AWS Well-Architected Framework

High Performance Computing Lens

High Performance Computing Lens: AWS Well-Architected Framework

Table of Contents

High Performance Computing Lens - AWS Well-Architected Framework

Publication date: **December 2019** (*Document Revisions*)

Abstract

This document describes the High Performance Computing (HPC) Lens for the AWS Well-Architected Framework. The document covers common HPC scenarios and identifies key elements to ensure that your workloads are architected according to best practices.

Introduction

The AWS Well-Architected Framework helps you understand the pros and cons of decisions you make while building systems on AWS. Use the Framework to learn architectural best practices for designing and operating reliable, secure, efficient, and cost-effective systems in the cloud. The Framework provides a way for you to consistently measure your architectures against best practices and identify areas for improvement. We believe that having well-architected systems greatly increases the likelihood of business success.

In this “Lens” we focus on how to design, deploy, and architect your High Performance Computing (HPC) workloads in the AWS Cloud. HPC workloads run exceptionally well in the cloud. The natural ebb and flow and bursting characteristic of HPC workloads make them well suited for pay-as-you-go cloud infrastructure. The ability to fine tune cloud resources and create cloud-native architectures naturally accelerates the turnaround of HPC workloads.

For brevity, we only cover details from the Well-Architected Framework that are specific to HPC workloads. We recommend that you consider best practices and questions from the AWS Well-Architected Framework whitepaper when designing your architecture.

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 and operating HPC in a cloud environment.

Definitions

The AWS Well-Architected Framework is based on six pillars: operational excellence, security, reliability, performance efficiency, cost optimization, and sustainability. When architecting solutions, you make tradeoffs between pillars based upon your business context. These business decisions can drive your engineering priorities. You might reduce cost at the expense of reliability in development environments, or, for mission-critical solutions, you might optimize reliability with increased costs. Security and operational excellence are generally not traded off against other pillars.

Throughout this paper, we make the crucial distinction between loosely coupled – sometimes referred to as high-throughput computing (HTC) in the community – and tightly coupled workloads. We also cover server-based and serverless designs. Refer to the Scenarios section for a detailed discussion of these distinctions.

Some vocabulary of the AWS Cloud may differ from common HPC terminology. For example, HPC users may refer to a server as a “node” while AWS refers to a virtual server as an “instance.” When HPC users commonly speak of “jobs,” AWS refers to them as “workloads.”

AWS documentation uses the term “vCPU” synonymously with a “thread” or a “hyperthread” (or *half* of a physical core). Don’t miss this factor of 2 when quantifying the performance or cost of an HPC application on AWS.

Cluster placement groups are an AWS method of grouping your compute instances for applications with the highest network requirements. A placement group is not a physical hardware element. It is simply a logical rule keeping all nodes within a low latency radius of the network.

The AWS Cloud infrastructure is built around **Regions** and **Availability Zones**. A Region is a physical location in the world where we have multiple Availability Zones. Availability Zones consist of one or more discrete data centers, each with redundant power, networking, and connectivity, housed in separate facilities. Depending on the characteristics of your HPC workload, you may want your cluster to span Availability Zones (increasing reliability) or stay within a single Availability Zone (emphasizing low-latency).

General Design Principles

In traditional computing environments, architectural decisions are often implemented as static, one-time events, sometimes with no major software or hardware upgrades during a computing system's lifetime. As a project and its context evolve, these initial decisions may hinder the system's ability to meet changing business requirements.

It's different in the cloud. A cloud infrastructure can grow as the project grows, allowing for a continuously optimized capability. In the cloud, the capability to automate and test on demand lowers the risk of impact from infrastructure design changes. This allows systems to evolve over time so that projects can take advantage of innovations as a standard practice.

The Well-Architected Framework proposes a set of general design principles to facilitate good design in the cloud with high-performance computing:

- **Dynamic architectures**: Avoid frozen, static architectures and cost estimates that use a steady-state model. Your architecture must be dynamic: growing and shrinking to match your demands for HPC over time. Match your architecture design and cost analysis explicitly to the natural cycles of HPC activity. For example, a period of intense simulation efforts might be followed by a reduction in demand as the work moves from the design phase to the lab. Or, a long and steady data accumulation phase might be followed by a large-scale analysis and data reduction phase. Unlike many traditional supercomputing centers, the AWS Cloud helps you avoid long queues, lengthy quota applications, and restrictions on customization and software installation. Many HPC endeavors are intrinsically bursty and well-matched to the cloud paradigms of elasticity and pay-as-you-go. The elasticity and pay-as-you-go model of AWS eliminates the painful choice between oversubscribed systems (waiting in queues) or idle systems (wasted money). Environments, such as compute clusters, can be "right-sized" for a given need at any given time.
- **Align the procurement model to the workload**: AWS makes a range of compute procurement models available for the various HPC usage patterns. Selecting the correct model ensure that you are only paying for what you need. For example, a research institute might run the same weather forecast application in different ways:
  - An academic research project investigates the role of a weather variable with a large number of parameter sweeps and ensembles. These simulations are not urgent, and cost is a primary concern. They are a great match for Amazon EC2 Spot Instances. Spot Instances let you take advantage of Amazon EC2 unused capacity and are available at up to a 90% discount compared to On-Demand prices.

  - During the wildfire season, up-to-the-minute local wind forecasts ensure the safety of firefighters. Every minute of delay in the simulations decreases their chance of safe evacuation. On-Demand Instances must be used for these simulations to allow for the bursting of analyses and ensure that results are obtained without interruption.
  - Every morning, weather forecasts are run for television broadcasts in the afternoon. Scheduled Reserved Instances can be used to make sure that the needed capacity is available every day at the right time. Use of this pricing model provides a discount compared with On-Demand Instances.
- **Start from the data**: Before you begin designing your architecture, you must have a clear picture of the data. Consider data origin, size, velocity, and updates. A holistic optimization of performance and cost focuses on compute and includes data considerations. AWS has a strong offering of data and related services, including data visualization, which enables you to extract the most value from your data.
- **Automate to simplify architectural experimentation:** Automation through code allows you to create and replicate your systems at low cost and avoid the expense of manual effort. You can track changes to your code, audit their impact, and revert to previous versions when necessary. The ability to easily experiment with infrastructure allows you to optimize the architecture for performance and cost. AWS offers tools, such as AWS ParallelCluster, that help you get started with treating your HPC cloud infrastructure as code.
- **Enable collaboration**: HPC work often occurs in a collaborative context, sometimes spanning many countries around the world. Beyond immediate collaboration, methods and results are often shared with the wider HPC and scientific community. It's important to consider in advance which tools, code, and data may be shared, and with whom. The delivery methods should be part of this design process. For example, workflows can be shared in many ways on AWS: you can use Amazon Machine Images (AMIs), Amazon Elastic Block Store (Amazon EBS) snapshots, Amazon Simple Storage Service (Amazon S3) buckets, AWS CloudFormation templates, AWS ParallelCluster configuration files, AWS Marketplace products, and scripts. Take full advantage of the AWS security and collaboration features that make AWS an excellent environment for you and your collaborators to solve your HPC problems. This helps your computing solutions and datasets achieve a greater impact by securely sharing within a selective group or publicly sharing with the broader community.
- **Use cloud-native designs**: It is usually unnecessary and suboptimal to replicate your on-premises environment when you migrate workloads to AWS. The breadth and depth of AWS services enables HPC workloads to run in new ways using new design patterns and cloud-native solutions. For example, each user or group can use a separate cluster, which can independently

scale depending on the load. Users can rely on a managed service, like AWS Batch, or serverless computing, like AWS Lambda, to manage the underlying infrastructure. Consider not using a traditional cluster scheduler, and instead use a scheduler only if your workload requires it. In the cloud, HPC clusters do not require permanence and can be ephemeral resources. When you automate your cluster deployment you can terminate one cluster and launch a new one quickly with the same or different parameters. This method creates environments as necessary.

- **Test real-world workloads**: The only way to know how your production workload will perform in the cloud is to test it on the cloud. Most HPC applications are complex, and their memory, CPU, and network patterns often can't be reduced to a simple test. Also, application requirements for infrastructure vary based on which application solvers (mathematical methods or algorithms) your models use, the size and complexity of your models, etc. For this reason, generic benchmarks aren't reliable predictors of actual HPC production performance. Similarly, there is little value in testing an application with a small benchmark set or "toy problem." With AWS, you only pay for what you actually use; therefore, it is feasible to do a realistic proof-of-concept with your own representative models. A major advantage of a cloud-based platform is that a realistic, full-scale test can be done before migration.
- **Balance time-to-results and cost reduction**: Analyze performance using the most meaningful parameters: time and cost. Focus on cost optimization should be used for workloads that are not time-sensitive. Spot Instances are usually the least expensive method for non-time-critical workloads. For example, if a researcher has a large number of lab measurements that must be analyzed sometime before next year's conference, Spot Instances can help analyze the largest possible number of measurements within the fixed research budget. Conversely, for time-critical workloads, such as emergency response modeling, cost optimization can be traded for performance, and instance type, procurement model, and cluster size should be chosen for lowest and most immediate execution time. If comparing platforms, it's important to take the entire time-to-solution into account, including non-compute aspects such as provisioning resources, staging data, or, in more traditional environments, time spent in job queues.

Scenarios

HPC cases are typically complex computational problems that require parallel-processing techniques. To support the calculations, a well-architected HPC infrastructure is capable of sustained performance for the duration of the calculations. HPC workloads span traditional applications, like genomics, computational chemistry, financial risk modeling, computer aided engineering, weather prediction and seismic imaging, as well as emerging applications, like machine learning, deep learning, and autonomous driving. Still, the traditional grids or HPC clusters that support these calculations are remarkably similar in architecture with select cluster attributes optimized for the specific workload. In AWS, the network, storage type, compute (instance) type, and even deployment method can be strategically chosen to optimize performance, cost, and usability for a particular workload.

HPC is divided into two categories based on the degree of interaction between the concurrently running parallel processes: loosely coupled and tightly coupled workloads. Loosely coupled HPC cases are those where the multiple or parallel processes don't strongly interact with each other in the course of the entire simulation. Tightly coupled HPC cases are those where the parallel processes are simultaneously running and regularly exchanging information between each other at each iteration or step of the simulation.

With loosely coupled workloads, the completion of an entire calculation or simulation often requires hundreds to millions of parallel processes. These processes occur in any order and at any speed through the course of the simulation. This offers flexibility on the computing infrastructure required for loosely coupled simulations.

Tightly coupled workloads have processes that regularly exchange information at each iteration of the simulation. Typically, these tightly coupled simulations run on a homogenous cluster. The total core or processor count can range from tens, to thousands, and occasionally to hundreds of thousands if the infrastructure allows. The interactions of the processes during the simulation place extra demands on the infrastructure, such as the compute nodes and network infrastructure.

The infrastructure used to run the huge variety of loosely and tightly coupled applications is differentiated by its ability for process interactions across nodes. There are fundamental aspects that apply to both scenarios and specific design considerations for each. Consider the following fundamentals for both scenarios when selecting an HPC infrastructure on AWS:

- **Network**: Network requirements can range from cases with low requirements, such as loosely coupled applications with minimal communication traffic, to tightly coupled and massively parallel applications that require a performant network with large bandwidth and low latency.
- **Storage**: HPC calculations use, create, and move data in unique ways. Storage infrastructure must support these requirements during each step of the calculation. Input data is frequently stored on startup, more data is created and stored while running, and output data is moved to a reservoir location upon run completion. Factors to be considered include data size, media type, transfer speeds, shared access, and storage properties (for example, durability and availability). It is helpful to use a shared file system between nodes. For example, using a Network File System (NFS) share, such as Amazon Elastic File System (EFS), or a Lustre file system, such as Amazon FSx for Lustre.
- **Compute**: The Amazon EC2 instance type defines the hardware capabilities available for your HPC workload. Hardware capabilities include the processor type, core frequency, processor features (for example, vector extensions), memory-to-core ratio, and network performance. On AWS, an instance is considered to be the same as an HPC node. These terms are used interchangeably in this whitepaper.
  - AWS offers managed services with the ability to access compute without the need to choose the underlying EC2 instance type. AWS Lambda and AWS Fargate are compute services that allow you to run workloads without having to provision and manage the underlying servers.
- **Deployment**: AWS provides many options for deploying HPC workloads. Instances can be manually launched from the AWS Management Console. For an automated deployment, a variety of Software Development Kits (SDKs) is available for coding end-to-end solutions in different programming languages. A popular HPC deployment option combines bash shell scripting with the AWS Command Line Interface (AWS CLI).
  - AWS CloudFormation templates allow the specification of application-tailored HPC clusters described as code so that they can be launched in minutes. AWS ParallelCluster is open-source software that coordinates the launch of a cluster through CloudFormation with already installed software (for example, compilers and schedulers) for a traditional cluster experience.
  - AWS provides managed deployment services for container-based workloads, such as Amazon Elastic Container Service (Amazon ECS), Amazon Elastic Kubernetes Service (Amazon EKS), AWS Fargate, and AWS Batch.
  - Additional software options are available from third-party companies in the AWS Marketplace and the AWS Partner Network (APN).

Cloud computing makes it easy to experiment with infrastructure components and architecture design. AWS strongly encourages testing instance types, EBS volume types, deployment methods, etc., to find the best performance at the lowest cost.

Loosely Coupled Scenarios

A loosely coupled workload entails the processing of a large number of smaller jobs. Generally, the smaller job runs on one node, either consuming one process or multiple processes with shared memory parallelization (SMP) for parallelization within that node.

The parallel processes, or the iterations in the simulation, are post-processed to create one solution or discovery from the simulation. Loosely coupled applications are found in many areas, including Monte Carlo simulations, image processing, genomics analysis, and Electronic Design Automation (EDA).

The loss of one node or job in a loosely coupled workload usually doesn't delay the entire calculation. The lost work can be picked up later or omitted altogether. The nodes involved in the calculation can vary in specification and power.

A suitable architecture for a loosely coupled workload has the following considerations:

- **Network**: Because parallel processes do not typically interact with each other, the feasibility or performance of the workloads is not sensitive to the bandwidth and latency capabilities of the network between instances. Therefore, clustered placement groups are not necessary for this scenario because they weaken the resiliency without providing a performance gain.
- **Storage**: Loosely coupled workloads vary in storage requirements and are driven by the dataset size and desired performance for transferring, reading, and writing the data.
- **Compute**: Each application is different, but in general, the application's memory-to-compute ratio drives the underlying EC2 instance type. Some applications are optimized to take advantage of graphics processing units (GPUs) or field-programmable gate array (FPGA) accelerators on EC2 instances.
- **Deployment**: Loosely coupled simulations often run across many — sometimes millions — of compute cores that can be spread across Availability Zones without sacrificing performance. Loosely coupled simulations can be deployed with end-to-end services and solutions such as AWS Batch and AWS ParallelCluster, or through a combination of AWS services, such as Amazon Simple Queue Service (Amazon SQS), Auto Scaling, AWS Lambda, and AWS Step Functions.

Tightly Coupled Scenarios

Tightly coupled applications consist of parallel processes that are dependent on each other to carry out the calculation. Unlike a loosely coupled computation, all processes of a tightly coupled simulation iterate together and require communication with one another. An iteration is defined as one step of the overall simulation. Tightly coupled calculations rely on tens to thousands of processes or cores over one to millions of iterations. The failure of one node usually leads to the failure of the entire calculation. To mitigate the risk of complete failure, application-level checkpointing regularly occurs during a computation to allow for the restarting of a simulation from a known state.

These simulations rely on a Message Passing Interface (MPI) for interprocess communication. Shared Memory Parallelism via OpenMP can be used with MPI. Examples of tightly coupled HPC workloads include: computational fluid dynamics, weather prediction, and reservoir simulation.

A suitable architecture for a tightly coupled HPC workload has the following considerations:

- **Network**: The network requirements for tightly coupled calculations are demanding. Slow communication between nodes results in the slowdown of the entire calculation. The largest instance size, enhanced networking, and cluster placement groups are required for consistent networking performance. These techniques minimize simulation runtimes and reduce overall costs. Tightly coupled applications range in size. A large problem size, spread over a large number of processes or cores, usually parallelizes well. Small cases, with lower total computational requirements, place the greatest demand on the network. Certain Amazon EC2 instances use the Elastic Fabric Adapter (EFA) as a network interface that enables running applications that require high levels of internode communications at scale on AWS. EFA's custom-built operating system bypass hardware interface enhances the performance of interinstance communications, which is critical to scaling tightly coupled applications.
- **Storage**: Tightly coupled workloads vary in storage requirements and are driven by the dataset size and desired performance for transferring, reading, and writing the data. Temporary data storage or scratch space requires special consideration.
- **Compute**: EC2 instances are offered in a variety of configurations with varying core to memory ratios. For parallel applications, it is helpful to spread memory-intensive parallel simulations across more compute nodes to lessen the memory-per-core requirements and to target the best performing instance type. Tightly coupled applications require a homogenous cluster built from similar compute nodes. Targeting the largest instance size minimizes internode network latency while providing the maximum network performance when communicating between nodes.

- **Deployment**: A variety of deployment options are available. End-to-end automation is achievable, as is launching simulations in a "traditional" cluster environment. Cloud scalability enables you to launch hundreds of large multi-process cases at once, so there is no need to wait in a queue. Tightly coupled simulations can be deployed with end-to-end solutions such as AWS Batch and AWS ParallelCluster, or through solutions based on AWS services such as CloudFormation or EC2 Fleet.

Reference Architectures

Many architectures apply to both loosely coupled and tightly coupled workloads and may require slight modifications based on the scenario. Traditional, on-premises clusters force a one-size-fits-all approach to the cluster infrastructure. However, the cloud offers a wide range of possibilities and allows for optimization of performance and cost. In the cloud, your configuration can range from a traditional cluster experience with a scheduler and login node to a cloud-native architecture with the advantages of cost efficiencies obtainable with cloud-native solutions. Five reference architectures are below.

Topics

Traditional Cluster Environment

Many users begin their cloud journey with an environment that is similar to traditional HPC environments. The environment often involves a login node with a scheduler to launch jobs.

A common approach to traditional cluster provisioning is based on an AWS CloudFormation template for a compute cluster combined with customization for a user's specific tasks. AWS ParallelCluster is an example of an end-to-end cluster provisioning capability based on AWS CloudFormation. Although the complex description of the architecture is hidden inside the template, typical configuration options allow the user to select the instance type, scheduler, or bootstrap actions, such as installing applications or synchronizing data. The template can be

constructed and executed to provide an HPC environment with the “look and feel” of conventional HPC clusters, but with the added benefit of scalability. The login node maintains the scheduler, shared file system, and running environment. Meanwhile, an automatic scaling mechanism allows additional instances to spin up as jobs are submitted to a job queue. As instances become idle, they are automatically terminated.

A cluster can be deployed in a persistent configuration or treated as an ephemeral resource. Persistent clusters are deployed with a login instance and a compute fleet that can either be a fixed sized, or tied to an Auto Scaling group which increases and decreases the compute fleet depending on the number of submitted jobs. Persistent clusters always have some infrastructure running. Alternatively, clusters can be treated as ephemeral where each workload runs on its own cluster. Ephemeral clusters are enabled by automation. For example, a bash script is combined with the AWS CLI, or a Python script with the AWS SDK provides end-to-end case automation. For each case, resources are provisioned and launched, data is placed on the nodes, jobs are run across multiple nodes, and the case output is either retrieved automatically or sent to Amazon S3. Upon completion of the job, the infrastructure is terminated. These clusters treat infrastructure as code, optimize costs, and allow for complete version control of infrastructure changes.

Traditional cluster architectures can be used for loosely and tightly coupled workloads. For best performance, tightly coupled workloads must use a compute fleet in a clustered placement group with homogenous instance types.

Reference Architecture

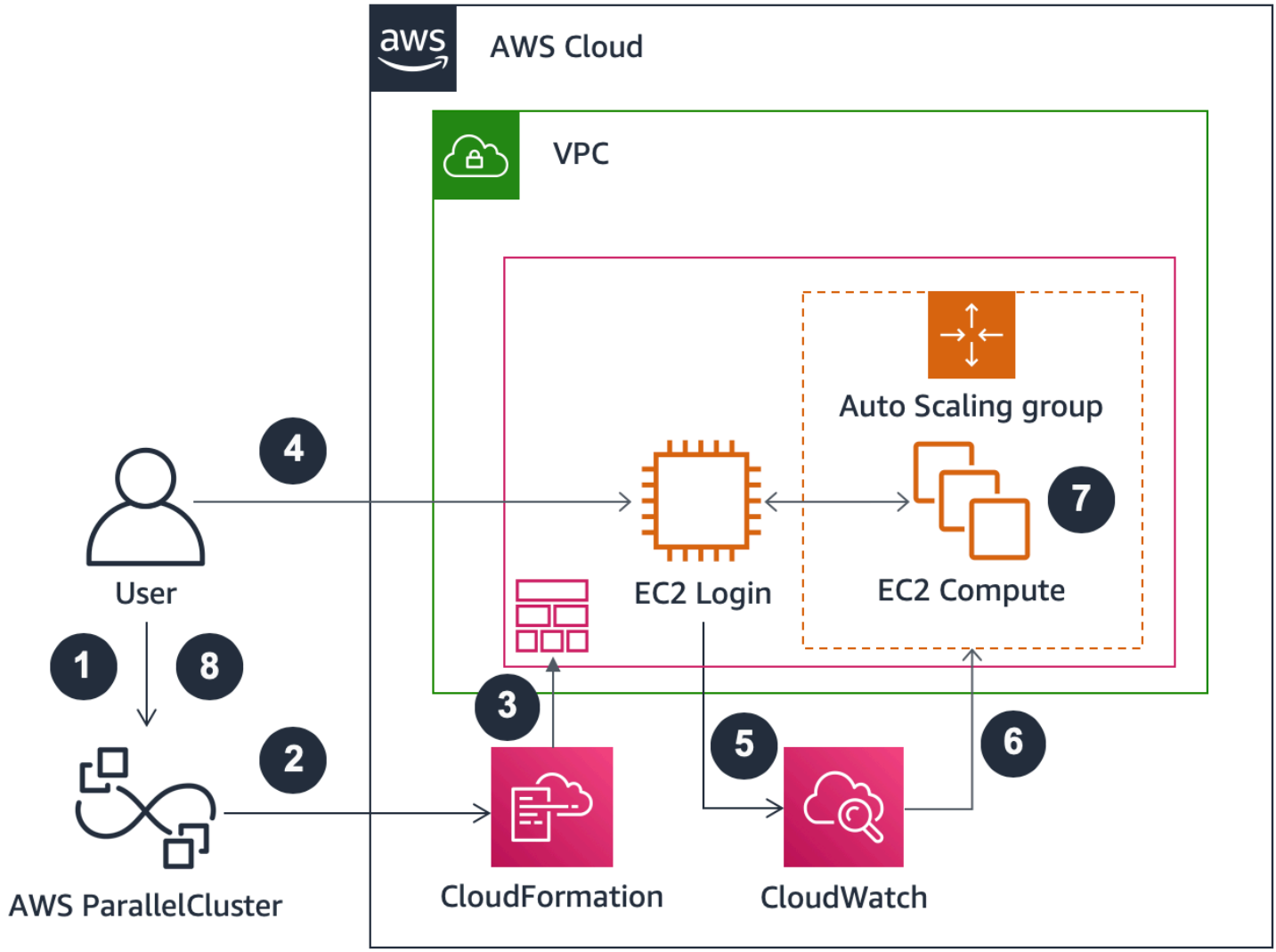

Figure 1: Traditional cluster deployed with AWS ParallelCluster

Workflow steps:

1. User initiates the creation of a cluster through the AWS ParallelCluster CLI and specification in the configuration file.
2. AWS CloudFormation builds the cluster architecture as described in the cluster template file, where the user contributed a few custom settings (for example, by editing a configuration file or using a web interface).
3. AWS CloudFormation deploys the infrastructure from EBS snapshots created with customized HPC software/applications that cluster instances can access through an NFS export.
4. The user logs into the login instance and submits jobs to the scheduler (for example, SGE, Slurm).

5. The login instance emits metrics to CloudWatch based on the job queue size.
6. CloudWatch triggers Auto Scaling events to increase the number of compute instances if the job queue size exceeds a threshold.
7. Scheduled jobs are processed by the compute fleet.
8. [Optional] User initiates cluster deletion and termination of all resources.

Batch-Based Architecture

AWS Batch is a fully managed service that enables you to run large-scale compute workloads in the cloud without provisioning resources or managing schedulers. AWS Batch enables developers, scientists, and engineers to easily and efficiently run hundreds of thousands of batch computing jobs on AWS. AWS Batch dynamically provisions the optimal quantity and type of compute resources (for example, CPU or memory-optimized instances) based on the volume and specified resource requirements of the batch jobs submitted. It plans, schedules, and executes your batch computing workloads across the full range of AWS compute services and features, such as Amazon EC2 and Spot Instances. Without the need to install and manage the batch computing software or server clusters necessary for running your jobs, you can focus on analyzing results and gaining new insights.

With AWS Batch, you package your application in a container, specify your job's dependencies, and submit your batch jobs using the AWS Management Console, the CLI, or an SDK. You can specify execution parameters and job dependencies and integrate with a broad range of popular batch computing workflow engines and languages (for example, Pegasus WMS, Luigi, and AWS Step Functions). AWS Batch provides default job queues and compute environment definitions that enable you to get started quickly.

An AWS Batch based architecture can be used for both loosely and tightly coupled workloads. Tightly coupled workloads should use Multi-node Parallel Jobs in AWS Batch.

Reference Architecture

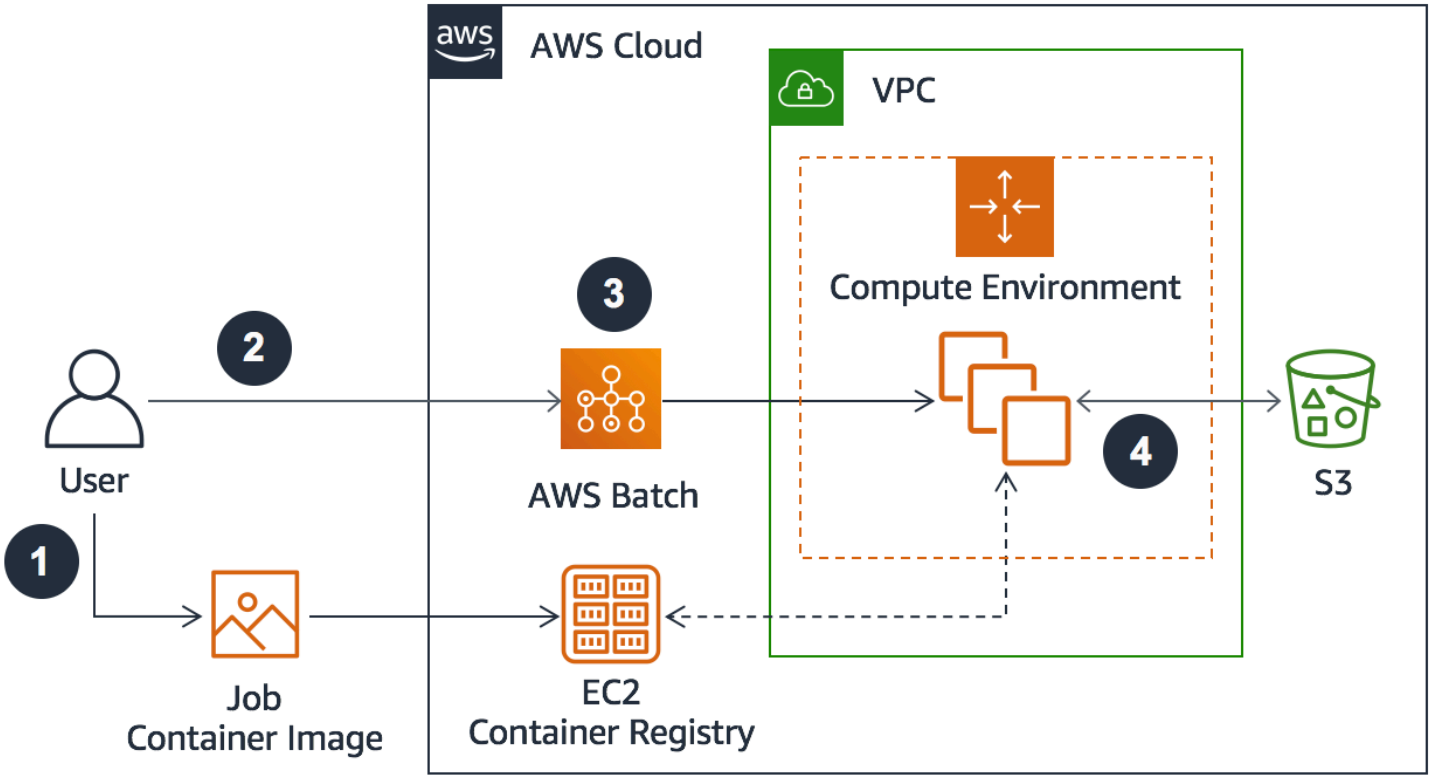

Figure 2: Example AWS Batch architecture

Workflow steps:

1. User creates a job container, uploads the container to the Amazon Elastic Container Registry or another container registry (for example, DockerHub), and creates a job definition to AWS Batch.
2. User submits jobs to a job queue in AWS Batch.
3. AWS Batch pulls the image from the container registry and processes the jobs in the queue
4. Input and output data from each job is stored in an S3 bucket.

Queue-Based Architecture

Amazon SQS is a fully managed message queuing service that makes it easy to decouple pre-processing steps from compute steps and post-processing steps. Building applications from individual components that perform discrete functions improves scalability and reliability. Decoupling components is a best practice for designing modern applications. Amazon SQS frequently lies at the heart of cloud-native loosely coupled solutions.

Amazon SQS is often orchestrated with AWS CLI or AWS SDK scripted solutions for the deployment of applications from the desktop without users interacting with AWS components directly. A

queue-based architecture with Amazon SQS and Amazon EC2 requires self-managed compute infrastructure, in contrast with a service-managed deployment, such as AWS Batch.

A queue-based architecture is best for loosely coupled workloads and can quickly become complex if applied to tightly coupled workloads.

Reference Architecture

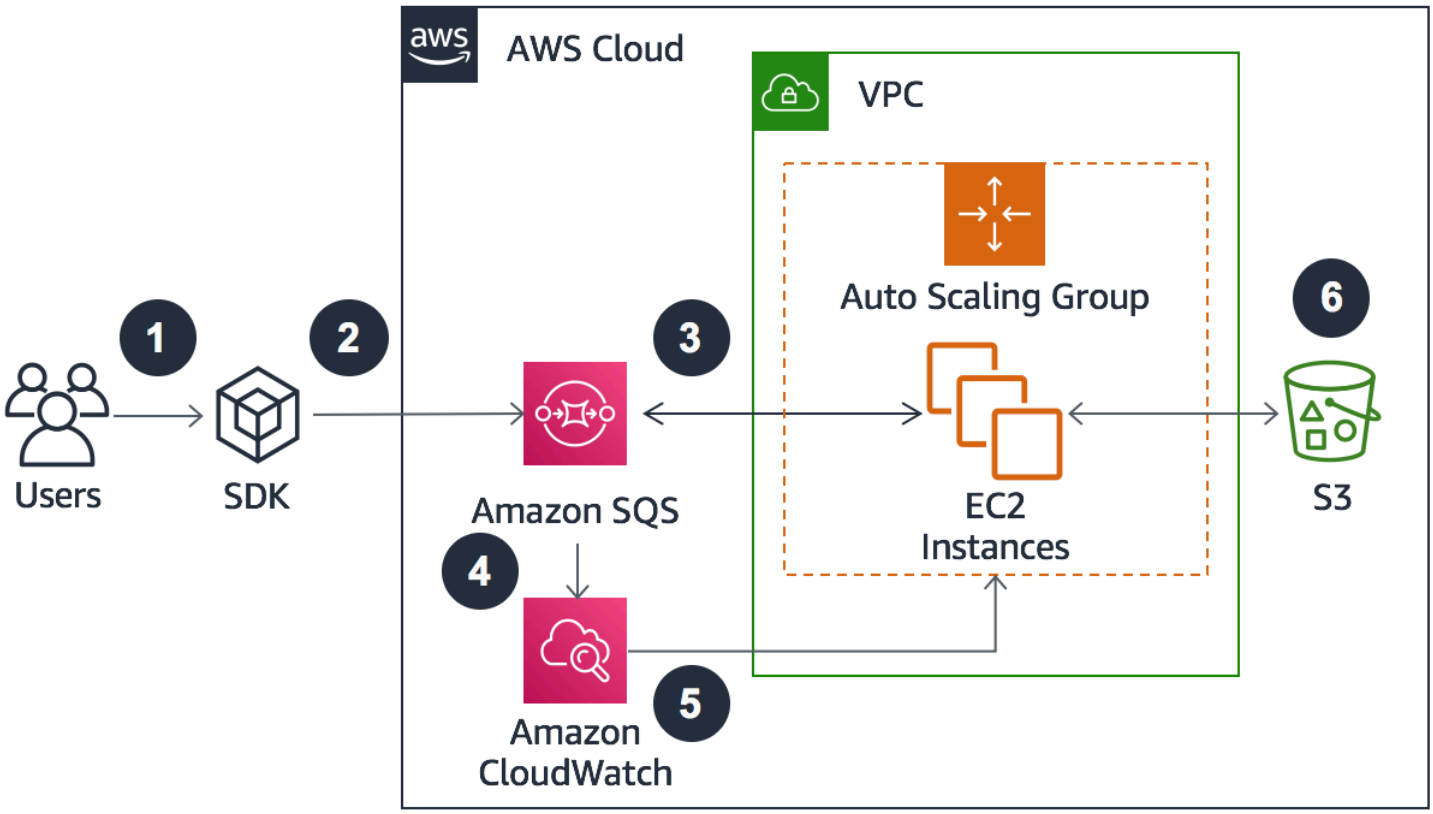

Figure 3: Amazon SQS deployed for a loosely coupled workload

Workflow steps:

1. Multiple users submit jobs with the AWS CLI or SDK.
2. The jobs are queued as messages in Amazon SQS.
3. EC2 Instances poll the queue and start processing jobs.
4. Amazon SQS emits metrics based on the number of messages (jobs) in the queue.
5. An Amazon CloudWatch alarm is configured to notify Auto Scaling if the queue is longer than a specified length. Auto Scaling increases the number of EC2 instances.
6. The EC2 instances pull source data and store result data in an S3 bucket.

Hybrid Deployment

Hybrid deployments are primarily considered by organizations that are invested in their on-premises infrastructure and also want to use AWS. This approach allows organizations to augment on-premises resources and creates an alternative path to AWS rather than an immediate full migration.

Hybrid scenarios vary from minimal coordination, like workload separation, to tightly integrated approaches, like scheduler driven job placement. For example, an organization may separate their workloads and run all workloads of a certain type on AWS infrastructure. Alternatively, organizations with a large investment in their on-premises processes and infrastructure may desire a more seamless experience for their end users by managing AWS resources with their job scheduling software and potentially a job submission portal. Several job schedulers – commercial and open-source – provide the capability to dynamically provision and deprovision AWS resources as necessary. The underlying resource management relies on native AWS integrations (for example, AWS CLI or API) and can allow for a highly customized environment, depending on the scheduler. Although job schedulers help manage AWS resources, the scheduler is only one aspect of a successful deployment.

Critical factors in successfully operating a hybrid scenario are data locality and data movement. Some HPC workloads do not require or generate significant datasets; therefore, data management is less of a concern. However, jobs that require large input data, or that generate significant output data, can become a bottleneck. Techniques to address data management vary depending on organization. For example, one organization may have their end users manage the data transfer in their job submission scripts, others might only run certain jobs in the location where a dataset resides, a third organization might choose to duplicate data in both locations, and yet another organization might choose to use a combination of several options.

Depending on the data management approach, AWS provides several services to aid in a hybrid deployment. For example, AWS Direct Connect establishes a dedicated network connection between an on-premises environment and AWS, and AWS DataSync automatically moves data from on-premises storage to Amazon S3 or Amazon Elastic File System. Additional software options are available from third-party companies in the AWS Marketplace and the AWS Partner Network (APN).

Hybrid-deployment architectures can be used for loosely and tightly coupled workloads. However, a single tightly coupled workload should reside either on-premises or in AWS for best performance.

Reference Architecture

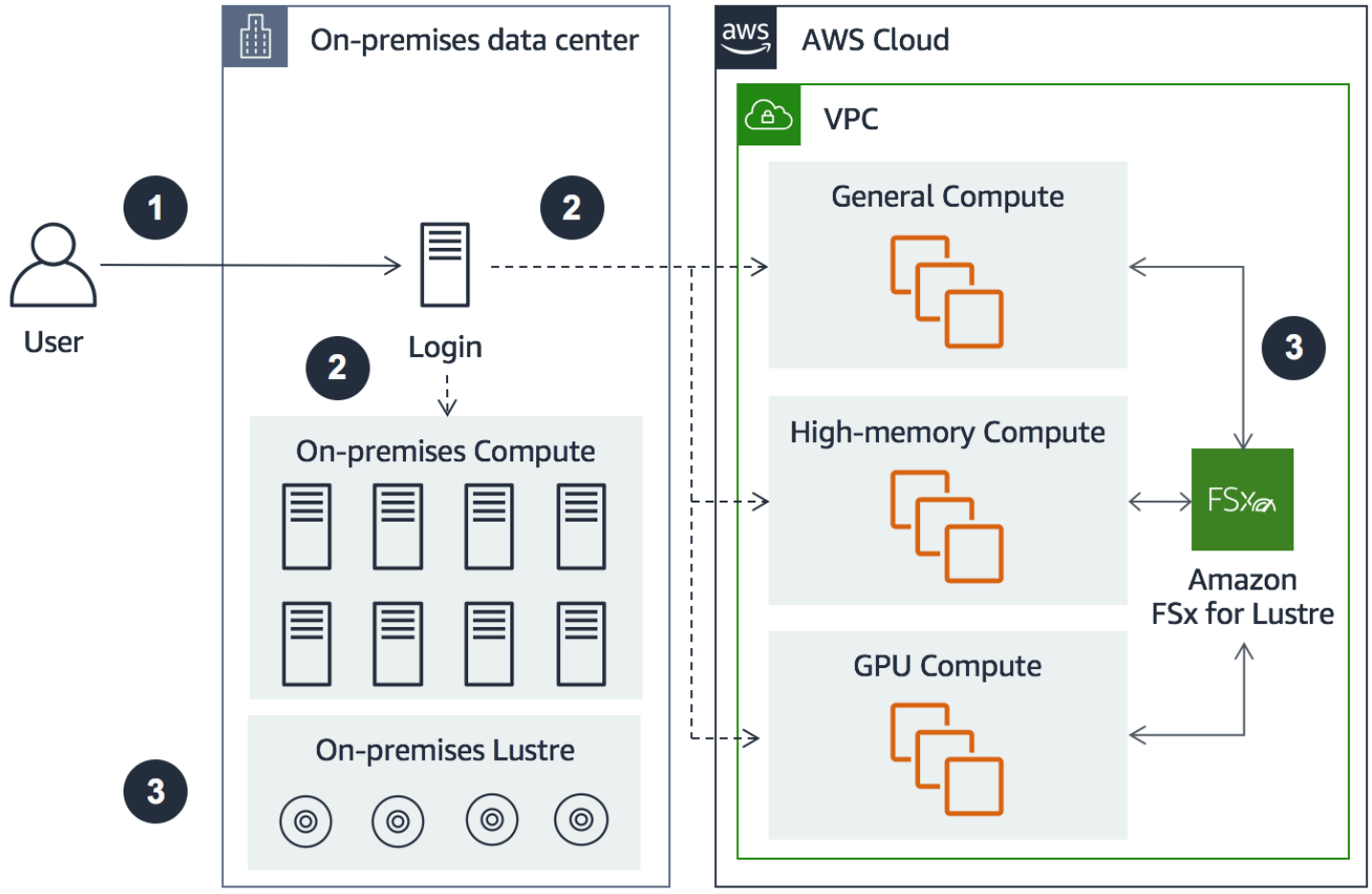

Figure 3: Example hybrid, scheduler-based deployment

Workflow steps:

1. User submits the job to a scheduler (for example, Slurm) on an on-premises login node.
2. Scheduler executes the job on either on-premises compute or AWS infrastructure based on configuration.
3. The jobs access shared storage based on their run location.

Serverless

The loosely coupled cloud journey often leads to an environment that is entirely serverless, meaning that you can concentrate on your applications and leave the server provisioning responsibility to managed services. AWS Lambda can run code without the need to provision or manage servers. You pay only for the compute time you consume — there is no charge when your code is not running. You upload your code, and Lambda takes care of everything required to run

and scale your code. Lambda also has the capability to automatically trigger events from other AWS services.

Scalability is a second advantage of the serverless Lambda approach. Although each worker may be modest in size – for example, a compute core with some memory – the architecture can spawn thousands of concurrent Lambda workers, thus reaching a large compute throughput capacity and earning the HPC label. For example, a large number of files can be analyzed by invocations of the same algorithm, a large number of genomes can be analyzed in parallel, or a large number of gene sites within a genome can be modeled. The largest attainable scale and speed of scaling matter. While server-based architectures require time on the order of minutes to increase capacity in response to a request (even when using virtual machines such as EC2 instances), serverless Lambda functions scale in seconds. AWS Lambda enables HPC infrastructure that responds immediately to any unforeseen requests for compute-intensive results, and can fulfill a variable number of requests without requiring any resources to be wastefully provisioned in advance.

In addition to compute, there are other serverless architectures that aid HPC workflows. AWS Step Functions let you coordinate multiple steps in a pipeline by stitching together different AWS services. For example, an automated genomics pipeline can be created with AWS Step Functions for coordination, Amazon S3 for storage, AWS Lambda for small tasks, and AWS Batch for data processing.

Serverless architectures are best for loosely coupled workloads, or as workflow coordination if combined with another HPC architecture.

Reference Architecture

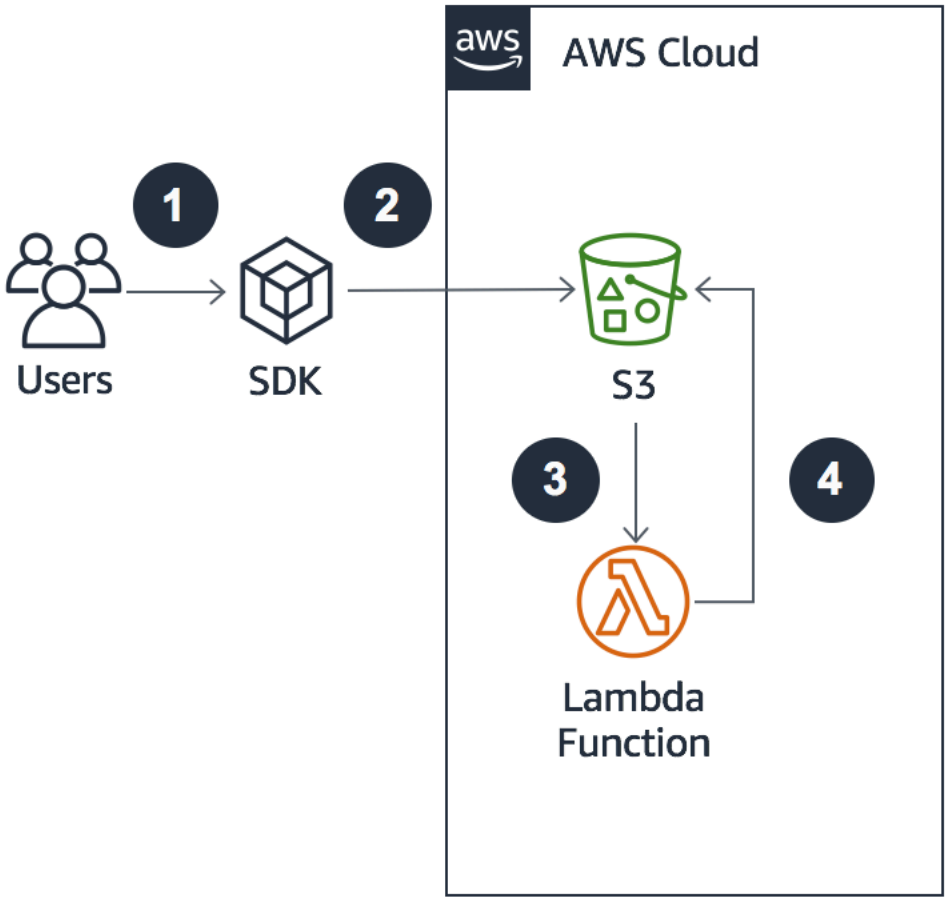

Figure 4: Example Lambda-deployed loosely coupled workload

Workflow steps:

1. The user uploads a file to an S3 bucket through the AWS CLI or SDK.
2. The input file is saved with an incoming prefix (for example, input/).
3. An S3 event automatically triggers a Lambda function to process the incoming data.
4. The output file is saved back to the S3 bucket with an outgoing prefix (for example, output/.)

The Pillars of the Well-Architected Framework

This section describes HPC in the context of the six pillars of the Well-Architected Framework. Each pillar discusses design principles, definitions, best practices, evaluation questions, considerations, key AWS services, and useful links.

Pillars

- Operational Excellence Pillar
- Security Pillar
- Reliability Pillar
- Performance Efficiency Pillar
- Cost Optimization Pillar
- Sustainability pillar

Operational Excellence Pillar

The **operational excellence** pillar includes the ability to run and monitor systems to deliver business value and continually improve supporting processes and procedures.

Topics

- Design Principles
- Best Practices

Design Principles

In the cloud, a number of principles drive operational excellence. In particular, the following are emphasized for HPC workloads. See also the design principles in the AWS Well-Architected Framework whitepaper.

- **Automate cluster operations**: In the cloud, you can define your entire workload as code and update it with code. This enables you to automate repetitive processes or procedures. You benefit from being able to consistently reproduce infrastructure and implement operational procedures. This includes automating the job submission process and responses to events, such as job start, completion, or failure. In HPC, it is common for users to expect to repeat multiple

steps for every job including, for example, uploading case files, submitting a job to a scheduler, and moving result files. Automate these repetitive steps with scripts or by event-driven code to maximize usability and minimize costs and failures.

- **Use cloud-native architectures where applicable**: HPC architectures typically take one of two forms. The first is a traditional cluster configuration with a login instance, compute nodes, and job scheduler. The second a cloud-native architecture with automated deployments and managed services. A single workload can run for each (ephemeral) cluster or use serverless capabilities. Cloud-native architectures can optimize operations with democratizing advanced technologies; however, the best technology approach aligns with the desired environment for HPC users.

Best Practices

There are three best practice areas for operational excellence in the cloud.

Topics

For more information on the **prepare, operate, and evolve** areas, see the AWS Well-Architected Framework whitepaper. Evolve is not described in this whitepaper.

Prepare

Review the corresponding section in the AWS Well-Architected Framework whitepaper.

As you prepare to deploy your workload, consider using specialized software packages (commercial or open source) to gain visibility into system information and leverage this information to define architecture patterns for your workloads. Use automation tools, such as AWS ParallelCluster or AWS CloudFormation, to define these architectures in a way that is configurable with variables.

The Cloud provides multiple scheduling options. One option is to use AWS Batch, which is a fully managed batch processing service with support for both single-node and multi-node tasks. Another option is to not use a scheduler. For example, you can create an ephemeral cluster to run a single job directly.

HPCOPS 1: How do you standardize architectures across clusters?

HPCOPS 2: How do you schedule jobs – traditional schedulers, AWS Batch, or no scheduler with ephemeral clusters?

Operate

Operations must be standardized and managed routinely. Focus on automation, small frequent changes, regular quality assurance testing, and defined mechanisms to track, audit, roll back, and review changes. Changes should not be large and infrequent, should not require scheduled downtime, and should not require manual execution. A wide range of logs and metrics based on key operational indicators for a workload must be collected and reviewed to ensure continuous operations.

AWS provides the opportunity to use additional tools for handling HPC operations. These tools can vary from monitoring assistance to automating deployments. For example, you can have Auto Scaling restart failed instances, use CloudWatch to monitor your cluster's load metrics, configure notifications for when jobs finish, or use a managed service (such as AWS Batch) to implement retry rules for failed jobs. Cloud-native tools can greatly improve your application deployment and change management.

Release management processes, whether manual or automated, must be based on small incremental changes and tracked versions. You must be able to revert releases that introduce issues without causing operational impact. Use continuous integration and continuous deployment tools such as AWS CodePipeline and AWS CodeDeploy to automate change deployment. Track source code changes with version control tools, such as AWS CodeCommit, and infrastructure configurations with automation tools, such as AWS CloudFormation templates.

HPCOPS 3: How are you evolving your workload while minimizing the impact of change?

HPCOPS 4: How do you monitor your workload to ensure that it is operating as expected?

Using the cloud for HPC introduces new operational considerations. While on-premises clusters are fixed in size, cloud clusters can scale to meet demand. Cloud-native architectures for HPC also operate differently than on-premises architectures. For example, they use different mechanisms for job submission and provisioning On-Demand Instance resources as jobs arrive. You must adopt operational procedures that accommodate the elasticity of the cloud and the dynamic nature of cloud-native architectures.

Evolve

There are no best practices unique to HPC for the **evolve** practice area. For more information, see the corresponding section in the AWS Well-Architected Framework whitepaper.

Security Pillar

The **security** pillar includes the ability to protect information, systems, and assets while delivering business value through risk assessments and mitigation strategies.

Topics

Design Principles

In the cloud, there are a number of principles that help you strengthen your system's security. The Design Principles from the AWS Well-Architected Framework whitepaper are recommended and do not vary for HPC workloads.

Definition

There are five best practice areas for security in the cloud:

- Identity and access management (IAM)
- Detective controls
- Infrastructure protection
- Data protection
- Incident response

Before architecting any system, you must establish security practices. You must be able to control permissions, identify security incidents, protect your systems and services, and maintain the confidentiality and integrity of data through data protection. You should have a well-defined and practiced process for responding to security incidents. These tools and techniques are important because they support objectives such as preventing data loss and complying with regulatory obligations.

The AWS Shared Responsibility Model enables organizations that adopt the cloud to achieve their security and compliance goals. Because AWS physically secures the infrastructure that supports our cloud services, you can focus on using services to accomplish your goals. The AWS Cloud provides access to security data, and an automated approach to responding to security events.

All of the security best practice areas are vital and well documented in the AWS Well-Architected Framework whitepaper. The **detective controls**, **infrastructure protection**, and **incident response** areas are described in the AWS Well-Architected Framework whitepaper. They are not described in this whitepaper and do not require modification for HPC workloads.

Best Practices

Topics

Identity and Access Management (IAM)

Identity and access management are key parts of an information security program. They ensure that only authorized and authenticated users are able to access your resources. For example, you define principals (users, groups, services, and roles that take action in your account), build out policies referencing these principals, and implement strong credential management. These privilege-management elements form the core concepts of authentication and authorization.

Run HPC workloads autonomously and ephemerally to limit the exposure of sensitive data. Autonomous deployments require minimal human access to instances, which minimizes the

exposure of the resources. HPC data is produced within a limited time, minimizing the possibility of potential unauthorized data access.

HPCSEC 1: How are you using managed services, autonomous methods, and ephemeral clusters to minimize human access to the workload infrastructure?

HPC architectures can use a variety of managed (for example, AWS Batch, AWS Lambda) and unmanaged compute services (for example, Amazon EC2). When architectures require direct access to the compute environments, such as connecting to an EC2 instance, users commonly connect through a Secure Shell (SSH) and authenticate with an SSH key. This access model is typical in a Traditional Cluster scenario. All credentials, including SSH keys, must be appropriately protected and regularly rotated.

Alternatively, AWS Systems Manager has a fully managed service (Session Manager) that provides an interactive browser-based shell and CLI experience. It provides secure and auditable instance management without the need to open inbound ports, maintain bastion hosts, and manage SSH keys. Session Manager can be accessed through any SSH client that supports ProxyCommand.

HPCSEC 2: What methods are you using to protect and manage your credentials?

Detective Controls

There are no best practices unique to HPC for the detective controls best practice area. Review the corresponding section in the AWS Well-Architected Framework whitepaper.

Infrastructure Protection

There are no best practices unique to HPC for the infrastructure best practice area. Review the corresponding section in the AWS Well-Architected Framework whitepaper.

Data Protection

Before architecting any system, you must establish foundational security practices. For example, *data classification* provides a way to categorize organizational data based on levels of sensitivity, and *encryption* protects data by rendering it unintelligible to unauthorized access. These tools

and techniques are important because they support objectives such as preventing data loss or complying with regulatory obligations.

HPCSEC 3: How does your architecture address data requirements for storage availability and durability through the lifecycle of your results?

In addition to the level of sensitivity and regulatory obligations, HPC data can also be categorized according to when and how the data will next be used. Final results are often retained while intermediate results, which can be recreated if necessary, may not need to be retained. Careful evaluation and categorization of data allows for programmatic data migration of important data to more resilient storage solutions, such as Amazon S3 and Amazon EFS.

An understanding of data longevity combined with programmatic handling of the data offers the minimum exposure and maximum protection for a Well-Architected infrastructure.

Incident Response

There are no best practices unique to HPC for the incident response best practice area. Review the corresponding section in the AWS Well-Architected Framework whitepaper.

Reliability Pillar

The **reliability** pillar includes the ability of a system to recover from infrastructure or service disruptions, dynamically acquire computing resources to meet demand, and mitigate disruptions such as misconfigurations or transient network issues.

Topics

- Design Principles
- Definition
- Best Practices

Design Principles

In the cloud, a number of principles help you increase reliability. In particular, the following are emphasized for HPC workloads. For more information, refer to the design principles in the AWS Well-Architected Framework whitepaper.

- **Scale horizontally to increase aggregate system availability**: It is important to consider horizontal scaling options that might reduce the impact of a single failure on the overall system. For example, rather than having one large, shared HPC cluster running multiple jobs, consider creating multiple clusters across the Amazon infrastructure to further isolate your risk of potential failures. Since infrastructure can be treated as code, you can horizontally scale resources inside a single cluster, and you can horizontally scale the number of clusters running individual cases.
- **Stop guessing capacity**: A set of HPC clusters can be provisioned to meet current needs and scaled manually or automatically to meet increases or decreases in demand. For example, terminate idle compute nodes when not in use, and run concurrent clusters for processing multiple computations rather than waiting in a queue.
- **Manage change in automation**: Automating changes to your infrastructure allows you to place a cluster infrastructure under version control and make exact duplicates of a previously created cluster. Automation changes must be managed.

Definition

There are three best practice areas for reliability in the cloud:

- Foundations
- Change management
- Failure management

The change management area is described in the AWS Well-Architected Framework whitepaper.

Best Practices

Topics

- Foundations
- Change Management
- Failure Management

Foundations

HPCREL 1: How do you manage AWS service limits for your accounts?

AWS sets service limits (an upper limit on the number of each resource your team can request) to protect you from accidentally over-provisioning resources.

HPC applications often require a large number of compute instances simultaneously. The ability and advantages of scaling horizontally are highly desirable for HPC workloads. However, scaling horizontally may require an increase to the AWS service limits before a large workload is deployed to either one large cluster, or to many smaller clusters all at once.

Service limits must often be increased from the default values in order to handle the requirements of a large deployment. Contact AWS Support to request an increase.

Change Management

There are no best practices unique to HPC for the change management best practice area. Review the corresponding section in the AWS Well-Architected Framework whitepaper.

Failure Management

Any complex system can expect for failures to occasionally occur, and it critical to become aware of these failures, respond to them, and prevent them from happening again. Failure scenarios can include the failure of a cluster to start up, or the failure of a specific workload.

HPCREL 2: How does your application use checkpointing to recover from failures?

Failure tolerance can be improved in multiple ways. For long-running cases, incorporating regular checkpoints in your code allows you to continue from a partial state in the event of a failure. Checkpointing is a common feature of application-level failure management already built into many HPC applications. The most common approach is for applications to periodically write out intermediate results. The intermediate results offer potential insight into application errors and the ability to restart the case as needed while only partially losing the work.

Checkpointing is useful on Spot Instances when you are using highly cost-effective, but potentially interruptible instances. In addition, some applications may benefit from changing the default Spot interruption behavior (for example, stopping or hibernating the instance rather than terminating it). It is important to consider the durability of the storage option when relying on checkpointing for failure management.

HPCREL 3: How have you planned for failure tolerance in your architecture?

Failure tolerance can be improved when deploying to multiple Availability Zones. The low-latency requirements of tightly coupled HPC applications require that each individual case reside within a single cluster placement group and Availability Zone. Alternatively, loosely coupled applications do not have such low-latency requirements and can improve failure management with the ability to deploy to several Availability Zones.

Consider the tradeoff between the reliability and cost pillars when making this design decision. Duplication of compute and storage infrastructure (for example, a head node and attached storage) incurs additional cost, and there may be data transfer charges for moving data to an Availability Zone or to another AWS Region. For non-urgent use cases, it may be preferable to only move into another Availability Zone as part of a disaster recovery (DR) event.

Performance Efficiency Pillar

The **performance efficiency** pillar focuses on the efficient use of computing resources to meet requirements and on maintaining that efficiency as demand changes and technologies evolve.

Topics

Design Principles

When designing for HPC in the cloud, a number of principles help you achieve performance efficiency:

- **Design the cluster for the application**: Traditional clusters are static and require that the application be designed for the cluster. AWS offers the capability to design the cluster for the application. A one-size-fits-all model is no longer necessary with individual clusters for each application. When running a variety of applications on AWS, a variety of architectures can be used to meet each application's demands. This allows for the best performance while minimizing cost.
- **Test performance with a meaningful use case**: The best method to gauge an HPC application's performance on a particular architecture is to run a meaningful demonstration of the application itself. An inadvertently small or large demonstration case – one without the expected compute, memory, data transfer, or network traffic –will not provide a meaningful test of application performance on AWS. Although system-specific benchmarks offer an understanding of the underlying compute infrastructure performance, they do not reflect how an application will perform in the aggregate. The AWS pay-as-you-go model makes a proof-of-concept quick and cost-effective.
- **Use *cloud-native* architectures where applicable:** In the cloud, managed, serverless, and cloud-native architectures remove the need for you to run and maintain servers to carry out traditional compute activities. Cloud**-native components for HPC target compute, storage, job orchestration and organization of the data and metadata**. The variety of AWS services allows each step in the workload process to be decoupled and optimized for a more performant capability.
- **Experiment often:** Virtual and automatable resources allow you to quickly carry out comparative testing using different types of instances, storage, and configurations.

Definition

There are four best practice areas for performance efficiency in the cloud:

- Selection
- Review
- Monitoring
- Tradeoffs

The **review**, **monitoring**, and **tradeoffs** areas are described in the AWS Well-Architected Framework whitepaper.

Best Practices

Topics

Selection

The optimal solution for a particular system varies based on the kind of workload you have. Well-Architected systems use multiple solutions and enable different features to improve performance. An HPC architecture can rely on one or more different architectural elements, for example, queued, batch, cluster compute, containers, serverless, and event-driven.

Compute

HPCPERF 1: How do you select your compute solution?

The optimal compute solution for a particular HPC architecture depends on the workload deployment method, degree of automation, usage patterns, and configuration. Different compute solutions may be chosen for each step of a process. Selecting the wrong compute solutions for an architecture can lead to lower performance efficiency.

Instances are virtualized servers and come in different families and sizes to offer a wide variety of capabilities. Some instance families target specific workloads, for example, compute-, memory-, or GPU-intensive workloads. Other instances are general purpose.

Both the targeted-workload and general-purpose instance families are useful for HPC applications. Instances of particular interest to HPC include the compute-optimized family and accelerated instance types such as GPUs and FPGAs.

Some instance families provide variants within the family for additional capabilities. For example, an instance family may have a variant with local storage, greater networking capabilities, or a different processor. These variants can be viewed in the Instance Type Matrix and may improve the performance of some HPC workloads.

Within each instance family, one or more instance sizes allow vertical scaling of resources. Some applications require a larger instance type (for example, 24xlarge) while others run on smaller types (for example, large) depending on the number or processes supported by the application. The optimum performance is obtained with the largest instance type when working with a tightly coupled workload.

The T-series instance family is designed for applications with moderate CPU usage that can benefit from bursting beyond a baseline level of CPU performance. Most HPC applications are compute-intensive and suffer a performance decline with the T-series instance family.

Applications vary in their requirements (for example, desired cores, processor speed, memory requirements, storage needs, and networking specifications). When selecting an instance family and type, begin with the specific needs of the application. Instance types can be mixed and matched for applications requiring targeted instances for specific application components.

Containers are a method of operating system virtualization that is attractive for many HPC workloads, particularly if the applications have already been containerized. AWS services such as AWS Batch, Amazon Elastic Container Service (ECS), and Amazon Elastic Container Service for Kubernetes (EKS) help deploy containerized applications.

Functions abstract the execution environment. AWS Lambda allows you to execute code without deploying, running, or maintaining, an instance. Many AWS services emit events based on activity inside the service, and often a Lambda function can be triggered off of service events. For example, a Lambda function can be executed after an object is uploaded to Amazon S3. Many HPC users use Lambda to automatically execute code as part of their workflow.

There are several choices to make when launching your selected compute instance:

- **Operating system**: A current operating system is critical to achieving the best performance and ensuring access to the most up-to-date libraries.
- **Virtualization type**: New-generation EC2 instances run on the AWS Nitro System. The Nitro System delivers all the host hardware's compute and memory resources to your instances, resulting in better overall performance. Dedicated Nitro Cards enable high-speed networking,

high-speed EBS, and I/O acceleration. Instances do not hold back resources for management software.

The Nitro Hypervisor is a lightweight hypervisor that manages memory and CPU allocation and delivers performance that is indistinguishable from bare metal. The Nitro System also makes bare metal instances available to run without the Nitro Hypervisor. Launching a bare metal instance boots the underlying server, which includes verifying all hardware and firmware components. This means it can take longer before the bare metal instance becomes available to start your workload, as compared to a virtualized instance. The additional initialization time must be considered when operating in a dynamic HPC environment where resources launch and terminate based on demand.

HPCPERF 2: How do you optimize the compute environment for your application?

Underlying hardware features: In addition to choosing an AMI, you can further optimize your environment by taking advantage of the hardware features of the underlying Intel processors. There are four primary methods to consider when optimizing the underlying hardware:

1. Advanced processor features
2. Intel Hyper-Threading Technology
3. Processor affinity
4. Processor state control

HPC applications can benefit from these advanced processor features (for example, Advanced Vector Extensions) and can increase their calculation speeds by compiling the software for the Intel architecture. The compiler options for architecture-specific instructions vary by compiler (check the usage guide for your compiler).

AWS enables Intel Hyper-Threading Technology, commonly referred to as “hyperthreading,” by default. Hyperthreading improves performance for some applications by allowing one process per hyperthread (two processes per core). Most HPC applications benefit from disabling hyperthreading, and therefore, it tends to be the preferred environment for HPC applications. Hyperthreading is easily disabled in Amazon EC2. Unless an application has been tested with hyperthreading enabled, it is recommended that hyperthreading be disabled and that processes

are launched and individually pinned to cores when running HPC applications. CPU or processor affinity allows process pinning to easily happen.

Processor affinity can be controlled in a variety of ways. For example, it can be configured at the operating system level (available in both Windows and Linux), set as a compiler flag within the threading library, or specified as an MPI flag during execution. The chosen method of controlling processor affinity depends on your workload and application.

AWS enables you to tune the processor state control on certain instance types. You may consider altering the C-state (idle states) and P-state (operational states) settings to optimize your performance. The default C-state and P-state settings provide maximum performance, which is optimal for most workloads. However, if your application would benefit from reduced latency at the cost of higher single- or dual-core frequencies, or from consistent performance at lower frequencies as opposed to spiky Turbo Boost frequencies, experiment with the C-state or P-state settings available on select instances.

There are many compute options available to optimize a compute environment. Cloud deployment allows experimentation on every level from operating system to instance type, to bare-metal deployments. Because static clusters are tuned before deployment, time spent experimenting with cloud-based clusters is vital to achieving the desired performance.

Storage

HPCPERF 3: How do you select your storage solution?

The optimal storage solution for a particular HPC architecture depends largely on the individual applications targeted for that architecture. Workload deployment method, degree of automation, and desired data lifecycle patterns are also factors. **AWS offers a wide range of storage options. As with compute, the best performance is obtained when targeting the specific storage needs of an application. AWS does not require you to over-provision your storage for a "one-size-fits-all" approach, and** large, high-speed, shared file systems are not always required. **Optimizing the compute choice is important for optimizing HPC performance and many HPC applications will not benefit from the fastest storage solution possible.**

HPC deployments often require a shared or high-performance file system that is accessed by the cluster compute nodes. There are several architecture patterns you can use to implement these storage solutions from AWS Managed Services, AWS Marketplace offerings, APN Partner solutions,

and open-source configurations deployed on EC2 instances. In particular, Amazon FSx for Lustre is a managed service that provides a cost effective and performant solution for HPC architectures requiring a high-performance parallel file system. Shared file systems can also be created from Amazon Elastic File System (EFS) or EC2 instances with Amazon EBS volumes or instance store volumes. Frequently, a simple NFS mount is used to create a shared directory.

When selecting your storage solution, you may select an EBS-backed instance for either or both of your local and shared storages. EBS volumes are often the basis for an HPC storage solution. Various types of EBS volumes are available including magnetic hard disk drives (HDDs), general-purpose solid state drives (SSDs), and Provisioned IOPS SSDs for high IOPS solutions. They differ in throughput, IOPS performance, and cost.

You can gain further performance enhancements by selecting an Amazon EBS-optimized instance. An EBS-optimized instance uses an optimized configuration stack and provides additional, dedicated capacity for Amazon EBS I/O. This optimization provides the best performance for your EBS volumes by minimizing contention between Amazon EBS I/O and other network traffic to and from your instance. Choose an EBS-optimized instance for more consistent performance and for HPC applications that rely on a low-latency network or have intensive I/O data needs to EBS volumes.

To launch an EBS-optimized instance, choose an instance type that enables EBS optimization by default, or choose an instance type that allows enabling EBS optimization at launch.

Instance store volumes, including nonvolatile memory express (NVMe) SSD volumes (only available on certain instance families), can be used for temporary block-level storage. Refer to the instance type matrix for EBS optimization and instance-store volume support.

When you select a storage solution, ensure that it aligns with your access patterns to achieve the desired performance. It is easy to experiment with different storage types and configurations. With HPC workloads, the most expensive option is not always the best performing solution.

Networking

HPCPERF 4: How do you select your network solution?

The optimal network solution for an HPC workload varies based on latency, bandwidth, and throughput requirements. Tightly coupled HPC applications often require the lowest latency possible for network connections between compute nodes. For moderately sized, tightly coupled

workloads, it is possible to select a large instance type with a large number of cores so that the application fits entirely within the instance without crossing the network at all.

Alternatively, some applications are network bound and require high network performance. Instances with higher network performance can be selected for these applications. The highest network performance is obtained with the largest instance type in a family. Refer to the instance type matrix for more details.[7]

Multiple instances with low latency between the instances are required for large tightly coupled applications. On AWS, this is achieved by launching compute nodes into a cluster placement group, which is a logical grouping of instances within an Availability Zone. A cluster placement group provides non-blocking and non-oversubscribed connectivity, including full bisection bandwidth between instances. Use cluster placement groups for latency sensitive tightly coupled applications spanning multiple instances.

In addition to cluster placement groups, tightly coupled applications benefit from an Elastic Fabric Adapter (EFA), a network device that can attach to your Amazon EC2 instance. EFA provides lower and more consistent latency and higher throughput than the TCP transport traditionally used in cloud-based HPC systems. It enables an OS-bypass access model through the *Libfabric* API that allows HPC applications to communicate directly with the network interface hardware. EFA enhances the performance of interinstance communication, is optimized to work on the existing AWS network infrastructure, and is critical for scaling tightly coupled applications.[13]

If an application cannot take advantage of EFA's OS-bypass functionality, or an instance type does not support EFA, optimal network performance can be obtained by selecting an instance type that supports enhanced networking. Enhanced networking provides EC2 instances with higher networking performance and lower CPU utilization through the use of pass-through rather than hardware-emulated devices. This method allows EC2 instances to achieve higher bandwidth, higher packet-per-second processing, and lower interinstance latency compared to traditional device virtualization.

Enhanced networking is available on all current-generation instance types and requires an AMI with supported drivers. Although most current AMIs contain supported drivers, custom AMIs may require updated drivers. For more information on enabling enhanced networking and instance support, refer to the enhanced networking documentation.

Loosely coupled workloads are generally not sensitive to very low-latency networking and do not require the use of a cluster placement group or the need to keep instances in the same Availability Zone or Region.

Review

There are no best practices unique to HPC for the review best practice area. Review the corresponding section in the AWS Well-Architected Framework whitepaper.

Monitoring

There are no best practices unique to HPC for the monitoring best practice area. Review the corresponding section in the AWS Well-Architected Framework whitepaper.

Trade-offs

There are no best practices unique to HPC for the monitoring best practice area. Review the corresponding section in the AWS Well-Architected Framework whitepaper.

Cost Optimization Pillar

The **cost optimization** pillar includes the continual process of refinement and improvement of an HPC system over its entire lifecycle. From the initial design of your first proof of concept, to the ongoing operation of production workloads, adopting the practices in this paper enables you to build and operate cost-aware systems that achieve business outcomes and minimize costs. This allows your business to maximize its return on investment.

Topics

Design Principles

For HPC in the cloud you can follow a number of principles to achieve cost optimization:

- **Adopt a consumption model**: Pay only for the computing resources that you consume. HPC workloads ebb and flow, providing the opportunity to reduce costs by increasing and decreasing resource capacity on an as-needed basis. For example, a low-level run-rate HPC capacity can be provisioned and reserved upfront so as to benefit from higher discounts, while burst requirements may be provisioned with spot or on-demand pricing and brought online, only as needed.

- **Optimize infrastructure costs for specific jobs:** Many HPC workloads are part of a data processing pipeline that includes the data transfer, pre-processing, computational calculations, post-processing, data transfer, and storage steps. In the cloud, rather than on a large and expensive server, the computing platform is optimized at each step. For example, if a single step in a pipeline requires a large amount of memory, you only need to pay for a more expensive large-memory server for the memory-intensive application, while all other steps can run well on smaller and less expensive computing platforms. Costs are reduced by optimizing infrastructure for each step of a workload.
- **Burst workloads in the most efficient way:** Savings are obtained for HPC workloads through horizontal scaling in the cloud. When scaling horizontally, many jobs or iterations of an entire workload are run simultaneously for less total elapsed time. Depending on the application, horizontal scaling can be cost neutral while offering indirect cost savings by delivering results in a fraction of the time.
- **Make use of spot pricing: Amazon EC2 Spot Instances offer spare compute capacity in AWS at steep discounts compared to On-Demand instances. However, Spot Instances can be interrupted when EC2 needs to reclaim the capacity.** Spot Instances are frequently the most cost-effective resource for flexible or fault-tolerant workloads. The intermittent nature of HPC workloads makes them well suited to Spot Instances. The risk of Spot Instance interruption can be minimized by working with the Spot Advisor, and the interruption impact can be mitigated by changing the default interruption behavior and using Spot Fleet to manage your Spot Instances. The need to occasionally restart a workload is easily offset by the cost savings of Spot Instances.
- **Assess the tradeoff of cost versus time**: Tightly coupled, massively parallel workloads are able to run on a wide range of core counts. For these applications, the run efficiency of a case typically falls off at higher core counts. A cost versus turnaround-time curve can be created if many cases of similar type and size will run. Curves are specific to both the case type and application as scaling depends significantly on the ratio of computational to network requirements. Larger workloads are capable of scaling further than smaller workloads. With an understanding of the cost versus turnaround-time tradeoff, time-sensitive workloads can run more quickly on more cores, while cost savings can be achieved by running on fewer cores and at maximum efficiency. Workloads can fall somewhere in between when you want to balance time sensitivity and cost sensitivity.

Definition

There are four best practice areas for cost optimization in the cloud:

- Cost-effective resources
- Matching supply and demand
- Expenditure awareness
- Optimizing over time

The **matching supply and demand**, **expenditure awareness**, and **optimizing over time** areas are described in the AWS Well-Architected Framework whitepaper.

Best Practices

Topics

Cost-Effective Resources

HPCCOST 1: How have you evaluated available compute and storage options for your workload to optimize cost?

HPCCOST 2: How have you evaluated the trade-offs between job completion time and cost?

Using the appropriate instances, resources, and features for your system is key to cost management. The instance choice may increase or decrease the overall cost of running an HPC workload. For example, a tightly coupled HPC workload might take five hours to run on a cluster of several smaller servers, while a cluster of fewer and larger servers may cost double per hour but compute the result in one hour, saving money overall. The choice of storage can also impact cost. Consider the potential tradeoff between job turnaround and cost optimization and test workloads with different instance sizes and storage options to optimize cost.

AWS offers a variety of flexible and cost-effective pricing options to acquire instances from EC2 and other services in a way that best fits your needs. On-Demand Instances allow you to pay for compute capacity by the hour, with no minimum commitments required. Reserved Instances allow you to reserve capacity and offer savings relative to On-Demand pricing. With Spot Instances, you can leverage unused Amazon EC2 capacity and offer additional savings relative to On-Demand pricing.

A Well-Architected system uses the most cost-effective resources. You can also reduce costs by using managed services for pre-processing and post-processing. For example, rather than maintaining servers to store and post-process completed run data, data can be stored on Amazon S3 and then post-processed with Amazon EMR or AWS Batch.

Many AWS services provide features that further reduce your costs. For example, Auto Scaling is integrated with EC2 to automatically launch and terminate instances based on workload demand. FSx for Lustre natively integrates with S3 and presents the entire contents of an S3 bucket as a Lustre filesystem. This allows you to optimize your storage costs by provisioning a minimal Lustre filesystem for your immediate workload while maintaining your long-term data in cost-effective S3 storage. S3 provides different Storage Classes so that you can use the most cost-effective class for your data; Glacier or Glacier Deep Storage Classes enable you to archive data at the lowest cost.

Experimenting with different instance types, storage requirements, and architectures can minimize costs while maintaining desirable performance.

Matching Supply and Demand

There are no best practices unique to HPC for the matching supply and demand best practice area. Review the corresponding section in the AWS Well-Architected Framework whitepaper.

Expenditure Awareness

There are no best practices unique to HPC for the expenditure awareness best practice area. Review the corresponding section in the AWS Well-Architected Framework whitepaper.

Optimizing Over Time

There are no best practices unique to HPC for the optimizing over time best practice area. Review the corresponding section in the AWS Well-Architected Framework whitepaper.

Sustainability pillar

The sustainability pillar includes the ability to continually improve sustainability impacts by reducing energy consumption and increasing efficiency across all components of a workload by maximizing the benefits from the provisioned resources and minimizing the total resources required.

There are no sustainability practices unique to this lens. For information on Sustainability, refer to the Sustainability Pillar whitepaper.

Conclusion

This lens provides architectural best practices for designing and operating reliable, secure, efficient, and cost-effective systems for High Performance Computing workloads in the cloud. We covered prototypical HPC architectures and overarching HPC design principles. We revisited the Well-Architected pillars through the lens of HPC, providing you with a set of questions to help you review an existing or proposed HPC architecture. Applying the Framework to your architecture helps you build stable and efficient systems, allowing you to focus on running HPC applications and pushing the boundaries of your field.

Contributors

The following individuals and organizations contributed to this document:

- Aaron Bucher, HPC Specialist Solutions Architect, Amazon Web Services
- Omar Shorbaji, Global Solutions Architect, Amazon Web Services
- Linda Hedges, HPC Application Engineer, Amazon Web Services
- Nina Vogl, HPC Specialist Solutions Architect, Amazon Web Services
- Sean Smith, HPC Software Development Engineer, Amazon Web Services
- Kevin Jorissen, Solutions Architect – Climate and Weather, Amazon Web Services
- Philip Fitzsimons, Sr. Manager Well-Architected, Amazon Web Services

Further Reading

For additional information, see the following:

- AWS Well-Architected Framework
- High Performance Computing on AWS
- AWS Architecture Center
- High Performance Computing on AWS Redefines What is Possible
- Optimizing Electronic Design Automation (EDA) Workflows on AWS
- Real World AWS Scalability

Document Revisions

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

Change	Description	Date
Minor update	Fixed broken links.	April 6, 2023
Minor update	Updated text to reflect the Sustainability Pillar.	October 3, 2022
Minor update	Updated links.	March 10, 2021
Whitepaper updated	Minor updates	December 19, 2019
Whitepaper updated	Minor updates	November 1, 2018
Initial publication	High Performance Computing Lens first published.	November 1, 2017

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.