar way for actuarial modeling and in support of their own regulatory requirements.

Unpredictable global events, seasonal variation, and regulatory reporting commitments contribute to a mixture of demands on HPC platforms. This includes short, latency-sensitive intraday pricing tasks, near real-time risk measures calculated in response to changing market conditions, or large overnight batch workloads and back-testing to measure the efficacy of new models to historic events. Combined, these workloads can generate hundreds of millions of tasks per day, with a significant proportion running for less than a second. As a result, these workloads are often characterized as *high-throughput computing* problems.

Because of the regulatory landscape, demand for these calculations continues to outpace the progress of Moore's law. Regulations such as the *Fundamental Review of the Trading Book* (FRTB) and *IFRS 17* require even more analysis. In turn, financial services organizations continue to grow their grid computing platforms and increasingly wrestle with the costs associated with purchasing and managing this infrastructure. The blog post *How cloud increases flexibility of trading risk infrastructure for FRTB compliance* explores this topic in greater detail, discussing the challenges of data, compute, and the agility benefits achieved by running these workloads in the cloud.

Risk and pricing calculations in financial services are most commonly embarrassingly parallel or *loosely coupled*, do not require communication between nodes to complete calculations, and broadly benefit from horizontal scalability. Because of this, they are well suited to a shared-nothing architectural approach, in which each compute node is independent from the other.

For example, a financial model based on the Monte Carlo method can create millions of scenarios to be divided across a large number (often hundreds or thousands) of compute nodes for calculation in parallel. Each scenario reflects a different market condition based on a number of variables.

In general, doubling the number of compute nodes allows these tasks to be distributed more widely, which reduces by half the overall duration of the job. Access to increased compute capacity

through AWS allows for additional scenarios and greater precision in the results in a given timeframe. Alternatively, you can use the additional capacity to complete the same calculations in less time.

Financial services firms typically use a third-party grid scheduler to coordinate the allocation of compute tasks to available capacity. Grid schedulers have these features in common:

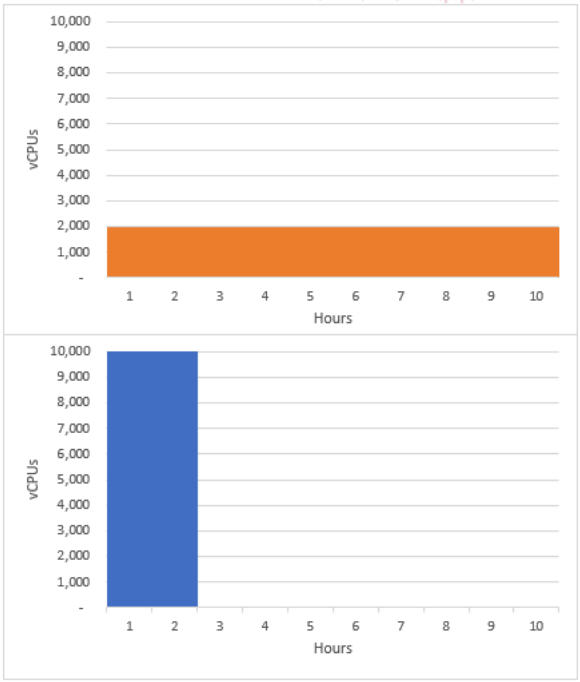
- Scheduling logic to manage the life-cycle of tasks including retry logic, prioritization, and resource allocation. This includes an engine to allow rules to be defined to ensure that certain workloads are prioritized over others in the event that the total capacity of the grid is exhausted. This component is key to the overall throughput of the system, typically needing to place many thousands of tasks per second with the goal of maximizing effective use of the resources.
- Infrastructure orchestration to manage the compute resources, to track which are available and their capabilities. When the grid is making use of cloud compute, this component will be responsible for coordinating scale-out/scale-in events.
- Deployment tools to ensure that software binaries and relevant data are reliably distributed to compute nodes that are allocated a specific task.
- Brokers are typically employed to manage the direct allocation of tasks that are submitted by a client to the compute grid. In some cases, an allocated compute node makes a direct connection back to a client to collect tasks to reduce latency. Brokers are usually horizontally scalable, and are well suited to the elasticity of cloud.

In some cases, the client is another grid node that generates further tasks. Such multi-tier, recursive architectures are not uncommon, but present further challenges for software engineers and HPC administrators who want to maximize utilization while managing risks, such as deadlock, when parent tasks are unable to yield to child tasks.

The key benefit of running HPC workloads on AWS is the ability to allocate large amounts of compute capacity on demand without the need to commit to the upfront and ongoing costs of a large hardware investment. Capacity can be scaled minute by minute according to your needs at the time. This avoids pre-provisioning of capacity according to some estimate of future peak demand. Because AWS infrastructure is charged by consumption of CPU-hours, it's possible to complete the same workload in less time, for the same price, by simply scaling the capacity.

The following figure shows two approaches to provisioning capacity. In the first, 2,000 vCPUs are provisioned for ten hours. In the second, 10,000 vCPUs are provisioned for two hours. In a vCPU-

hour billing model, the overall cost is the same, but the latter produces results in one fifth of the time.



Two approaches to provisioning 20,000 vCPU-hours of capacity

Developers of the analytics calculations used in HPC applications can use the latest CPUs, graphics processing units (GPUs), and field-programmable gate arrays (FPGAs) available through the many Amazon EC2 instance types. This drives efficiency-per-core, and differs from on-premises grids that tend to be a mixture of infrastructure, which reflects historic procurement rather than current needs.

Diverse pricing models offer flexibility to these customers. For example, Amazon EC2 Spot Instances can reduce compute costs by up to 90%. These instances are occasionally interrupted by AWS, but HPC schedulers can typically react to these events and reschedule tasks accordingly.

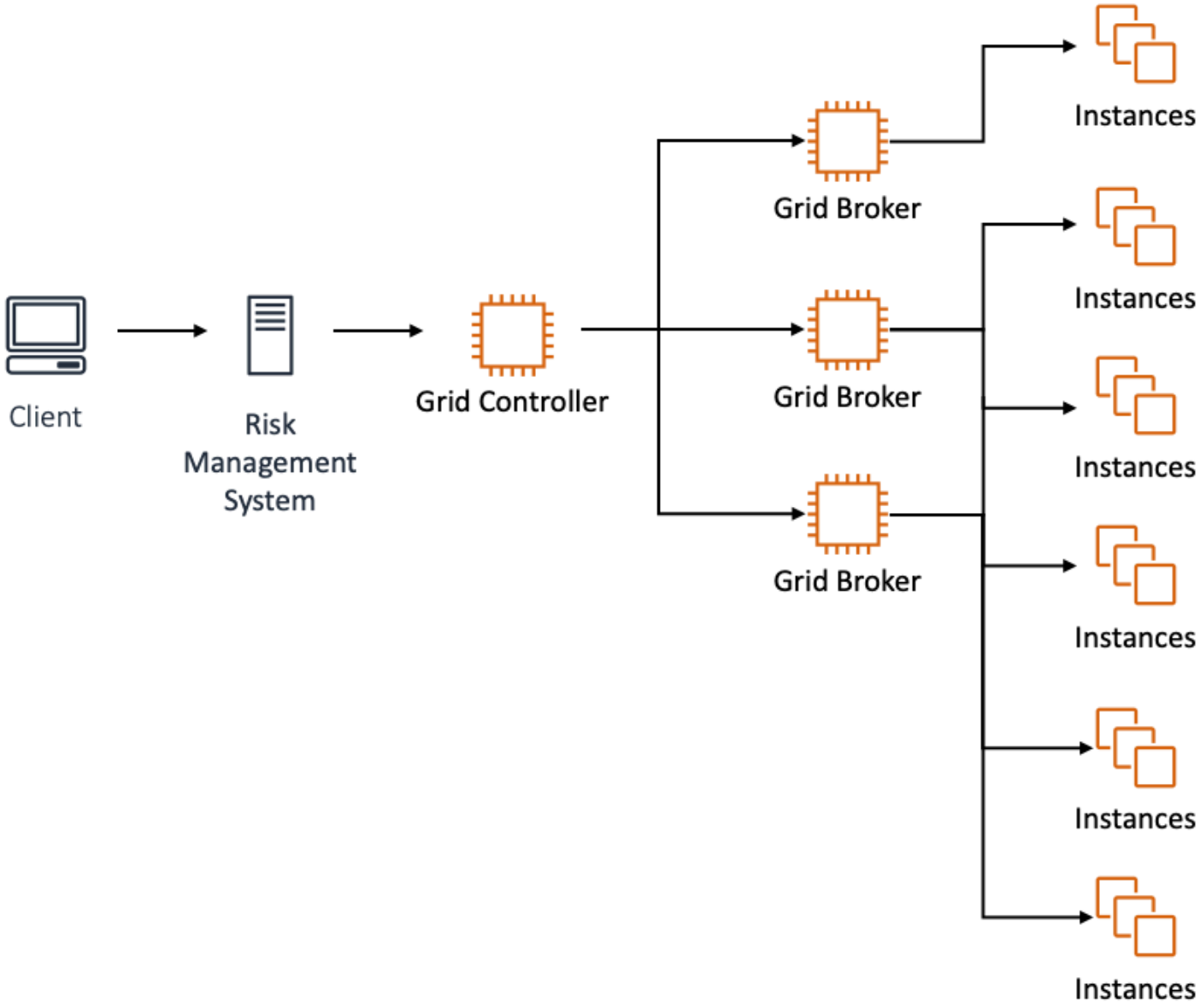
This document includes several recommended approaches to building HPC systems in the cloud, and highlights AWS services that are used by financial services organizations to help to address their compute, networking, storage, and security requirements.

Grid computing on AWS

A key driver for the migration of HPC workloads from on-premises environments to the cloud is flexibility. AWS offers HPC teams the opportunity to build reliable and cost-efficient solutions for their customers, while retaining the ability to experiment and innovate as new solutions and approaches become available.

HPC teams that want to migrate an existing HPC solution to the cloud, or to build a new solution, should review the AWS Well-Architected Framework which also includes a specific Financial Services Industry Lens with a focus on how to design, deploy, and architect financial services industry (FSI) workloads that promote resiliency, security, and operational performance in line with risk and control objectives. This framework applies to any cloud deployment and seeks to ensure that systems are architected according to best practices. Additionally, the HPC-specific lens document also identifies key elements to help ensure the successful deployment and operation of HPC systems in the cloud.

The following sections include information about AWS services that are most relevant to HPC systems, particularly those that support financial services customers.



A typical HPC architecture with the key components, including the risk management system (RMS), grid controller, grid brokers, and two compute instance pools

Compute and capacity

AWS offers a wide range of Amazon Elastic Compute Cloud (Amazon EC2) instance types, which enable you to select the configuration that is best suited to your needs at any given time. This is a departure from the typical *Bill of Materials* approach, which limits the configurations available on-premises in favor of deployment simplicity. It also offers *evergreening*, which enables you to take

advantage of the latest CPU technologies as they are released without consideration for any prior investment. HPC customers in financial services should consider the following instances types:

- **Amazon EC2 compute-optimized instances** — These instances are optimized for compute-intensive workloads and deliver cost-effective high performance at a low price per compute ratio.
- **Amazon EC2 General-purpose instances** —
  - ***M*** — Commonly used in HPC applications because they offer a good balance of compute, memory, and networking resources.
  - ***Z*** — Offer the highest CPU frequencies with a high memory footprint. These instances might be useful for latency sensitive workloads such as product pricing.
  - ***T*** — Provide a baseline level of CPU performance with the ability to burst to a higher level when required. The use of these instances for HPC workloads can be attractive for some workloads; however, their variable performance profile can result in inconsistent behavior, which might be undesirable.
- **Amazon EC2 memory optimized instances**—
  - ***R*** – These instances offer higher memory-to-CPU ratios and so may be applied to X-Valuation Adjustment (XVA) calculations such as Credit Value Adjustments which typically require additional memory to calculate.
- **Instances with the suffix *a*** have AMD processors, for example, *M7a*.
- **Instances with the suffix *i*** have Intel processors, for example, *M6i*.
- **Instances with the suffix *g*** have Arm-based AWS Graviton processors, for example, *M7g*.
- **Amazon EC2 Accelerated Computing instances** use hardware accelerators, or co-processors, to perform functions such as floating-point number calculations, graphics processing, or data pattern matching, more efficiently than is possible in software running on CPUs.
  - ***P*** — Intended for general-purpose GPU compute applications.
  - ***G*** — Intended for graphical rendering and inferences.
  - ***F*** — Offer customizable hardware acceleration with field programmable gate arrays (FPGAs).
  - ***Trn*** — Powered by AWS Trainium, purpose-built for cost-effective deep learning training.
  - ***Inf*** — Powered by AWS Inferentia, use a custom AI/ML processor from Amazon that provides low-latencey inference.

HPC class instances are designed to maximize the performance of tightly coupled workloads such as computational fluid dynamics (CFD), weather forecasting and molecular analysis.

The latest AWS instances are based on the AWS Nitro System. The Nitro System is collection of AWS-built hardware and software components that enable high performance, high availability, high security, and bare metal capabilities to eliminate virtualization overhead. By selecting Nitro based instances, HPC applications can expect performance levels that are indistinguishable to a bare-metal system while retaining all of the benefits of an ephemeral virtual host (see Bare metal performance with the AWS Nitro System.

Table 1 – Amazon EC2 instance types that are typically used for HPC workloads

Instance Type	Family	Description
General-purpose	M	General-purpose instances
Compute-optimized	C	For compute intensive workloads
Memory-optimized	R	For memory intensive workloads
	X	For memory intensive workloads
	Z	High compute capacity and high memory
Accelerated computing	P / G / F	General-purpose GPU (P), Graphical GPU (G), or FPGA (F) capabilities
HPC optimized	HPC	Designed for tightly-coupled, compute-intensive HPC workloads such as computati onal fluid dynamics (CFD), weather forecasting, and multiphysics simulations

This diverse selection of instance types helps support a wide variety of workloads with optimal hardware and promotes experimentation. HPC teams can benchmark various sets of instances to

optimize their scheduling strategies. Quantitative developers can try new approaches with GPUs, FPGAs, or the latest CPUs, without upfront costs or protracted procurement processes. You can immediately deploy at scale your optimal approach, without the traditional hardware lifecycle considerations.

When you run experiments, or if a subset of production workloads requires a specific instance type, grid schedulers typically enable tasks to be directed to the appropriate hardware through compute resource groups.

To simplify instance selection, AWS also offers attribute-based instance type selection. With this feature, teams can express compute requirements in terms of: vCPUs, memory, storage, and more. Using these attributes, Amazon EC2 Auto Scaling or Amazon EC2 Fleet will choose the instances that fit the specified attributes. This removes the need to manually pick instance types, ensures access a broader range of capacity via Amazon EC2 Spot and automatically enables new instance types as they become available as long as they meet the criteria that you define.

x86 instances

Each vCPU on x86-based Amazon EC2 instances is thread on the processor, except for C7a, R7a, M7a, T2, and m3.medium instances. Each thread is represented as a virtual CPU (vCPU) on the instance. An instance has a default number of CPU cores, which varies according to instance type.

To ensure that each vCPU is used effectively, it's important to understand the behavior of the calculations running in the HPC environment. If all processes are single-threaded, a good initial strategy is to have the scheduler assign one process per vCPU on each instance. However, if the calculations require multithreading, tuning might be required to maximize the use of vCPUs without introducing excessive CPU context switching.

By default, most x86 based Amazon EC2 instances have hyperthreading (HT) enabled. You can disable HT either at boot or at runtime if the workload performs better without it, which you can establish through benchmarking. The Disabling Intel Hyper-Threading Technology on Amazon Linux blog post has an explanation of the methods you can use to configure HT on an Amazon Linux instance. The Amazon EC2 documentation shows how to specify CPU options during instance launch and this includes setting the number of threads per physical core.

Another potential optimization is over-subscription. This approach is useful when you know processes spend time on non-CPU intensive activities, such as waiting on data transfers or loading binaries into memory. For example, if this overhead is estimated at 10%, you might be able to

schedule one additional task on the host for every 10 vCPUs to achieve higher CPU utilization and throughput.

Graviton instances

There are many performance benefits of AWS Graviton processors. AWS Graviton processors are custom built by AWS using 64-bit Arm Neoverse cores. Since the first A1 Graviton instance launched in 2019, AWS has launched three additional generations including the latest: Graviton4. Each generation has brought improvements in performance and efficiency, offering better price performance over comparable current generation x86-based instances for a wide variety of workloads, including application servers, microservices, high performance computing, electronic design automation, gaming, open-source databases, and in-memory caches. The latest Graviton instances support DDR5 memory providing 50% more memory bandwidth compared to DDR4.

One of the Amazon EC2 instances powered by AWS Graviton is the Hpc7g, which offers up to 20% higher performance compared to Hpc6a. The Hpc7g offers high-memory bandwidth and 200 Gbps of Elastic Fabric Adapter (EFA) network bandwidth, and can be used with AWS ParallelCluster. If higher network bandwidth is required for more loosely coupled applications, consider the C7gn instances which offer up to 200 Gbps of network bandwidth.

AWS Graviton has no simultaneous multithreading (SMT) or hyper threading. There is no sharing of execution units or interference between threads and sharing of L1 or L2 caches, as every vCPU is a physical core. Therefore, on AWS Graviton, well-optimized HPC applications that may have competed for resources on other instances, can push the CPUs to a higher threshold and limit extending execution time when gaps in execution stream cannot be filled.

Interpreted and bytecode-compiled languages such as Python, Java, Node.js and .NET Core on Linux may run on AWS Graviton without modification. Support for Arm architectures is also increasingly common in third-party numerical libraries, aiding the path to adoption. For certain bytecode-compiled language, extensions modules written in lower-level languages like C or C++ need to be compiled to arm64. Code changes will also be required for lower-level languages if they use to hardware specific features like single instruction, multiple data (SIMD) instructions.

Recommended flags to compile applications for AWS Graviton based instances can be found in the AWS Graviton Technical Guide on Github. Additionally, AWS offers the Porting Advisor for Graviton, a command line tool that analyzes source code and generates a report with any discovered incompatibilities with Graviton CPUs.

Software considerations

Compiler selection is another consideration. The use of a complier that is optimized for the target CPU architecture can yield performance improvements. For example, quantitative analysts might see value in developing analytics using the Intel Compiler when targeting x86-based Amazon EC2 instances offering AVX-512 SIMD instructions. The AVX-512 instruction set allows developers to run twice the number of floating-point operations per second (FLOPS) per clock-cycle. Similarly, AMD offers the AMD Optimizing C/C++ Compiler which optimizes for AMD EPYC architectures and 4th generation AMD EPYC processors like c7a also support AVX-512.

On AWS Graviton-based Amazon EC2 instances, AWS highly recommends using optimization flags specifically targeting AWS Graviton processors as documented in the Graviton Technical Guide and using recent versions of the compilers. Optimized mathematical functions provided by ARM can be found in libamath.

Graviton processors are compliant with the IEEE 754 standard, so they treat floating point numbers the same way as x86 processors. It is important to note that different compilers may generate numerical conversions in varying ways. This highlights the importance of ensuring floating point error mitigation is properly accounted for in code.

Instance procurement

In addition to the instance types shown in Table 1, there are also options for procuring instances in AWS:

- Amazon EC2 On-Demand Instances offer capacity as required, for as long as they are needed. You are only charged for the time that the instance is active. These are ideal for components that benefit from elasticity and predictable availability, such as brokers, compute instances hosting long-running tasks, or tasks that generate further generations of tasks.
- Amazon EC2 Spot Instances are particularly appropriate for HPC compute instances because they let you take advantage of unused Amazon EC2 capacity in the AWS Cloud. Spot Instances can occasionally be ended by AWS when capacity is constrained in a specific pool. Auto Scaling groups help to re-provision Spot capacity from other eligible pools where spare capacity is available. Grid schedulers can typically accommodate these occasional interruptions and reschedule tasks accordingly.
- Savings Plans are a flexible pricing model that also provides savings of up to 72% on your AWS compute usage regardless of instance family, size, operating system (OS), tenancy or AWS

Region. Savings Plans offer significant discounts in exchange for a commitment to use a specific amount of compute power (measured in $/hour) for a one- or three-year period. Savings Plans are ideal for long-running hosts such as HPC Controller nodes or for a *baseline* of compute capacity that will be kept online for a majority of the time.

AWS Capacity Blocks for ML allow you to reserve latest generation GPU instances for a future date. Capacity Blocks can be used to reserve instances with a duration from one day up to two weeks, enabling experimentation and prototyping as well as ML model training and fine-tuning on highly sought-after GPU instances. Instances will be available for the reserved time and until 30 minutes before the end time of the Capacity Block. The last 30 minutes of the reservation are not charged in the price of the Capacity Block.

It's important to note that regardless of the procurement model selected the instances delivered by AWS are exactly the same.

Compute instance provisioning and management strategies

Spot Instances are unsuitable for workloads that are not instance flexible; for example, systems where deterministic performance is critical. Spot is not suitable either for stateful, fault-intolerant, or workloads that require tightly coupled communication between instance nodes. They are also not recommended for workloads that are intolerant of occasional periods when the target capacity is not completely available. However, many financial services organizations make use of Spot Instances for part of their HPC workloads. AWS provides services and features to maximize the benefits and availability of Spot instances. For example, when provisioning Spot instances using EC2 Fleet the following allocation strategies can be employed:

- **Price-capacity-optimized (recommended)** - This strategy identifies the deepest pools of capacity to reduce interruption rates and then requests instances from the lowest priced instances from those pools.
- **Capacity-optimized** - Automatically launches Spot Instances into the most available pools by looking at real-time capacity data and predicting which are the most available.
- **Lowest-price** - Instances come from the lowest priced pool that has available capacity.
- **Diversified** - Spot Instances are distributed across all Spot capacity pools.

AWS provides two mechanisms to advise on interruptions. The first, a Spot Instance interruption notice, is a warning that is issued two minutes before Amazon EC2 interrupts a Spot Instance.

You can configure your Spot Instances to be stopped or hibernated, instead of being ended when they are interrupted. Amazon EC2 will then automatically resume them when capacity becomes available in the same Availability Zone and instance type. The second mechanims is the EC2 Instance rebalance recommendation, a signal that notifies you when a Spot Instance is at elevated risk of interruption. The signal gives you the opportunity to proactively manage the Spot Instance in advance of a possible two-minute Spot Instance interruption notice. You can decide to rebalance your workload to new or existing Spot Instances that are not at an elevated risk of interruption. AWS has made it easy for you to use this new signal by using the Capacity Rebalancing feature in EC2 Auto Scaling groups and Spot Fleet.

If hibernation is configured, this feature operates like closing and opening the lid on a laptop computer, and saves the memory state to an Amazon Elastic Block Store (Amazon EBS) disk. However, this approach to managing interruptions should be used with caution because the grid scheduler might not be able to track such quiesced workloads, which could result in timeouts and rescheduling tasks if the hibernated image is not reactivated quickly.

Amazon EC2 Fleet enables you to quickly create fleets that are diversified by using EC2 On-Demand Instances, Reserved Instances, and Spot Instances. With this approach, you can optimize your HPC capacity management plan according to the changing demands of your workloads. Note that EC2 Spot Fleet is a legacy API with no planned investment.

EC2 Fleet integrates with Amazon EventBridge to notify you about important Fleet events, state changes, and errors. This enables you to automate actions in response to Fleet state changes, and monitor the state of your Fleet from a central place without needing to continuously poll Fleet APIs.

Amazon EC2 Auto Scaling groups contain a collection of Amazon EC2 instances that are treated as a logical grouping for the purposes of automatic scaling and management. An Auto Scaling group enables you to use Amazon EC2 Auto Scaling features, such as health check replacements and scaling policies. Additionally, you can select instance purchase options such as On-Demand and Spot, use attribute-based instance selection, and price capacity automation.

A note on EC2 Fleet vs. Autoscaling groups. EC2 Fleet is suitable for workloads that require a fixed set of instances to run. Autoscaling can achieve many of the same goals through attribute-based instance selection: Selecting a range of CPUs, and memory for instance as well as the pricing strategy you'd prefer (Spot, On-Demand or Savings Plans). In most HPC cases we suggest you use Autoscaling.

Amazon EC2 launch templates contain the configuration information used to launch an instance. The template can define the Amazon Machine Image (AMI) ID (Operating system image), instance type, and network settings for the compute instances. You can use Launch Templates with EC2 Fleet or Amazon EC2 Auto Scaling and they make it easier to implement and track configuration standards.

Launch Template versioning can be used within the EC2 Auto Scaling Group 'Instance Refresh' feature to update pools of capacity while minimizing interruptions to the workload. All you need to do is specify the percentage of healthy instances to keep in the group while the Auto Scaling group terminates and launches instances. You can also specify the warm-up time, which is the time period that the Auto Scaling group waits between instances that get refreshed via Instance Refresh.

Capacity management

As the financial services industry increasingly relies on complex computational models, the need for scalable and cost-effective capacity management solutions becomes paramount. This section explores various strategies and best practices for capacity management in AWS, empowering financial organizations to optimize their compute resources and meet the demands of high throughput computing workloads. By leveraging AWS services and tools, financial institutions can enhance performance, reduce costs, and ensure smooth operations for their critical financial applications.

One option to begin an HPC deployment is to use only On-Demand Instances. After you understand the performance of your workloads, you can develop and optimize a strategy to provision instances using Savings Plans, Spot, Auto Scaling Groups, or Amazon EC2 Fleet.

For example, you can deploy a number of instances covered by Savings Plans to host core grid services, such as schedulers, that are required to be available at all times. You can then provision On-Demand Instances during the intraday period to ensure predictable performance for synchronous pricing calculations. Highly optimized grids end up using a mix of all the purchasing options. For End of Day (EOD) batches, it's common for 60% to 80% of capacity to be provisioned using Spot Instances.

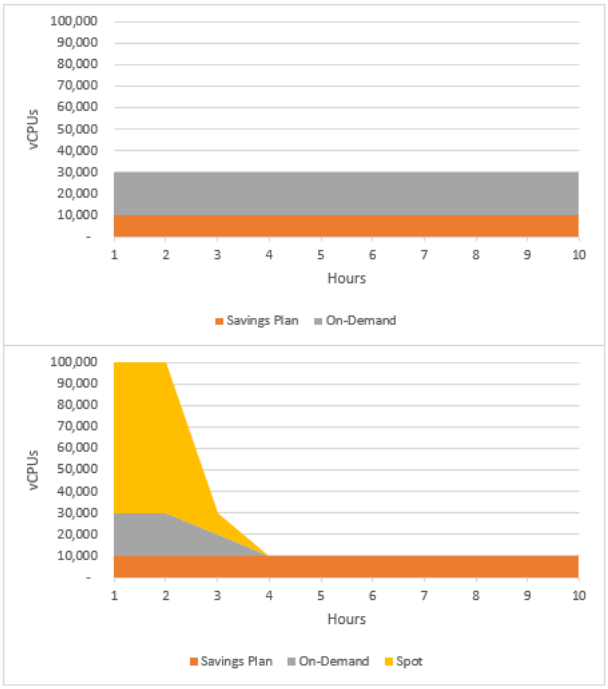
Understanding the importance of instance diversification and availability is key to successfully provisioning Spot capacity at scale. The Spot Placement Score (SPS) can be used to get a score from 1-9 indicating how likely a Spot capacity request will succeed in a region or Availability Zone. Spot is spare capacity and as such it fluctuates over time. Projects like the Spot Placement Score Tracker can be used to track the best regions to run large Spot workloads at scale. This can be

useful in the definition of multi-region strategies, where, when regulatory framework allows, the workload can be run on the regions that optimize for capacity availability and price performance.

The following figure shows two approaches to provisioning. In each case, 10,000 vCPUs of Savings Plan-based capacity remain online for the stateful scheduling components.

In the first case, 20,000 further vCPUs are provisioned using On-Demand Instances for ten hours to accommodate a batch that runs for 200,000 vCPU hours with a ten-hour SLA.

In the second approach, the 20,000 vCPUs are provisioned at the outset using On-Demand Instances to provide confidence in the batch delivery, but 70,000 vCPUs based on low-cost Spot Instances are also added. Because of the volume of Spot Instances, the batch completes much more quickly (in about three hours) and at a significantly reduced cost. However, if the Spot Instances were not available for any reason, the batch would still complete on time with the On-Demand Instances provisioned.



Two contrasting AWS instance provisioning strategies

Attribute-based instance type selection

Attribute-based instance type selection circumvents the need to directly specify instance types for your workloads and instead allows you to express requirements such as vCPUs, memory, storage, and other attributes and leave AWS to select the instances. There are advantages to this approach: it reduces the need for you to research instance types, it ensures that as new instance

types become available that you can take advantage of the immediately (as long as they meet your requirements), and it helps with diversification of instance types, opening up more pools from which to provision Spot instances.

Amazon EC2 Fleet

EC2 Fleet enables you to quickly create fleets that are diversified by using EC2 On-Demand Instances, Savings Plans, and Spot Instances. With this approach, you can optimize your HPC capacity management plan according to the changing demands of your workloads.

For capacity management use-cases requiring custom scale in and scale out strategies we recommend using EC2 Fleet Instant request type. This is a synchronous one-time request that makes only one attempt to launch your desired capacity. This will allow you to bulk request instances without the autoscaling functionality, allowing you to develop scaling strategies for your specific use-case.

Amazon EC2 Auto Scaling

One of the key benefits of deploying applications in the AWS Cloud is *elasticity*. Amazon EC2 Auto Scaling enables HPC managers to configure Amazon EC2 instance provisioning and decommissioning events based on the real-time demands of their platform. The concept of 'Instance Weightings' allows Auto Scaling groups to start instances from a diverse pool of instance types to meet an overall capacity target for the workload. Though grids were previously provisioned based on predictions of peak demands (with periods of both constraint and idle capacity), Amazon EC2 Auto Scaling has a rich API that enables it to be integrated with schedulers to easily manage scaling events.

When you remove hosts from a running cluster, make sure to allow for a *drain down* period. During this period, the targeted host stops taking on new work, but is allowed to complete work in progress. When you select nodes for removal, avoid any long-running tasks, so that the shutdown is not delayed and you don't lose progress on those calculations. If the scheduler allows a query of total runtime of tasks in progress, grouped by instance, you can use this to identify which are the optimal candidates for removal, specifically the instances with the lowest aggregate total of runtime by tasks in progress.

Where capacity is managed automatically, Amazon EC2 Auto Scaling groups offer 'scale-in' protection as well as configurable termination policies to allow HPC managers to minimize disruption to tasks in flight. Scale-in protection allows an Auto Scaling Group, or an individual instance to be marked as 'InService' and so ineligible for termination in a 'scale-in' event. You also

have the option to build custom ending policies using AWS Lambda to give more control over which instances are ended. These protections can be controlled by an API for integration with the scheduler to automate the drain down process.

Paradoxically, adding instances to a cluster can temporarily slow the flow of tasks if those new instances need some time to reach optimal performance, as binaries are loaded into memory and local caches are populated. Amazon EC2 Auto Scaling groups also support *warm pools*. A warm pool is a pool of pre-initialized EC2 instances that sits alongside the Auto Scaling group. Whenever your application needs to scale out, the Auto Scaling group can draw on the warm pool to meet its new desired capacity. The goal of a warm pool is to ensure that instances are ready to quickly start serving application traffic, accelerating the response to a scale-out event. This is known as a *warm start*.

So far, this section has addressed compute instance provisioning at the host level. Increasingly customers are looking to serverless solutions based on either container technologies such as Amazon Elastic Container Service (Amazon ECS), Amazon Elastic Kubernetes Service (Amazon EKS), or AWS Lambda.

For both Amazon ECS and Amazon EKS, the AWS Fargate serverless compute engine removes the need to orchestrate infrastructure capacity to support containers. Fargate allocates the right amount of compute, eliminating the need to choose instances and scale cluster capacity. You pay only for the resources required to run your containers, so there is no over-provisioning and paying for additional servers.

Fargate supports both Spot Pricing for ECS and Compute Savings Plans for Amazon ECS and Amazon EKS.

For customers using AWS Lambda, there are no instances to be scaled; however there is the concept of *Concurrency* which is the number of instances of a function which can serve requests at a time. There are default Regional concurrency limits which can be increased through a request in the Support Center console. Financial services firms have already built completely serverless HPC solutions based on Lambda (similar to the architecture outlined here) that support tens of millions of calculations per day.

In addition to considering alternative CPU architectures and accelerated computing options, customers are increasingly looking at their existing dependencies on commercial operating systems such as Microsoft Windows. Such dependencies are often historical, stemming from risk management systems built around spreadsheets, however today the cost premiums can be very material especially when compared to deeply discounted EC2 capacity under Amazon EC2 Spot.

AWS offers a variety of Linux distributions including Red Hat, SUSE, CentOS, Debian, Kali, Ubuntu, and Amazon Linux. The latter is a supported and maintained Linux image provided by AWS for use on Amazon EC2 (it can also be run on-premises for development and testing). It is designed to provide a stable, secure, and high-performance run environment for applications running on Amazon EC2. It supports the latest EC2 instance type features, and includes packages that enable easy integration with AWS. AWS provides ongoing security and maintenance updates to all instances running the Amazon Linux AMI, and it is provided at no additional charge to Amazon EC2 users.

Storage and data sharing

In HPC systems, there are two primary data distribution challenges. The first is the distribution of binaries. In financial services, large and complex analytical packages are common. These packages are often 1GB or more in size, and often multiple versions are in use at the same time on the same HPC platform, to support different businesses or back-testing of new models.

In a constrained, on-premises environment, you can mitigate this challenge through relatively infrequent updates to the package and a fixed set of instances. However, in a cloud-based environment, instances are short-lived and the number of instances can be much larger. As a result, multiple packages may be distributed to thousands of instances on an hourly basis as new instances are provisioned and new packages are deployed.

There are a number of possible approaches to this problem. One is to maintain a build pipeline that incorporates binary packages into the Amazon Machine Images (AMIs). This means that once the machine has started, it can process a workload immediately because the packages are already in place. The EC2 Image Builder tool simplifies the process of building, testing and deploying AMIs. A limitation of this approach is that it doesn't accommodate the deployment of new packages to running instances, and it requires them to be ended and replaced to get new versions.

Another approach is to update running instances. There are two different methods for this type of update, which are sometimes combined:

- ***Pull* (or *lazy*) deployment** — In this mode, when a task reaches an instance and it depends on a package that is not in place, the engine *pulls* it from a central store before it runs the task. This approach minimizes the distribution of packages and saves on local storage because only the minimum set of packages is deployed. However, these benefits are at the expense of delaying tasks in an unpredictable way, such as the introduction of a new instance in the middle of a latency sensitive pricing job. This approach may not be acceptable if large volumes of tasks have

to wait for the grid nodes to pull packages from a central store which could struggle to service very large numbers of requests for data.

- ***Push* deployment** — In this mode you can instruct instance engines to proactively get a specific package before they receive a task that depends on it. This approach allows for rolling upgrades and ensures tasks are not delayed by a package update. One challenge with this method is the possibility that new instances (which can be added at any time) might miss a *push* message, which means you must keep a list of all currently *live* packages.

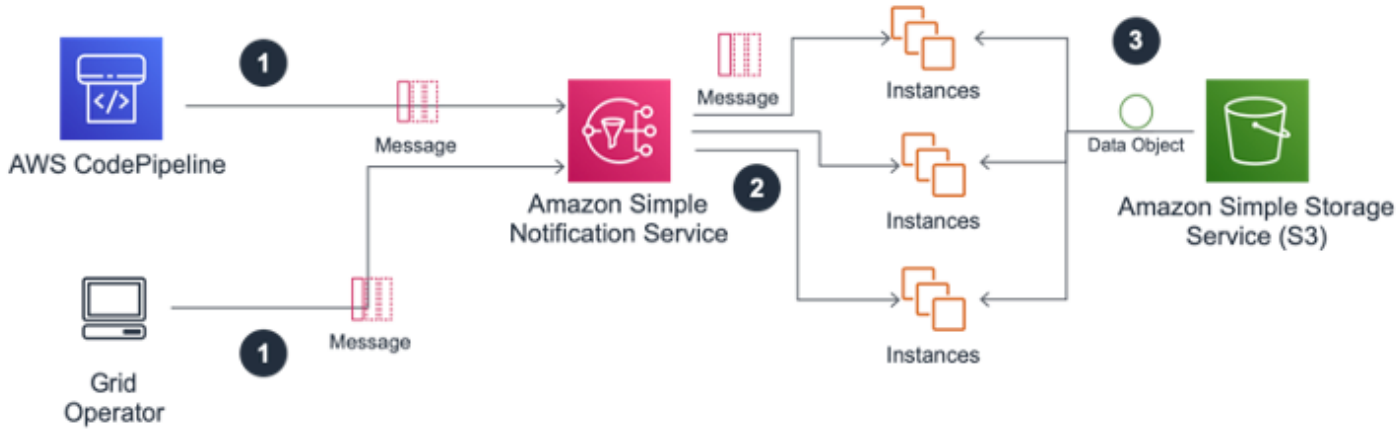
In practice, a combination of these approaches is common. Standard analytics packages are *pushed* because they're likely to be needed by the majority of tasks. Experimental packages or incremental 'Delta' releases are then *pulled*, perhaps to a smaller set of instances.

It might also be necessary to purge deprecated packages, especially if you deploy experimental packages. In this case, you can use a list of *live* packages to enable your compute instances to purge any packages that are not in the list and thus are not current.

The following figure shows a cloud-native implementation of these approaches. It uses a centralized package store in Amazon Simple Storage Service (Amazon S3) with agents that respond to messages delivered through an Amazon Simple Notification Service (Amazon SNS) topic.

After the package is in place on Amazon S3, notifications of new releases can be generated either by an operator or as a final step in an automated build pipeline. Compute instances subscribed to an SNS topic (or to multiple topics for different applications) use these messages as a trigger to retrieve packages from Amazon S3. You can also use the same mechanism to distribute *delete* messages to remove packages, if required.

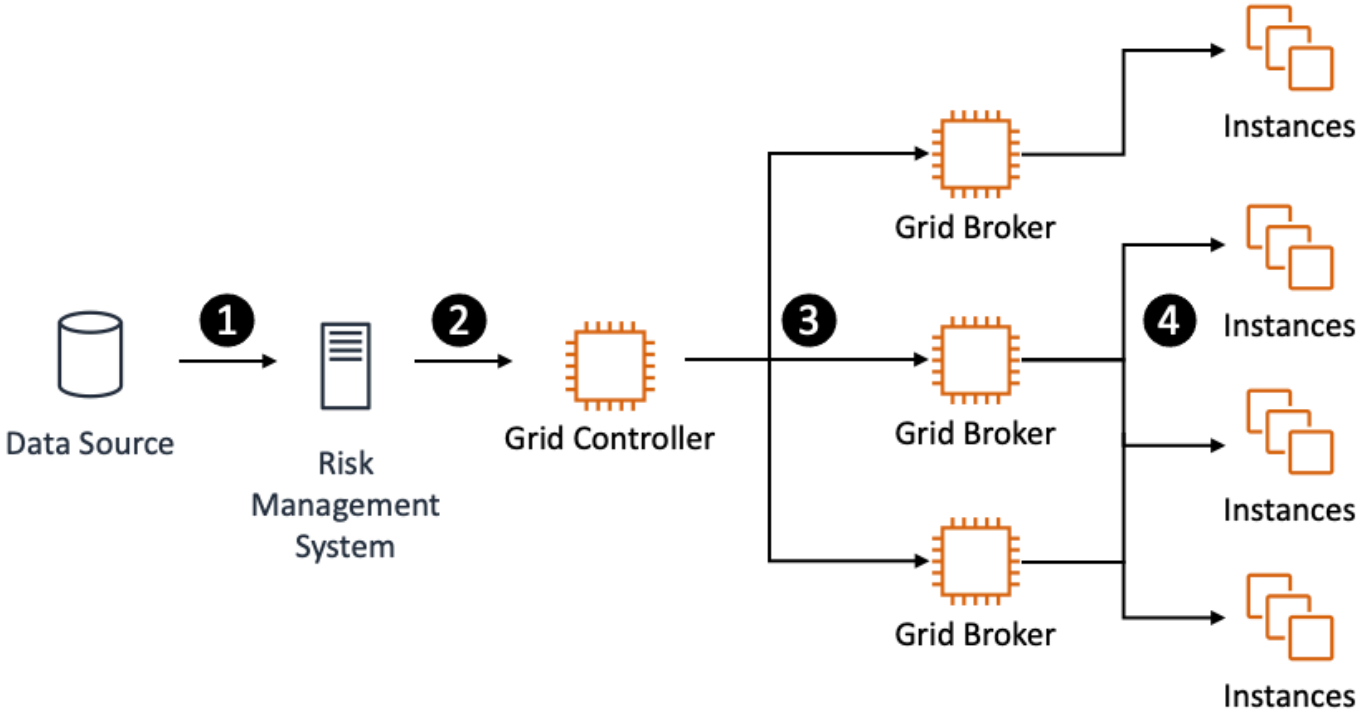
Note that in some cases it might be possible to pull binaries on-demand from Amazon S3 by using MountPoint for Amazon S3, an open-source client that you can mount an S3 bucket on a compute instance. Your application can then access the files as needed as a local file system. MountPoint automatically translates local file system calls to REST API calls on S3 objects.



Data distribution architecture using Amazon SNS messages and S3 Object Storage

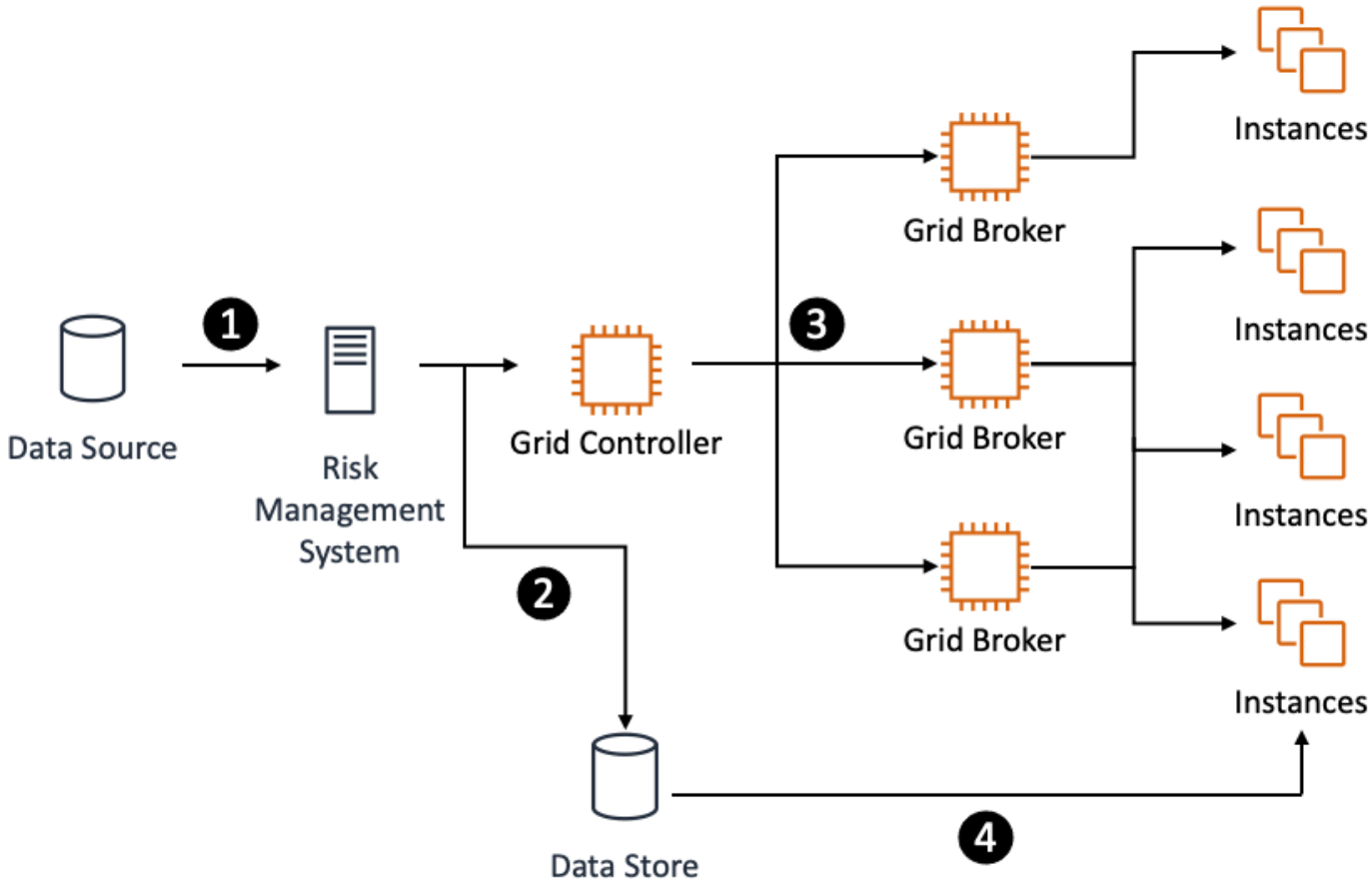
The second data distribution challenge in HPC is managing data related to the tasks being processed. Typically, this is bi-directional, with data flowing to the engines that support the processing and resulting data passed back to the clients. There are three common approaches for this process:

- In the first approach, communications are *inbound* (see the following figure) with all data passing through the grid scheduler along with task data. This is less common because it can cause a performance bottleneck as the cluster grows.



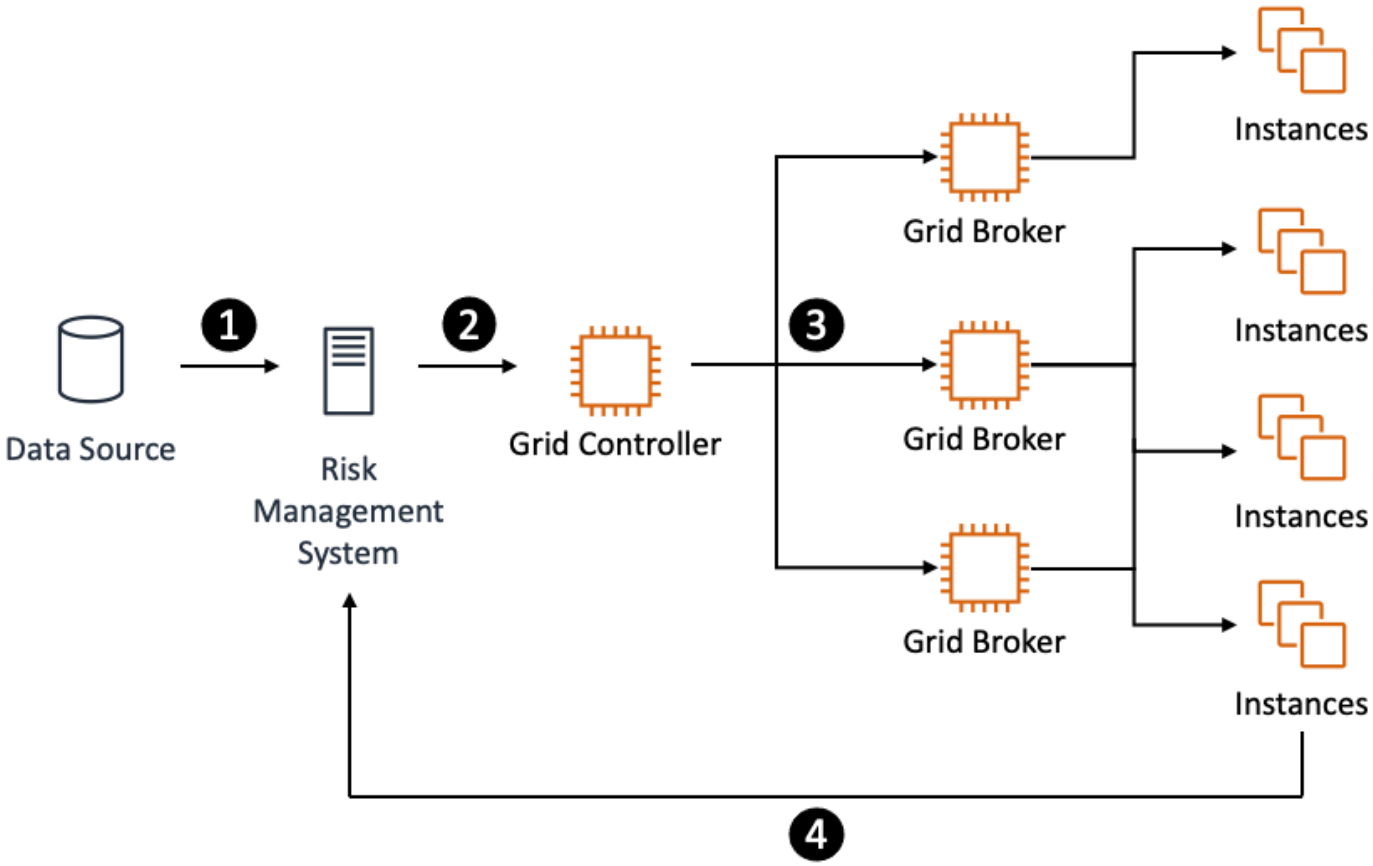
An inbound data distribution approach

- In another approach, tasks pass through the scheduler, but the data is handled *out-of-bounds* through a shared, scalable data store or an in-memory data grid (see the following figure). The task data contains a reference to the data's location and the compute instances can retrieve it as required.



An out-of-bounds data distribution approach

- Finally, some schedulers support a direct data transfer (DDT) approach. In this model the scheduler grid broker allocates compute instances which then communicate directly with the client. This architecture can work well, especially with very short running tasks with little data. However, in a hybrid model, with thousands of engines running on AWS that need to access a single, on-premises client, this can present challenges to on-premises firewall rules, or to the availability of ephemeral ports on the client host.



DDT (direct data transfer) data distribution approach

All of these approaches can be enhanced with caches located as close as possible to, or hosted on, the compute instances. Such caches help to minimize the distribution of data, especially if a significantly similar set is required for many calculations. Some schedulers support a form of data-aware scheduling that tries to ensure that tasks that require a specific dataset are scheduled to instances that already have that dataset. This cannot be guaranteed, but often provides a significant performance improvement at the cost of local memory or storage on each compute instance.

Though the combination of grid schedulers and distributed cache technologies used on premises can provide solutions to these challenges, their capabilities vary and they are not typically engineered for a cloud deployment with highly elastic, ephemeral instances. You can consider the following AWS services as potential solutions to the typical HPC data management use cases.

Amazon Simple Storage Service (Amazon S3)

The Amazon S3 provides virtually unlimited object storage designed for 99.999999999% of durability and high availability. For binary packages, it offers both versioning and various immutability features, such as S3 Object Lock, which prevents deletion or replacement of objects and has been assessed by Cohasset Associates for use in environments that are subject to SEC 17a-4, CFTC, and FINRA regulations.

Binary immutability is a common audit requirement in regulated industries, which require you to demonstrate that the binaries approved in the testing phase are identical to those used to produce reports. You can include this feature in your deployment pipeline to make sure that the analytics binaries you use in production are the same as those that you validated. This service also offers easy to implement encryption and granular access controls.

Some HPC architectures use checkpointing (compute instances save a snapshot of their current state to a datastore) to minimize the computational effort that could be lost if a node fails or is interrupted during processing. For this purpose, a distributed object store (such as Amazon S3) might be an ideal solution. Because the data is likely to only be needed for the life of the batch, you can use S3 life cycling rules to automatically purge these objects after a small number of days to reduce costs.

For HPC systems that require access to a shared POSIX filesystem, Mountpoint for Amazon S3 offers access to objects stored in S3 through operations including `open` and `read`. The combination of Mountpoint and Amazon S3 allows many clients to read objects at once.

Amazon Elastic File System (Amazon EFS)

Amazon EFS offers shared network storage that is elastic, which means it grows and shrinks as required. Thousands of Amazon EC2 instances can mount EFS volumes at the same time, which enables shared access to common data such as analytics packages. Amazon EFS does not currently support Windows clients.

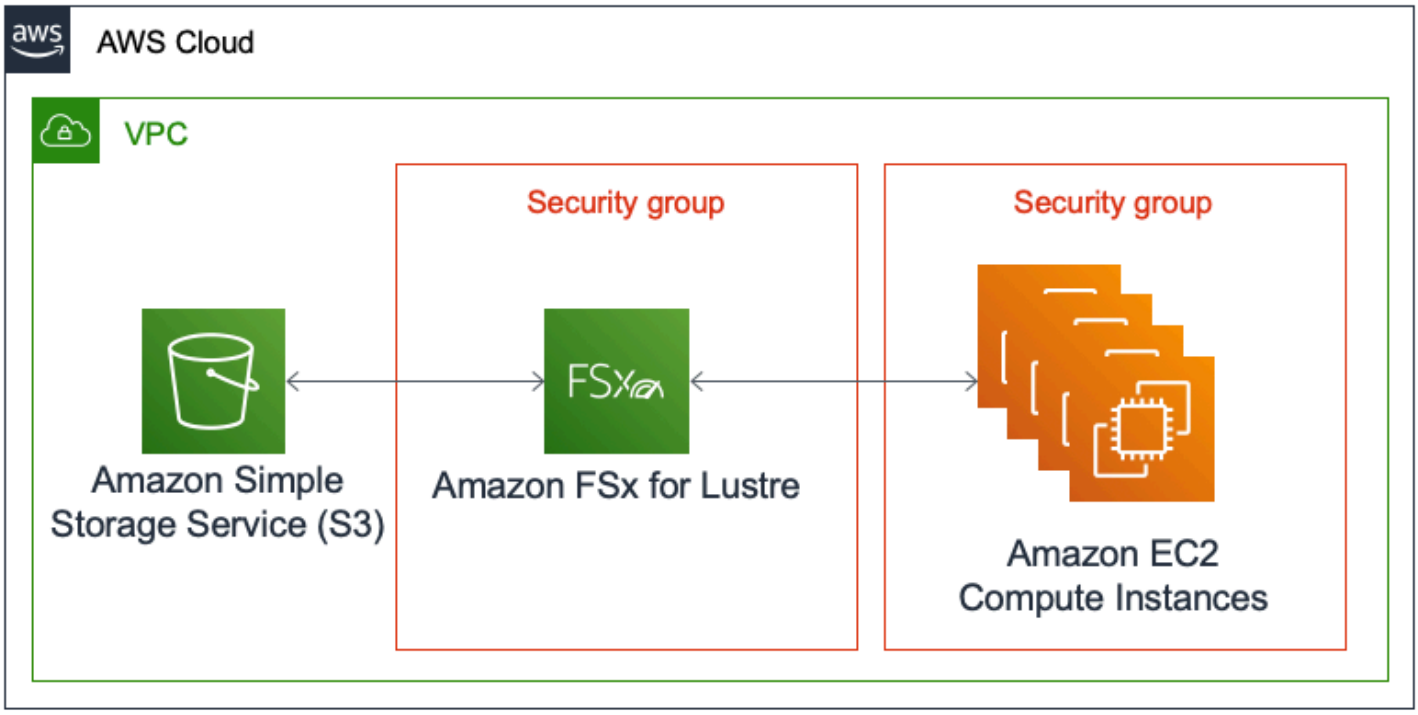
Amazon FSx for Windows File Server

Amazon FSx for Windows File Server provides fully managed, highly reliable, and scalable file storage that is accessible over the open standard Server Message Block (SMB) protocol. It is built on Windows Server, delivering a wide range of administrative features such as user quotas, end user file restores, and Microsoft Active Directory integration. It offers single- and Multi-Availability Zone deployment options, fully managed backups, and encryption of data at rest and in transit.

Amazon FSx for Lustre

For transient job data, the Amazon FSx for Lustre service provides a high-performance file system that offers sub-millisecond access to data and read/write speeds of up to hundreds of gigabytes per second with millions of IOPs. Amazon FSx for Lustre can link to an S3 bucket, which makes it easy for clients to write data objects to the bucket (including clients from an on-premises system) and have those objects available to thousands of compute nodes in the cloud (see the following figure).

FSx for Lustre is ideal for HPC workloads because it provides a file system that's optimized for the performance and costs of high-performance workloads, with file system access across thousands of EC2 instances.



An example of an Amazon FSx for Lustre implementation

Amazon FSx for NetApp ONTAP

Amazon FSx for NetApp ONTAP is a storage service that allows you to launch and run fully managed NetApp ONTAP file systems in the AWS Cloud. To support HPC use-cases it provides multiple GB/s of throughput and hundreds of thousands of IOPS per filesystem with consistent sub-millisecond latencies on SSD-based storage.

Amazon Elastic Block Store (Amazon EBS)

After a compute instance has binary or job data, it might not be possible to keep it in memory, so you might want to keep a copy on a local disk. Amazon EBS offers persistent block storage volumes for Amazon EC2 instances.

Though the volumes for compute nodes can be relatively small (10GB can be sufficient to store a variety of binary package versions and some job data) there might be some benefit to the higher IOPS and throughput offered by the Amazon EBS-provisioned input/output operations per second (IOPS) solid state drives (SSDs). These offer up to 64,000 IOPS per volume and up to 1,000MB/s of throughout, which can be valuable for workloads that require frequent, high-performance access to these datasets.

Because these volumes incur additional cost, you should complete an analysis of whether they provide any additional value over the standard, general-purpose volumes.

AWS Cloud hosted data providers

AWS Data Exchange makes it easy to find, subscribe to, and use third-party data in the cloud. The catalog includes hundreds of financial services datasets from a wide variety if providers. Once subscribed to a data product, you can use the AWS Data Exchange API to load data directly into S3.

The Bloomberg Market Data Feed (B-PIPE) is a managed service providing programmatic access to Bloomberg's complete catalog of content (all the same asset classes as the Bloomberg Terminal). Network connectivity with Bloomberg B-PIPE leverages AWS PrivateLink, exposing the services as set of local IP addresses within your Amazon Virtual Private Cloud (Amazon VPC) subnet and eliminating DNS issues. B-PIPE services are presented via Network Load Balancers to further simplify the architecture.

Additionally, Refinitiv's Elektron Data Platform provides cost-efficient access to global, real-time exchange, 'over the counter' (OTC), and contributed data. The data is also provided using AWS PrivateLink, allowing simple and secure connectivity from your Virtual Private Cloud (VPC).

Data management and transfer

Although HPC systems in financial services are typically loosely coupled, with limited need for *East-West* communication between compute instances, there are still significant demands for *North-South* communication bandwidth between layers in the stack. A key consideration for networking is where in the stack any separation between on-premises systems and cloud-based systems occurs.

This is because communication within the AWS network is typically of higher bandwidth and lower cost than communication to external networks. As a result, any architecture that causes hundreds or thousands of compute instances to connect to an external network—particularly if they're requesting the same binaries or task data—would create a bottleneck.

Ideally, the *fanout* point (the point in the architecture at which large numbers of instances are introduced) is in the cloud. This means that the larger volumes of communication stay in the AWS network with relatively few connections to on-premises systems.

AWS offers networking services that complement the financial services HPC systems. A common starting point is to deploy AWS Direct Connect connections between customer data centers and an AWS Region through a third-party point of presence (PoP) provider. A Direct Connect link offers a consistent and predictable experience with speeds of up to 100Gbps. You can employ multiple diverse Direct Connect links to provide highly resilient, high-bandwidth connectivity.

Though most HPC applications within financial services are loosely coupled, this isn't universal and there are times when network bandwidth is a significant component of overall performance. The current AWS Nitro–based instances offer various levels of network bandwidth with the largest compute instance types such as the c6in.32xlarge or c7gn.16xlarge instances, which offer up to 200Gbps (in the case of c6i.32xlarge, two network interfaces must be attached to get a maximum of 200Gbps) and GPU enabled p5 instances offering up to 3,200 Gbps. Additionally, a cluster placement group packs instances close together inside an Availability Zone. This strategy enables workloads to achieve the low-latency network performance necessary for tightly-coupled node-to-node communication that is typical of some HPC applications.

The Elastic Fabric Adaptor service (EFA) enhances the Elastic Network Adaptor (ENA), and is specifically engineered to support tightly-coupled HPC workloads which require low latency communication between instances. An EFA is a virtual network device which can be attached to an Amazon EC2 instance. EFA is suited to workloads using the Message Passing Interface (MPI). EFA may be worthy of consideration for some financial services workloads, such as weather predictions, as part of an insurance industry catastrophic event model.

EFA traffic that bypasses the operating system (OS-bypass) is not routable, so it's limited to a single subnet. As a result, any peers in this network must be in the same subnet and Availability Zone, which could alter resiliency strategies. The OS-bypass capabilities of EFA are also not supported on Windows.

Some Amazon EC2 instance types support *jumbo frames* where the Network Maximum Transmission Unit (the number of bytes per packet) is increased. AWS supports MTUs of up to 9001

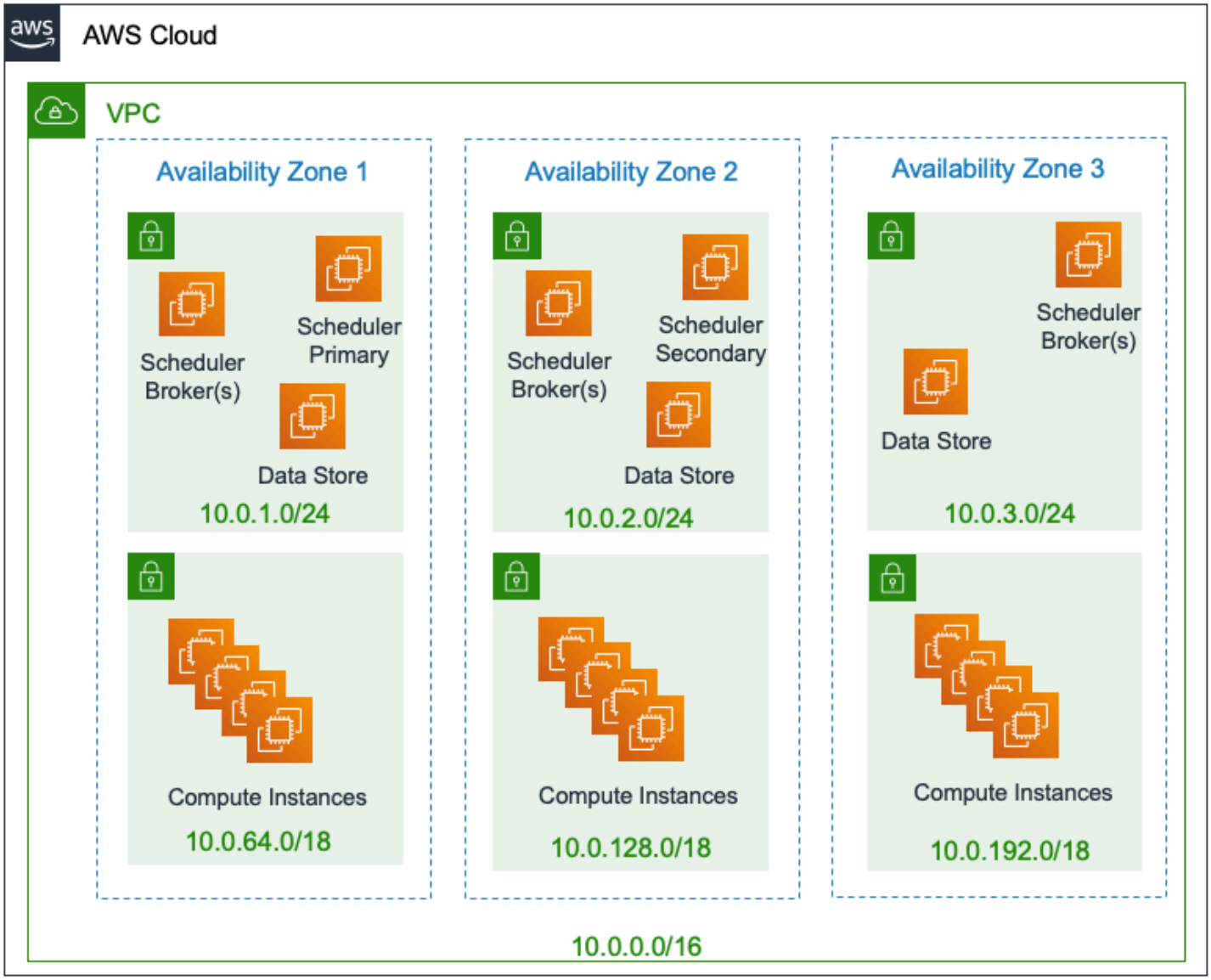
bytes. By using fewer packets to send the same amount of data, end-to-end network performance is improved.

Operations and management

HPC systems are traditionally highly decoupled and resilient to the failure of any given component, with minimal disruption. However, HPC systems in financial services organizations tend to be both mission critical and limited by the capabilities of traditional approaches, such as physical primary and secondary data centers. In this model, HPC teams have to choose between having secondary infrastructure sitting mostly idle in case of the loss of a data center, or using all of the infrastructure on a daily basis but with the possibility of losing up to 50% of that capacity in a disaster event. Some add a third or fourth location to reduce the impact of the loss of an individual site, but at the cost of an increased likelihood of an outage and network inefficiencies.

When you move to the cloud, you not only open up the availability of new services, but also new approaches to solving these problems. AWS operates a model with Regions and Availability Zones that are always active and offer high levels of availability.

By architecting HPC systems for multiple AWS Availability Zones, financial services you can benefit from high levels of resiliency and utilization. In the unlikely event of the loss of an Availability Zone, additional instances can be automatically provisioned in the remaining Availability Zones to enable workloads to continue without any loss of data and only a brief interruption in service.

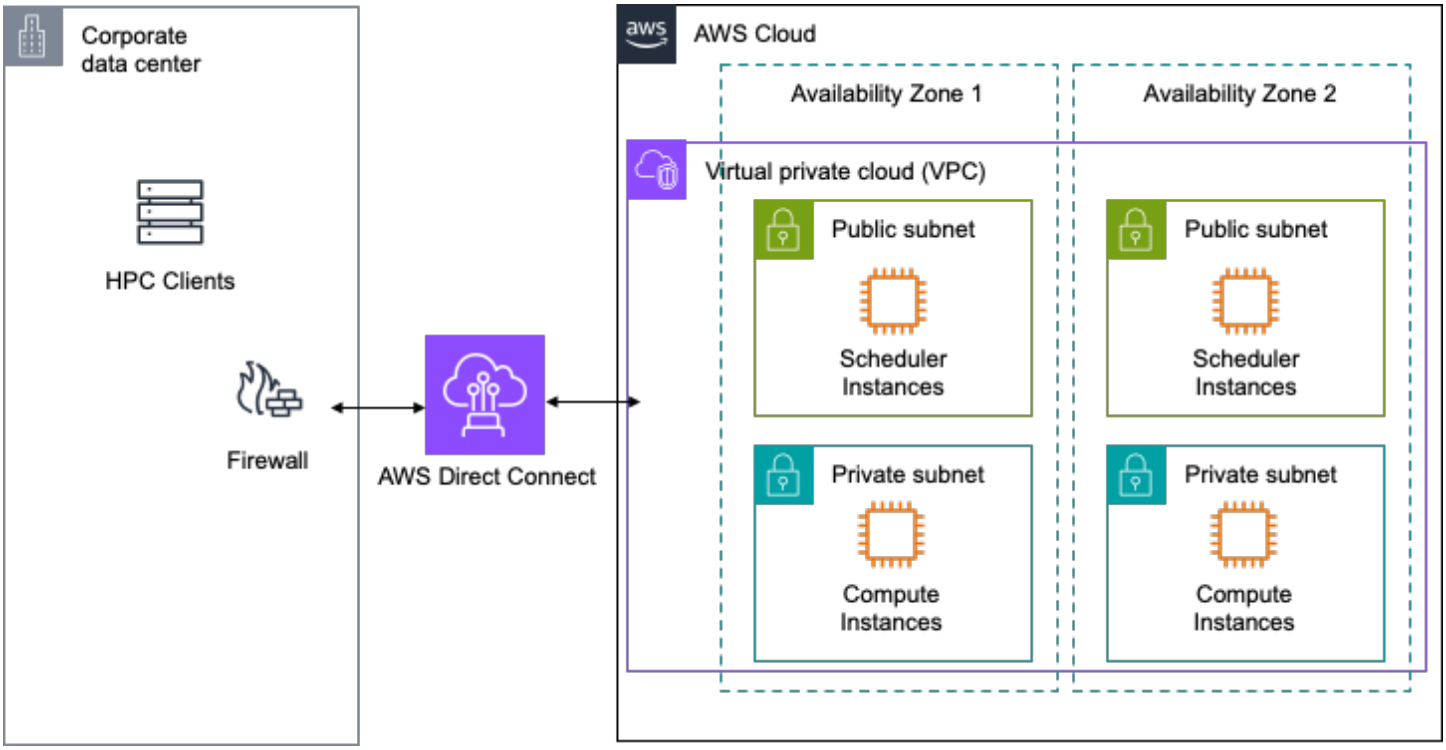


A sample HPC architecture for a Multi-AZ deployment

The high-level architecture in the preceding figure shows the use of multiple Availability Zones and separate subnets for the stateful scheduler infrastructure (including schedulers, brokers, data stores) and the compute instances. You can base your scheduler instances on long-running instances (procured with a Savings Plan) with static IP addresses to help them communicate with on-premises infrastructure by simplifying firewall rules.

Conversely, you can provision your compute instances using On-Demand or Spot with dynamically allocated IP addresses. Security groups act as a virtual firewall, which you can configure to allow the compute instances to communicate only with scheduler instances.

With the Compute Instances being inherently ephemeral and with potentially limited connectivity needs, it can be beneficial to have them sit within separate private address ranges to avoid the need for you to manage demand for and allocate IPs from your own pools. This can be a significant benefit to organizations that might not want to allocate large IPv4 address ranges to an HPC platform that can scale to many thousands of instances. This can be achieved either through a secondary CIDR on the VPC, or with a separate VPC for the compute infrastructure, connected through VPC peering.



An example network architecture using AWS Direct Connect for private connectivity and a combination of Public/Routable and Private/Non-routable subnets

The majority of AWS services relevant to financial services customers are accessible from within the VPC using AWS PrivateLink, which offers private connectivity to those services, and services hosted by other AWS accounts and supported AWS Marketplace partner solutions. Traffic between your VPC and the service does not leave the Amazon network and is not exposed to the public internet.

One of the keys to effective HPC operations are the metrics you collect and the tools to explore and manipulate them. A common question from end users is, “Why is my job taking so long?” It’s important to set up your HPC operation in a way that enables you to either answer that question, or to empower users to find it for themselves.

AWS offers tools you can use to collect metrics and logs, at scale. Amazon CloudWatch is a monitoring and management service that not only collects metrics and logs related to AWS services, but through an agent, it can also be a target for telemetry from HPC systems and the applications running on them. This provides a valuable central store for your data, and allows diverse data sources to be presented on a common time series, and helps you to correlate events when you diagnose issues. You can also use CloudWatch as an auditable record of the calculations that were completed, with the analytics binary versions that were used. You can export these logs to S3 and protect them with the object lock feature for long term, immutable retention.

You may want to use a third-party log analytics tool. Many of the most common products have native integrations with Amazon Web Services. Additionally, Amazon Managed Service for Grafana enables you to analyze, monitor, and alarm on metrics, logs, and traces across multiple data sources, including AWS, third-party independent software vendors (ISVs), databases, and other resources.

Some grid schedulers require a relational database for the retention of statistics data. For this purpose, you can use Amazon Relational Database Service (Amazon RDS), which provides cost-efficient and resizable database capacity, while automating administration tasks such as hardware provisioning, patching, and backups.

Another common challenge with shared tenancy HPC systems is the apportioning of cost. The ability to provide very granular cost metrics according to usage can drive effective business decisions within financial services.

The *pay as you go* pricing model of AWS empowers HPC managers and their end customers to realize the benefits from the optimization of the system or its use. AWS tools such as resource tagging and the AWS Cost Explorer can be combined to provide rich cost data and to build reports that highlight the sources of cost within the system. Tags can include details of report types, cost centers, or other information pertinent to the client organization. There's also an AWS Budgets tool that can be used to create reports and alerts according to consumption.

When you combine detailed infrastructure costs with usage statistics, you can create granular cost attribution reports. Some trades are particularly demanding of HPC capacity, to the extent that the business might decide to exit the trade instead of continuing to support the cost.

Task scheduling and infrastructure orchestration

A high-performance computing system needs to achieve two goals:

- ***Scheduling*** — Encompasses the lifecycle of compute tasks including: capturing and prioritizing tasks, allocating them to the appropriate compute resources, and handling failures.
- ***Orchestration*** — Making compute capacity available to satisfy those demands.

It's common for financial services organizations to use a third-party grid scheduler to coordinate HPC workloads, but orchestration is often a slow-moving exercise in procurement and physical infrastructure provisioning. Traditional schedulers are therefore highly optimized for making low-latency scheduling decisions to maximize usage of a relatively fixed set of resources.

As customers migrate to the cloud, the dynamics of the problem changes. Instead of near-static resource orchestration, capacity can be scaled to meet the demands at that instant. As a result, the scheduler doesn't need to reason about which task to schedule next but rather just inform the orchestrator that additional capacity is needed.

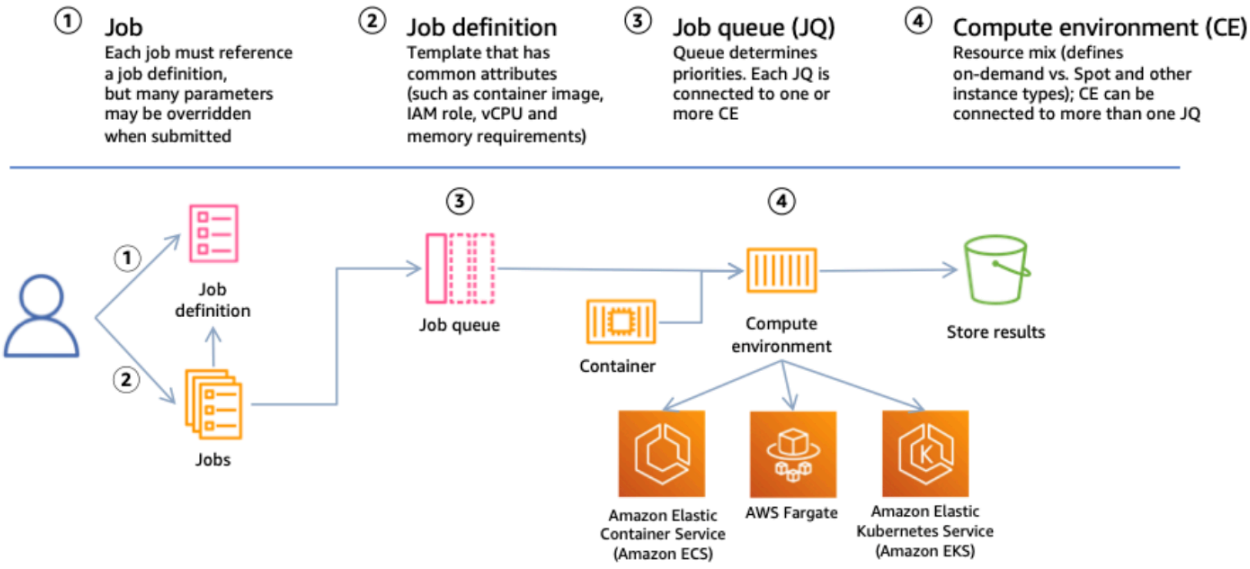
Table 2 — Task scheduling and infrastructure orchestration approaches

HPC hosting	Task scheduling approach	Infrastructure orchestration approach
On-Premises	Rapid task scheduling decisions to manage prioritiz ation and maximize utilizati on while minimizing queue times.	Static, a procurement and physical provisioning process run over weeks or months.
Cloud based	Focus on managing the task lifecycle, decisions around prioritization and queue times are minimized by dynamic orchestration.	Highly dynamic, capacity on-demand with 'pay as you go' pricing. Optimized for cost and performance through selection of instance type and procurement model.

When you plan a migration, a valid option is to migrate the on-premises solution first, and then consider optimizations. For example, an initial 'lift and shift' implementation might use Amazon EC2 On-Demand Instances to provision capacity, which yields some immediate benefits from elasticity. Some of the commercial schedulers also have integrations with AWS, which enable them to add and remove nodes according to demand.

When you are comfortable with running critical workloads on AWS, you can further optimize your implementation with options such as using more native services for data management, capacity provisioning, and orchestration. Ultimately, the scheduler might be in scope for replacement, at which point you can consider a few different approaches.

Though financial services workloads are often composed of very large volumes of relatively short-running calculations, there are some cases where longer-running calculations need to be scheduled. In these situations, AWS Batch could be a viable alternative or a complementary service. AWS Batch plans, schedules, and runs batch workloads while dynamically provisioning compute resources using containers. You can configure parallel computation and job dependencies to allow for workloads where the results of one job are used by another. AWS Batch is offered at no additional charge; only the AWS resources it consumes generate costs.



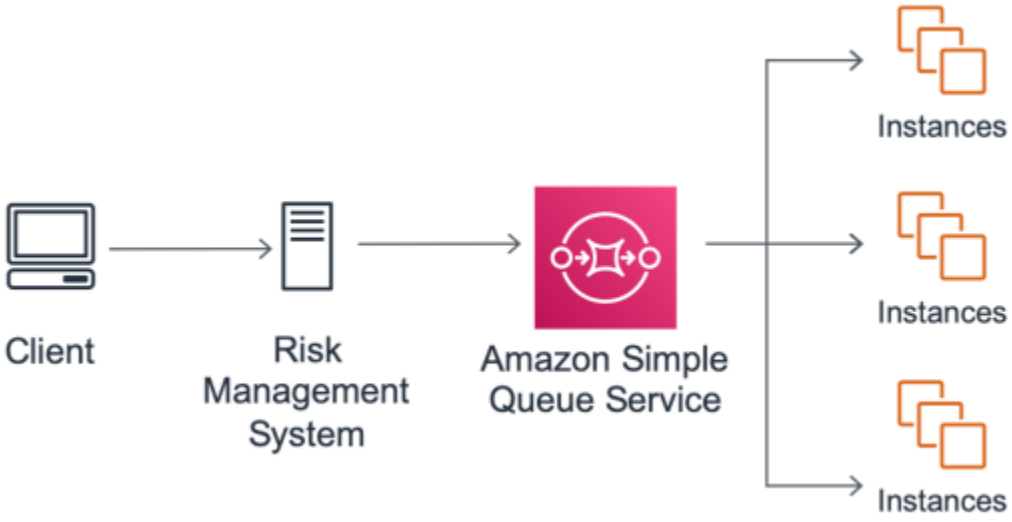
AWS Batch overview

AWS Batch decouples and templates the definition of jobs and submission into queues. Jobs are then run in compute environments linked to the queues. AWS Batch supports Amazon ECS, Fargate, and Amazon EKS as compute environments.

Customers looking to simplify their architecture might consider a queue-based architecture in which clients submit tasks to a stateful queue. This can then be serviced by an elastic group of *hungry worker* processes that take pending workloads, process them, and then return results. The Amazon SQS can be used for this purpose. Amazon SQS is a fully managed message queuing

service that is ideal for this type of decoupled architecture. As a serverless offering, it reduces the administrative burden of infrastructure management and offers seamless elastic scaling.

A simple HPC approach with Amazon SQS and Amazon EC2



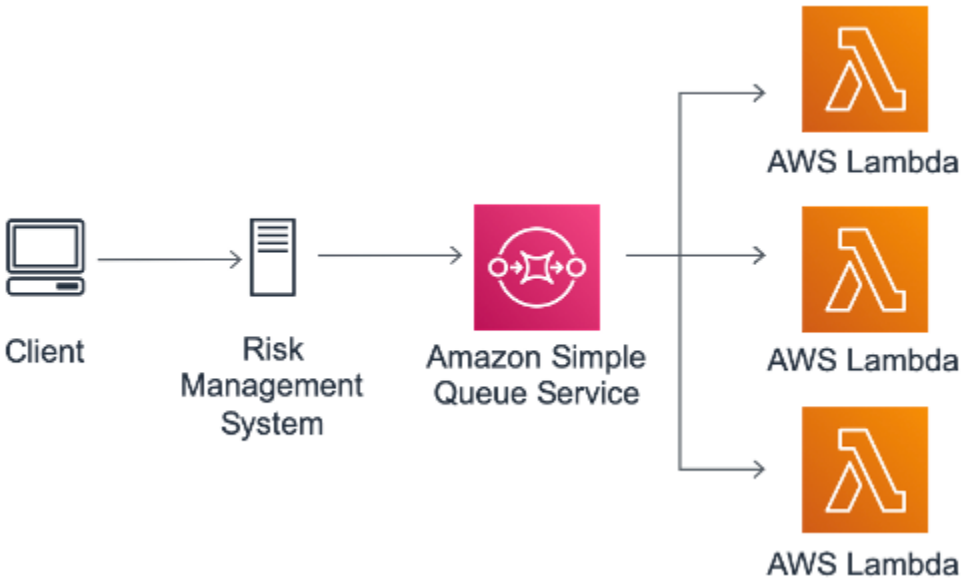
A simple HPC approach with Amazon SQS

Amazon SQS queues can be serviced by groups of Amazon EC2 instances that are managed by AWS Auto Scaling groups. You can configure the AWS Auto Scaling groups to scale capacity up or down based on metrics such as average CPU load, or the depth of the queue. AWS Auto Scaling groups can also incorporate provisioning strategies that can combine Amazon EC2 On-Demand Instances or Spot Instances to provide flexible and low-cost capacity.

A simple HPC approach with Amazon SQS and Lambda

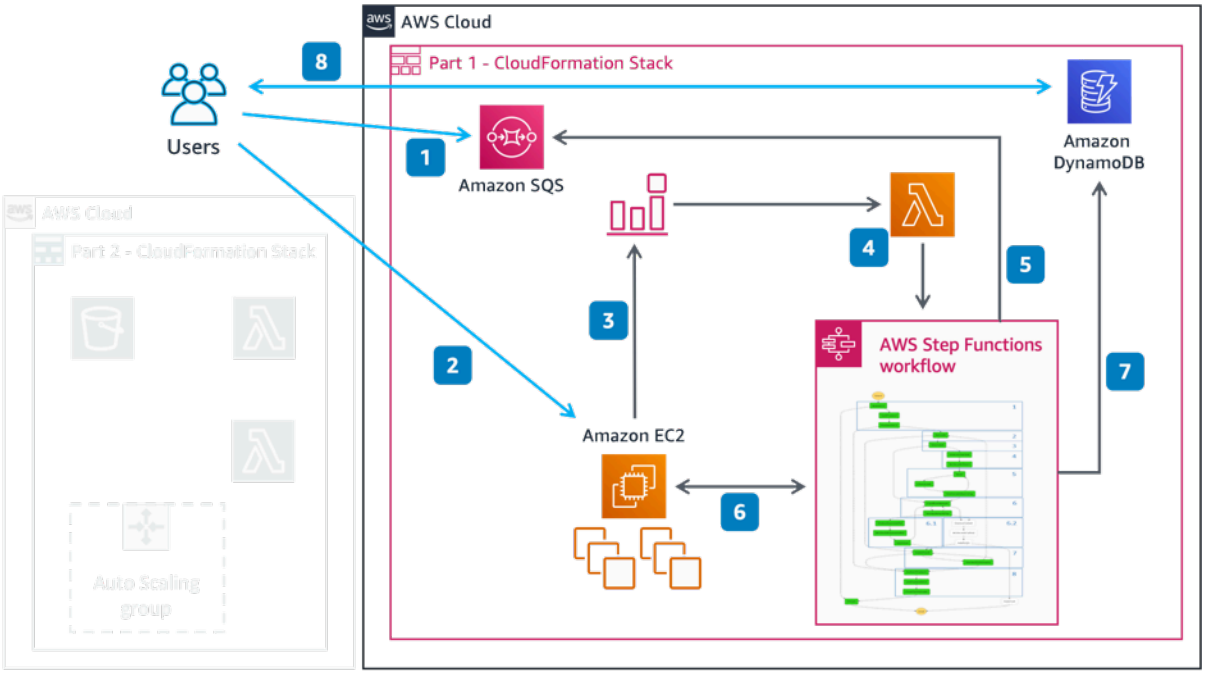
With serverless queuing provided by Amazon SQS, it's logical to think about serverless compute capacity. With AWS Lambda, you can run code without provisioning or managing any servers. This function-as-a-service product allows you to only pay for the computation time you consume.

You can also configure Lambda to process workloads from SQS, scaling out horizontally to consume messages in a queue. Lambda attempts to process the items from the queue as quickly as possible, and is constrained only by the maximum concurrency allowed by the account, memory, and runtime limits. You can also allocate up to 10GB of memory and six vCPUs to your functions, which also have support for the AVX2 instruction set. This makes Lambda functions suitable for a wide range of HPC applications.



A serverless, event-driven approach to HPC

A serverless, event-driven approach to HPC



A cloud-native serverless scheduler architecture

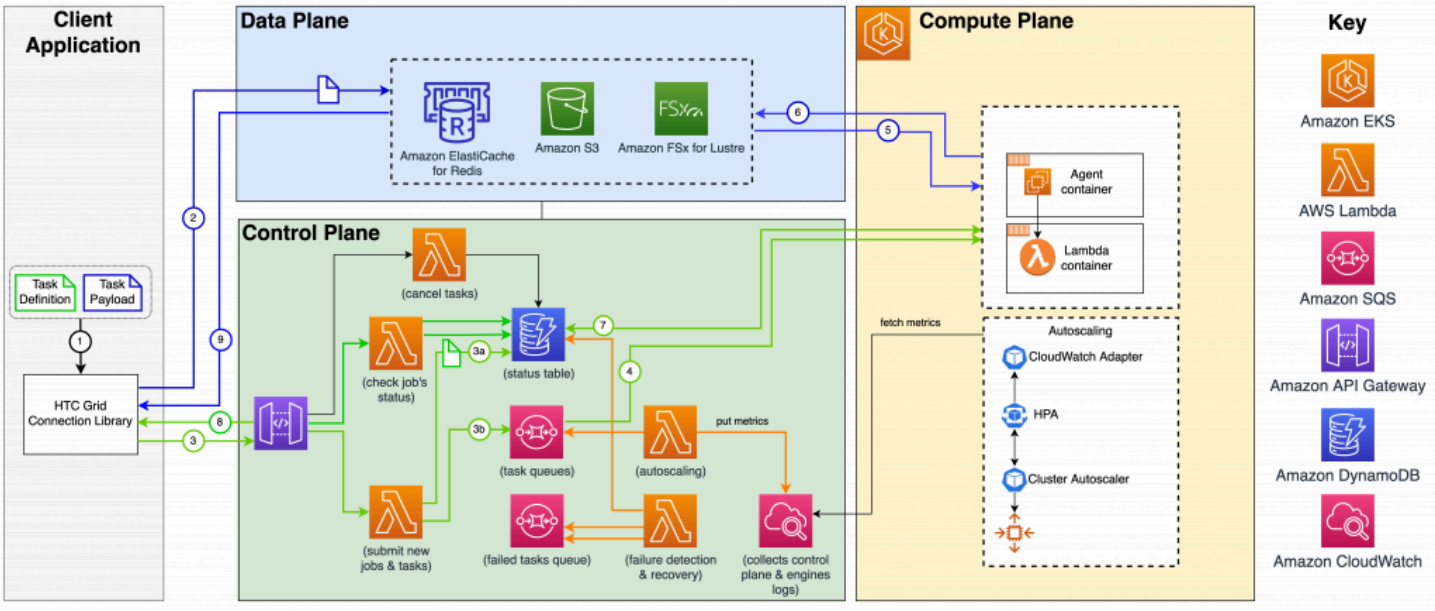
Some FSI customers choose to relay on the existing container orchestration platforms such as Amazon ECS and Amazon EKS for their grid systems. These are fully managed AWS solutions that provide scalability, availability, load balancing, and integration with other AWS services. Both

Amazon ECS and Amazon EKS have a concept of tasks (or jobs in EKS) which represent a unit of compute; in other words, a task can be submitted to a cluster, which in turn will deploy a container that will do the computation and will end the container when the task is completed.

Unfortunately, this approach does not work very well with the high volume of short running tasks, which is common for financial services. For example, when the duration of the computation is only one second, it is unacceptably wasteful to spend a few more seconds just to deploy a container to do the task. Moreover, some FSI workloads rely on computational context or state which can take time to be established at the first invocation of a container. Thus, sending tasks directly "as is" to ECS or EKS clusters does not always fit well into FSI workload characteristics.

Therefore, some FSI customers are looking into implementing their own pull-based mechanisms where a worker container persists between consecutive task computations; in other words, a worker container is created once and persists until there are no more tasks to compute. This requires implementing an additional layer to do the state management of each task.

Taking these concepts further, the blog Cloud-native, high throughput grid computing using the HTC-Grid solution describes an open-source, scalable, cloud-native high-throughput compute scheduling solution. HTC-Grid is an architectural blueprint that shows an example of a pull-base architecture built using different computational backends (ECS, EKS, or EC2). Note that HTC-Grid is not a supported AWS service offering; however, it can be used to validate the properties of such an architecture and evolve a custom solution based on fully managed, cloud-native services.



HTC-Grid - A cloud-native serverless scheduler architecture

When you explore these alternative cloud-native approaches, especially in comparison to established schedulers, it's important to consider all of the features required to run what can be a critical system. Metrics gathering, data management, and management tooling are only some of the typical requirements that must be addressed and should not be overlooked.

A key benefit of running HPC workloads on AWS is the flexibility of the offerings that enable you to combine various solutions to meet very specific needs. An HPC architect can use Amazon EC2 Savings Plans for long-running, stateful hosts. You can use Amazon EC2 On-Demand Instances for long-running tasks, or to secure capacity at the start of a batch. Additionally, you can provision Amazon EC2 Spot Instances to try to deliver a batch more quickly and at lower cost. Some workloads can then be directed to alternative platforms, such as GPU enabled instances or Lambda functions. You can optimize the overall mix of these options on a regular basis to adapt to the changing needs of your business.

Migration approaches, patterns, and anti-patterns

Many financial services organizations already have some form of HPC environment hosted in an on-premises data center. If you're migrating from such an implementation, it's important to consider what might be the best method to complete the migration. The optimal approach depends on the desired outcome, risk appetite, and timescale, but typically begins with one of the 6 Rs: Rehosting, Replatforming, Repurchasing, Refactoring/Re-architecting, and (to a lesser degree) Retiring, or Retaining (revisiting).

HPC cloud migrations typically progress through the following stages. Firstly, they begin on premises; then, bursting to cloud to augment capacity, customers 'lift and shift' whole clusters to the cloud; optimizations are then applied to make the most of new provisioning models before finally exploring 'cloud native' solution. The nuances and timings of each stage depends on the individual businesses involved.

The first stage customers explore is *Bursting* capacity. In this mode, very little changes with the existing on-premises HPC environment. However, at times of peak demand, Amazon EC2 instances can be created and added to the system to provide additional capacity. The trigger for the creation of these instances is usually either:

- **Scheduled** – If workloads are predictable in terms of timing and scale, then a simple schedule to add and remove a fixed number of hosts at predefined times can be effective. The schedule can be managed by an on-premises system, or with Amazon EventBridge rules.
- **Demand based** – In this mode, a component can monitor the performance of workloads, and add or remove capacity based on demand. If a task queue starts to increase, additional instances can be requested through the AWS API, and if the queue decreases, some instances can be removed.
- **Predictive** – In some cases, especially when the startup time for a new instance is long (perhaps because of very large package dependencies or complex OS builds), it might be desirable to use a simple machine learning model to analyze historic demand and determine when to bring capacity online. This approach is rare, but can work well when combined with a demand-based approach.

As customers build confidence in their ability to supplement existing capacity with cloud-based instances, they often make a decision to complete a migration. However, with existing on-premises hardware still available, customers want to keep the value of that infrastructure before it can be

decommissioned. In this case, it can make sense to provision a new strategic grid — with all of the same scheduler components — into the cloud, and retain the existing on-premises grid. It's then left to the upstream clients to direct workloads accordingly, switching to the cloud-based grid as the on-premises capacity is gradually retired.

When you have completed migration and are running all of their HPC workloads in the cloud, the on-premises infrastructure can be removed. At this point, you have completed a *Rehosting* approach. When your infrastructure is in the cloud, you then have the flexibility to look at *Replatforming* or *Refactoring* your environment. The ability to build entirely new architectures in the cloud alongside existing production systems means that new approaches can be fully tested before they're put into production.

One anti-pattern that's occasionally proposed by customers involves *platform stacking*. In this approach, solutions such as virtualization and/or container platforms are placed under the HPC platform to try to create portability or parity between cloud-based systems and on-premises systems. This approach can have some disadvantages:

- **Computational inefficiency** – By adding more layers between the analytics binaries and CPUs performance, computational efficiency is inevitably degraded as CPU cycles are consumed by the abstraction layers.
- **Licensing costs** – HPC environments are large and continue to grow. Though enterprise licenses can keep the upfront costs of using these technologies very low, the large number of CPU cores involved in HPC workloads can mean significant additional costs when the licenses are due for renewal.
- **Management overhead** – In the simplest approach, an Amazon EC2 instance can be created on demand using an Amazon Linux 2 AMI. This AMI is patched and up to date and because it exists for just a few hours, it requires no further management. However, by building HPC stacks on top of other abstractions, those long-running layers need patching and upgrading, and when multiple layers are involved, the scope for disruption through planned maintenance or an unplanned outage increases significantly.
- **Scaling challenges** – Amazon EC2 instances can be available very quickly on demand. If scaling out involves the creation of a complex stack before processes can run, this adds to the billing time of the instance before useful work can be done. In worst-case scenarios, there can be a temptation to leave large numbers of instances running so that they're available if additional workloads arise.
- **Optimization challenges** – HPC systems are already complex, especially when supporting huge volumes of variable workloads with different CPU and memory requirements. Knowing where

CPU and memory resources are consumed is vital to identifying bottlenecks or debugging failures. If an HPC platform is based on a series of abstraction layers, this can introduce additional variables that make it difficult to see where inefficiencies exist, and as a result they might never be found.

- **Security challenges** – Securing a more complex stack can be challenging because there are more components to configure, monitor, and maintain to ensure the integrity of the system.

By defining portability in terms of a virtual machine image or a Docker image, you can find a good balance of portability while off-setting some of the disadvantages through the use of cloud-native virtualization with Amazon EC2 and/or container management solutions such as Amazon ECS and EKS, especially when combined with AWS Fargate.

Keeping HPC systems as simple as possible provides the best performance at the lowest cost. Most HPC solutions are already platforms by design and offer portability through simple deployment patterns to standard operating systems.

Once fully in the cloud, the next logical step for an HPC team is to focus on optimization. Ultimately, getting the calculations done at the right time and at the lowest cost is a key measure of success and the cloud offers more levers to control how this is achieved, in particular in relationship to the supply of compute. The process of optimization could be triggered by a new customer workload pattern, the availability of a new EC2 Instance type or a price reduction. Fortunately, the elastic nature of cloud means that it's easy to experiment, to review the results, and to make changes without the risk of a costly mistake.

Once the team is confident in optimizing a shared HPC cluster, it's common to then review the multi-tenant approach entirely. Multi-tenancy was instrumental in driving up the overall utilization of large on-premises HPC clusters, allowing any user group with pending tasks to make use of idle resources. However, the cloud's ability to right size according to the demands of the day means that there should be very little idle capacity available. This has led to a trend where an organization's consumers of HPC have their own individual clusters, each dynamically sized and optimized to their individual needs without any increase in aggregate costs.

Such federated models typically see individual consumer groups (like a trading desk or particular business line) have their own AWS accounts within which they are free to optimize for their individual workloads while benefitting from standardized tooling and expert guidance from the central HPC team. Separating consumers increases agility as changes need only be tested within a more limited scope. Additionally, the separate accounts make it easier to understand the costs associated with an instance of the HPC environment and the cost impact of any changes made.

Lastly, customers are increasingly looking to the future as they make longer-term plans for HPC. The overall trend is towards serverless technologies which can abstract away many of the concerns of availability and capacity.

Conclusion

AWS has a long history of helping customers from various industries — including financial services — to optimize their HPC workloads. This experience over many years from customers with diverse requirements has directly contributed to the products and services offered today, and will continue to do so. AWS regularly accommodates very large-scale requests for Amazon EC2 instances. Some of these clusters are large enough to be recognized among the world's largest supercomputers, scaling to millions of vCPUs in support of batch processing.

HPC platforms are crucial enablers for many different types of financial services organizations including capital markets, insurance, banking and payments. However, as demands on these platforms increase as a result of regulatory requirements it's clear that the traditional approaches to provisioning HPC infrastructure are inefficient and ultimately unsustainable. Constraints on capital and capital expenditure further compound the challenge.

By migrating these systems to AWS, customers benefit from a wide variety of compute instances and relevant services, but also from a fundamental change in the delivery of compute capacity. This new approach offers tremendous flexibility, both in terms of the management of workloads that vary day-to-day, but also in the overall approach to cost optimizations, security, availability, and operations.

HPC workloads already have much in common with stateless, function-as-a-service architectural patterns. Just as financial services moved from local calculations to clusters and into grids, they are starting to explore decentralized, *serverless* approaches. As scaling become transparent, bottlenecks will continue to be removed until processing becomes near real-time. This trend is explored further in the blog post The Convergent Evolution of Grid Computing in Financial Services

If you have challenges with the scale, cost, and capacity challenges of managing a high performance computing system today, AWS has a number of services and partner relationships that can help.

To learn more, you can contact AWS Financial Services through the AWS Financial Services – Contact Sales form.

Contributors

Contributors to this document include:

- Alex Kimber, Global Financial Services, Principal Technologist for HPC, Amazon Web Services
- Carlos Manzanedo Rueda, Solutions Architect Leader, Efficient Computing, Amazon Web Services
- Richard Nicholson, Solutions Architect, Global Financial Services, Amazon Web Services
- Ian Meyers, Solutions Architect Head of Technology, Amazon Web Services

Further reading

For additional information, see:

- AWS Well-Architected Framework
- AWS Well-Architected Framework – HPC Lens
- AWS Well-Architected Framework – Financial Services Industry Lens
- AWS HPC Blog
- AWS Architecture Center

Document history

To be notified about updates to this whitepaper, subscribe to the RSS feed.

Change	Description	Date
Whitepaper updated	Updates throughout whitepaper to reflect current AWS capabilities and service features.	August 9, 2024
Whitepaper updated	Updates to reflect AWS service improvements, more modern and inclusive terminology, and new cloud-native architectures.	August 24, 2021
Whitepaper updated	Updates to services, diagrams, and topology.	September 1, 2019
Whitepaper updated	Updates to services and topology.	January 1, 2016
Initial publication	Whitepaper first published.	January 1, 2015

Note

To subscribe to RSS updates, you must have an RSS plug-in enabled for the browser that you are using.

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. 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.

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.