High Performance Computing Lens

AWS Well-Architected Framework

November 2018

Notices

Contents

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 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.[1] By using the Framework you will learn architectural best practices for designing and operating reliable, secure, efficient, and cost-effective systems in the cloud. It 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** on the AWS Cloud. HPC workloads run exceptionally well in the cloud. The natural ebb and flow and bursting characteristic of HPC workloads make them particularly well suited for pay-as-you-go cloud infrastructure. The ability to fine tune cloud resources and create cloud-native architectures accelerates the turnaround of HPC workloads naturally.

For brevity, we have only covered details from the Well-Architected Framework that are specific to HPC workloads. You should still consider best practices and questions that haven’t been included in this document when designing your architecture. We recommend that you read the AWS Well-Architected Framework whitepaper.[2]

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 five pillars: operational excellence, security, reliability, performance efficiency, and cost optimization. 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 (or high-throughput) and tightly coupled (or high-performance) workloads. We will 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 generally uses the term "vCPU" somewhat synonymously with a "thread" or a hyperthread (or, if you will, *half* of a physical core). Don't miss this factor of 2 when quantifying the performance or cost of an HPC application on AWS.

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 but simply a logical rule keeping all nodes within a low latency radius of the network. This is a crucial requirement of a tightly-coupled HPC architecture. Placement groups are not recommended for loosely coupled applications, where they would just introduce an unnecessary constraint.

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 (decreasing 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 should 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).
- **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 will 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. Hence, they are a great match for Amazon EC2 Spot Instances. Spot Instances are often available at a discount compared to On-Demand pricing.
    - 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 while ensuring 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 the needed capacity is available every day at the right time. Use of this pricing model also provides a discount compared with On-Demand Instances.

- **Start from the data**: Before you start designing the architecture, you should have a clear picture of where the data comes from, how large it is, how fast it needs to travel, how frequently data sources are updated, and where the data needs to be stored. Similarly, a holistic optimization of performance and cost shouldn't focus exclusively on compute cores and Amazon Elastic Compute Cloud (Amazon EC2) performance. AWS has a strong offering of data services that are not part of the traditional HPC landscape, but they can enable 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 the impact, and revert to previous parameters when necessary. The ability to easily experiment with infrastructure allows you to optimize the architecture for performance and cost.
- **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 Marketplace products, and scripts. Take full advantage of AWS security and collaboration features that make AWS an excellent environment for you and your collaborators to

solve your HPC problems. This will help you achieve greater impact for your computing solutions and datasets.

- **Cloud-native designs**: It might not be necessary to replicate your on-premises environment when you move workloads to AWS. It is often suboptimal to do so. 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 AWS Lambda computing, to manage the underlying compute. Consider not using a traditional cluster scheduler. Instead, use a scheduler only if your workload requires it. In the cloud, clusters no longer require permanence. After you automate cluster deployment, you can tear one cluster down and launch a new one on demand, with different parameters.
- **Test real-world workloads**: The only way to know how your production workload will perform on 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. 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." Since you will only pay for the hours of compute and storage you actually use, it's quite feasible to do a realistic proof-of-concept on AWS. This is a major advantage of a cloud-based platform: a realistic, full-scale test can be done before migration.
- **Balance time-to-results and cost reduction**: Analyze performance using the most meaningful parameters: how long it will take, and how much it will cost. Cost optimization should be used for workloads that are not time-sensitive. On AWS, Spot Instances are frequently the least expensive method for such workloads. For example, if a researcher has a large number of lab measurements that need to be analyzed sometime before next year's conference, Spot Instances can help her analyze the largest possible number of measurements within her fixed research budget. For urgent 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 the lowest execution time. If comparing between platforms, it's important to take

the entire time-to-solution into account, including non-compute aspects such as provisioning resources, staging data, or 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 is used in a wide range of areas, from Bioscience to Geoscience, Manufacturing, Electronic Design Automation, Climate Simulation, Finance, Media and Entertainment, and so on. Still, the grids or HPC clusters that support these calculations are remarkably similar to each other, with select cluster attributes optimized for the specific workload. In the cloud, the network, storage type, compute (instance) type, and even deployment method can be strategically chosen to optimize performance, cost, and usability for a particular use case or application.

HPC is generally divided into two categories based on the degree of interaction between the concurrently running parallel processes. 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. We refer to these loosely coupled calculations as high-throughput computing or HTC. Conversely, 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. Tightly coupled cases are often simply referred to as HPC cases.

With HTC, each parallel process is called an “iteration” of one simulation. The completion of an entire calculation or simulation often requires hundreds to millions of iterations (or more). The iterations typically occur in any order and at any speed, along the way to completion of the entire simulation. This offers flexibility on the computing infrastructure required for these simulations.

Unlike with HTC, the HPC processes are “tightly coupled,” meaning that information is exchanged regularly between the processes at each iteration or step of the simulation. Typically, the simulation runs on a homogenous cluster, with each process pinned to one core. The total core or processor count can range from tens, to hundreds, to thousands, and occasionally to hundreds of thousands if the infrastructure allows. The interactions of the processes during the simulation places extra demands on the computing infrastructure, such as the need for high-performance compute nodes and network infrastructure.

The infrastructure that is used to run the huge variety of HTC and HPC applications is thus differentiated by the degree of support for coupling of processes across nodes. We present two separate categories of example architectures for HTC and tightly-coupled HPC workloads in this whitepaper. Each workload is presented generically, though callouts are made for variations in scenarios.

Traditional, on-premises clusters force a one-size-fits-all approach to the HPC/HTC 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 a master node, to a cloud-native architecture with the advantages of cost efficiencies obtainable with cloud-native solutions.

Consider the following fundamentals when selecting an HPC infrastructure on AWS:

- **Network**: Network requirements can range from cases with low requirements, such as HTC 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 at each step include data size, media type, transfer speeds, shared access, and storage properties (for example, durability and availability).
- **Compute**: The AWS instance type defines the processor type, speed, accelerator options, and the memory-to-core ratio. Each instance type has access to different network options as well. On AWS, an instance is considered to be the same as an HPC node. These terms will be used interchangeably in this whitepaper.

  AWS also offers managed services with the ability to access compute without the need to choose the underlying EC2 instance type. AWS

Lambda, AWS Batch, and AWS Fargate are examples of compute services that pick the underlying instance type based on the requested vCPU and memory requirements.

- **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) are available for coding end-to-end solutions. (Available SDKs include Python, Ruby, Java, and many others.) A popular HPC deployment option combines bash shell scripting with the AWS Command Line Interface (AWS CLI). In addition, AWS CloudFormation templates allow the specification of application-tailored HPC clusters as code so that they can be launched in minutes.

  AWS also provides managed services for container-based workloads, such as Amazon EC2 Container Service (Amazon ECS), AWS EKS, AWS Fargate, and AWS Batch. In addition, AWS offers CfnCluster which is open source software that coordinates the launch of a cluster with already installed software (for example, compilers and schedulers) for a traditional cluster experience. Lastly, 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, High-Throughput Computing

An HTC problem entails the processing of a large number of smaller jobs. Each small job is called an “iteration.” Generally, an HTC iteration is run on one node, either consuming one process or the entire multiprocessor node 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. HTC applications are found in many areas, including Monte Carlo simulations, image processing, and genomics analysis.

With HTC, the loss of one node or job doesn't delay the entire calculation. The lost work can be picked up later or, often, even omitted altogether. The nodes involved in the calculation can vary in specification and power.

A suitable architecture for an HTC workload has the following considerations:

- **Network**: Because parallel processes in HTC do not interact with each other, the feasibility or performance of the workloads is not sensitive to the bandwidth and latency capabilities of the network. Therefore, network requirements for HTC workloads are minimal. You should not use placement groups for this scenario because they would provide no performance gain (due to little to no communication between nodes) and potentially weaken resiliency.
- **Storage**: HTC workloads have varying storage requirements. Storage requirements are usually driven by the dataset size and the performance requirements for moving, reading, and writing data. Occasionally, it's helpful to use a shared file system (for example, EFS or NFS) between the nodes. High Performance File Systems are also an option, if desired. Cloud Native applications optimize the use of Amazon S3 object storage in conjunction with local storage.
- **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**: HTC simulations often run across many—sometimes millions—of compute instances. Due to their loosely coupled nature, simulations can be deployed across Availability Zones without sacrificing performance. HTC simulations can be deployed with end-to-end solutions such as AWS Batch and CfnCluster, or through solutions based on AWS services such as Amazon Simple Queue Service (Amazon SQS), Auto Scaling, and AWS Lambda.

There are four example architectures to consider as a starting point for design patterns for HTC applications:

- Batch
- Queue
- Traditional
- Serverless

Batch-Based Architecture

AWS Batch is a fully managed service that enables you to run large-scale compute workloads on the cloud without having to provision resources or manage schedulers.[3] 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[4] and Spot Instances.[5] Without the need to install and manage batch computing software or server clusters that you use to run your jobs, you can focus on analyzing results and gaining new insights.

With AWS Batch, you package the code for your batch jobs, specify their dependencies, and submit your batch jobs using the AWS Management Console, CLIs, or SDKs. 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.

Reference Architecture

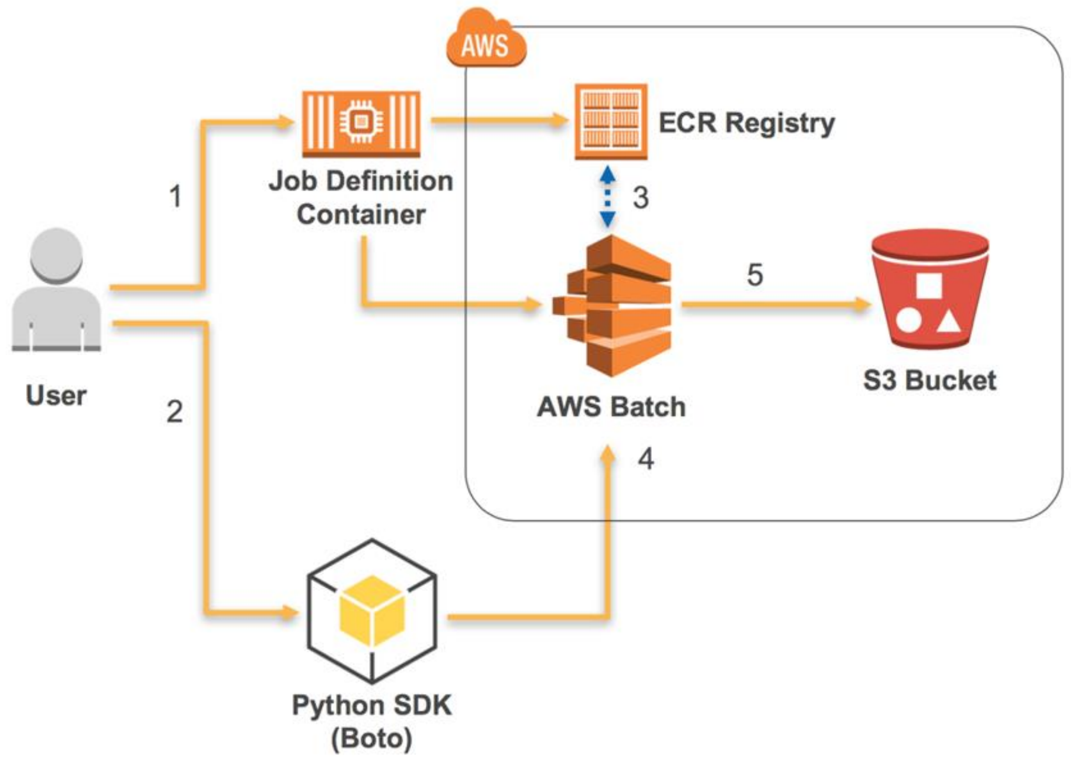

Figure 1: Example AWS Batch architecture for HTC workloads

Workflow steps:

1. User creates a job container, uploads the container to Amazon EC2 Container Registry (Amazon ECR) or another container registry (for example, DockerHub), and submits a job definition to AWS Batch.
2. User submits jobs to a job queue in AWS Batch.
3. AWS Batch pulls the container from the container registry.
4. AWS Batch processes the jobs in the queue.
5. The results from each job are 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.[6] Building applications from individual components that each perform a discrete function improves scalability and reliability. Decoupling components is a best practice for designing modern applications. Amazon SQS frequently lies at the heart of cloud-native HTC solutions.

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

Reference Architecture

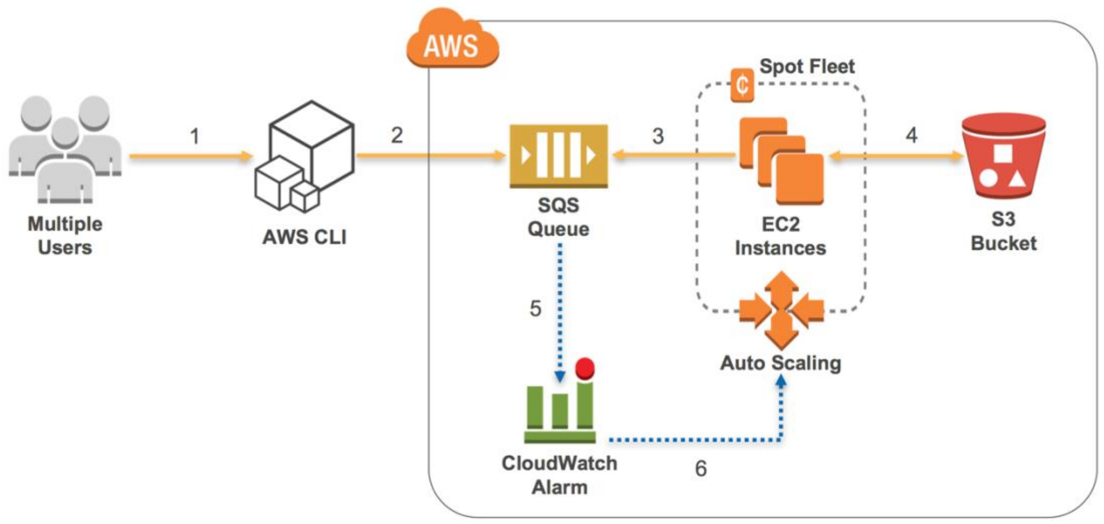

Figure 2: Example Amazon SQS-deployed HTC workload

Workflow steps:

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

Traditional Cluster Environment

Many HTC users begin their cloud journey with, or prefer an environment that is similar to, other HTC environments they've worked with before. This environment often involves a master node with a scheduler to launch runs or iterations of an HTC simulation.

A common approach to HTC cluster provisioning is based on an AWS CloudFormation template for a compute cluster combined with customization for a user's specific tasks. CfnCluster is an example of an end-to-end cluster provisioning capability based on AWS CloudFormation. While the complex description of the architecture is hidden inside the template, typical customization 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 HTC environment with the "look and feel" of conventional HPC clusters but with the added benefit of being scalable. The master node maintains the scheduler, shared file system, and running environment. Meanwhile, an auto-scaling mechanism allows for additional nodes to spin up as jobs are submitted to a job queue. As nodes become idle, they are automatically shut down again.

Reference Architecture

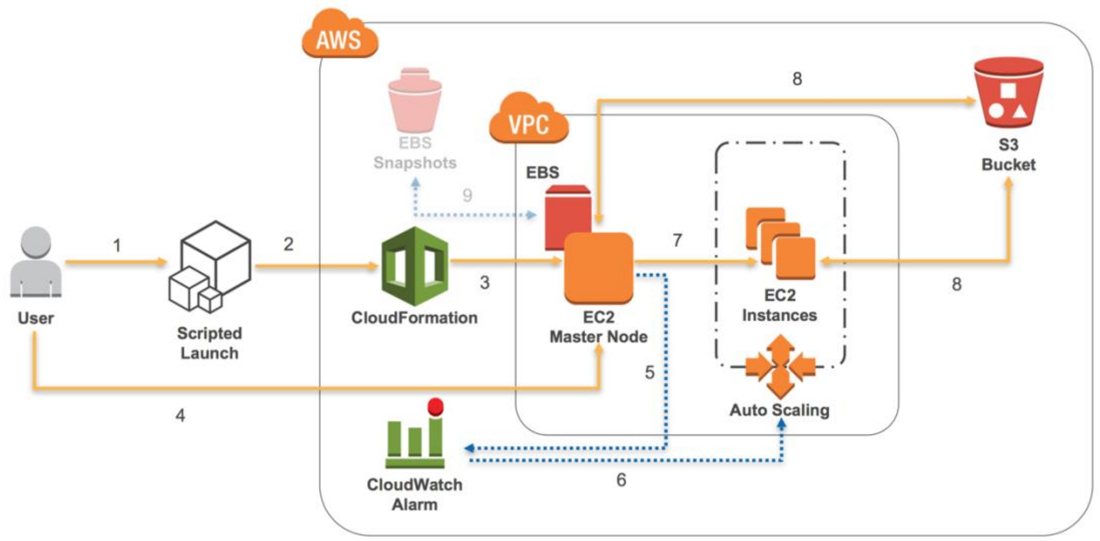

Figure 3: Example template-deployed HTC workload

Workflow steps:

1. A user initiates a cluster using the AWS CLI.
2. AWS CloudFormation executes the cluster architecture as described in a cluster template file, to which the user has contributed a few custom settings (for example, by editing a configuration file or using a web interface).
3. AWS CloudFormation deploys the infrastructure from snapshots created with customized HPC software/applications that cluster instances can access through an NFS export.
4. The user logs into the master node and submits jobs to the scheduler (for example, SGE).
5. The master node emits metrics to CloudWatch based on job queue size.
6. CloudWatch triggers Auto Scaling events to increase the number of compute nodes if the job queue size exceeds a threshold.
7. Scheduled jobs are processed by the compute fleet.
8. Source data and result data are stored in an S3 bucket.
9. Optional EBS snapshots can be taken to preserve changes to the EBS volumes.

Serverless

The HTC cloud journey often leads to an environment that is entirely serverless, meaning that you can concentrate on your applications and leave the server provisioning to managed services. AWS Lambda lets you 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 offers the capability of automatically triggering off of events from other AWS services.

Scalability is a second advantage of the serverless Lambda approach. Although each worker may be modest in size–say, 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. Not only does the largest attainable scale matter, but so does the speed of scaling. 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. In other words, AWS Lambda enables HPC infrastructure that can respond immediately to an unforeseen request for compute-intensive results and can fulfill a variable number of requests without requiring any resources to be provisioned wastefully in advance.

Reference Architecture

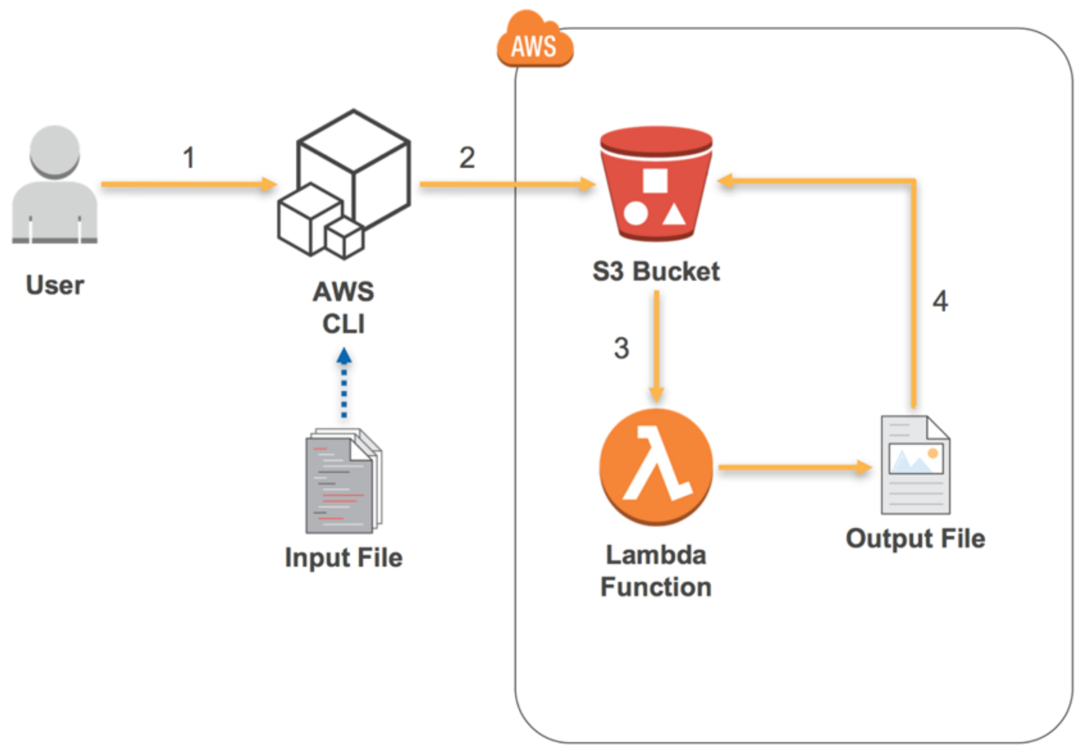

Figure 4: Example Lambda-deployed HTC workload

Workflow steps:

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

Tightly Coupled, High-Performance Computing

Tightly coupled HPC applications consist of parallel processes that are dependent on each other to carry out the calculation. Unlike an HTC simulation, all processes of a tightly coupled HPC simulation iterate together. An iteration is defined as one step of the overall simulation. HPC calculations generally 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, checkpointing regularly occurs during a simulation to allow for the restarting of a case.

HPC cases typically rely on a Message Processing Interface (MPI) for inter-process communication. Shared Memory Parallelism via OpenMP can be used in conjunction 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 generally very demanding. Slow communication between nodes typically results in the slowdown of the entire calculation. The largest instance size, enhanced networking, and placement groups are required for consistent networking performance. 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.
- **Storage**: Like HTC workloads, the storage requirements for tightly coupled workloads vary, driven by dataset size and the performance requirements for reading and writing data. A shared file system is often used, either from an NFS export on an instance with an EBS volume, Amazon Elastic File System (Amazon EFS) file system, or a high-performance file system. High-performance file systems can be obtained either from a third party in the AWS Marketplace or can be installed by the user.
- **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 allows the opportunity to launch hundreds of large multi-process cases at once, so there is no need to wait in a queue as there is with an on-premises cluster.

There are three example architectures to consider as starting points for design patterns for HPC applications:

- Persistent cluster
- Ephemeral cluster
- HPC microservices (a variation of the ephemeral cluster)

Persistent Cluster

A persistent cluster mimics a traditional on-premises cluster or supercomputer experience. Clusters include a master instance with a scheduler that allows multiple users to submit jobs. The compute node fleet can be a fixed size or tied to an Auto Scaling group to increase and decrease the compute fleet depending on the number of submitted jobs. The cluster is almost identical to that of the HTC cluster except that the cluster requires a placement group and homogenous instance types. In addition to the scheduler and shared volume, other libraries such as MPI may be added to the compute nodes. CfnCluster is an example of a persistent cluster.

Reference Architecture

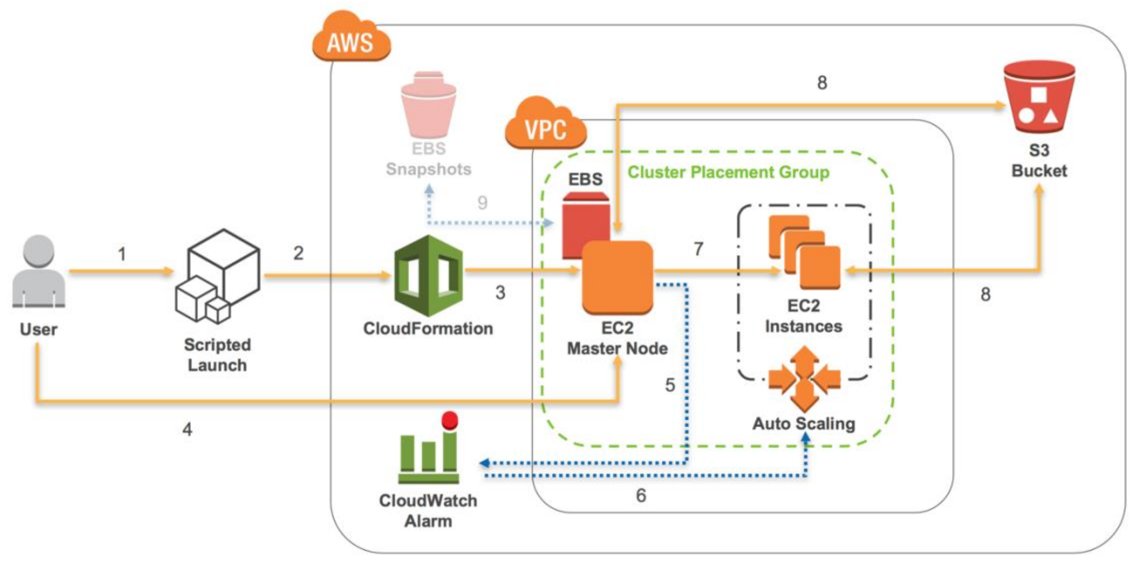

Figure 5: Example template- Based on a tightly-coupled architecture

Workflow steps:

1. AWS CloudFormation executes the cluster architecture as described in a cluster template file, to which the user has contributed a few custom settings (for example, by editing a configuration file or using a web interface). A placement group is included.
2. A snapshot created with customized HPC software/applications is used as the basis for the NFS-mounted EBS volume.
3. The user logs into the master node and submits jobs to the scheduler (for example, SGE).
4. The master node emits metrics to CloudWatch based on job queue size.
5. CloudWatch triggers Auto Scaling events to increase the number of compute nodes if the job queue size exceeds a threshold.
6. Scheduled jobs are processed by the compute fleet.
7. Results are stored on the shared volume and can be copied to Amazon Glacier or Amazon S3 for long-term storage.

Ephemeral Cluster

A cloud-native approach to tightly-coupled HPC ties each run to its own cluster. 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 retrieved automatically or sent to Amazon S3. Upon completion of the job, the infrastructure is terminated. Clusters designed this way treat infrastructure as code and allow for complete version control of infrastructure changes.

Master nodes and job schedulers are less critical and often not used at all with an ephemeral cluster. The Auto Scaler, a mainstay of the traditional cloud cluster, is also not used because clusters are stood up once and then terminated.

Reference Architecture

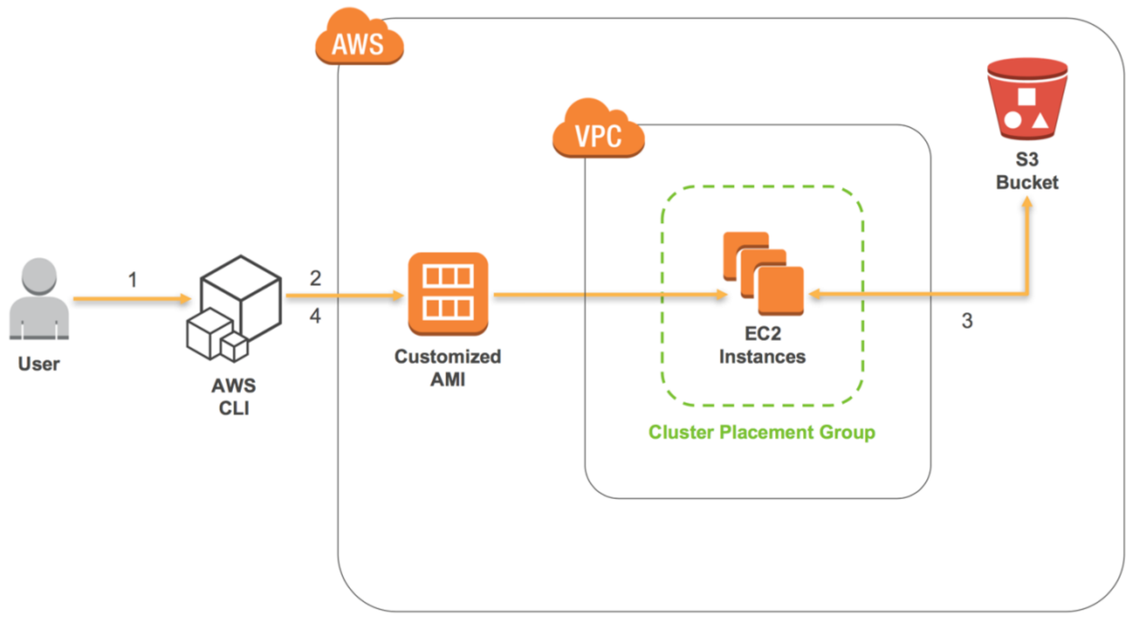

Figure 6: Example ephemeral-deployed tightly coupled HPC cluster

Workflow steps:

1. Users deploy each job or workload using a shell script using AWS CLI calls.
2. Each workload is launched onto its own customized cluster.
3. Output data is copied to Amazon S3 or securely copied back to the user's desktop.
4. Upon completion, the ephemeral cluster is terminated entirely.

HPC Microservices

A microservices architecture is a variation of the ephemeral cluster that provisions transient compute capacity based on an API call.

The goal is to present an HPC computation as an API to be initiated or consumed by another software agent, such as a web portal for users or a business application. HPC can then be integrated as part of an overall workflow or business process. This architectural approach presents an HPC computation as a service and enforces a modular approach. It also lends itself to continuous delivery allowing an HPC service owner to modify or enhance the service while maintaining a uniform interface via a defined API. Similar to ephemeral clusters, this approach is a departure from traditional HPC architectures and doesn't require the use of master nodes and schedulers.

With AWS, the API interface can be custom-defined using Amazon API Gateway or a service API such as for Amazon S3 or Amazon SQS.

Reference Architecture

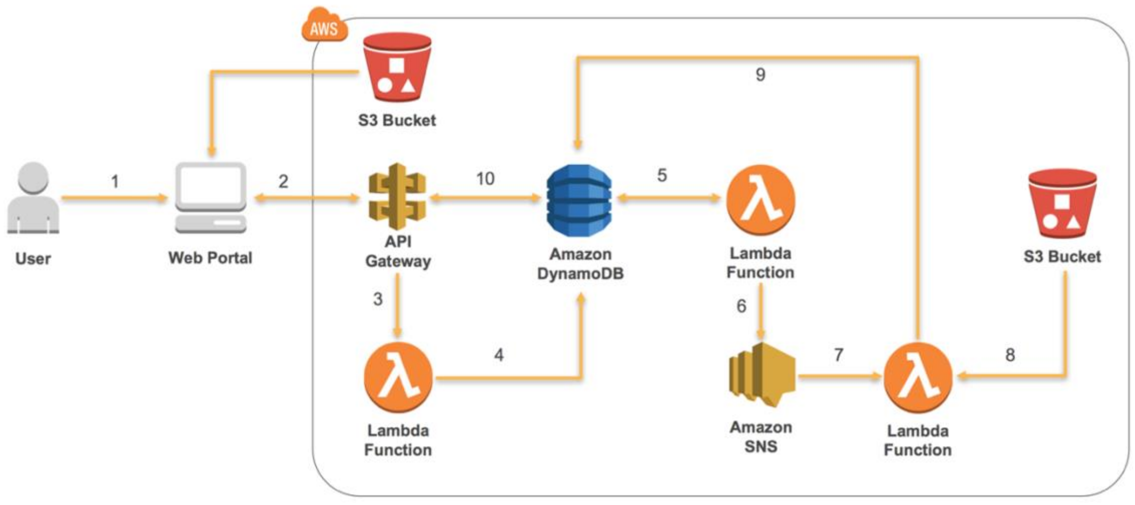

Figure 7: API-deployed microservices cluster

Workflow Steps:

1. User browses to a web portal hosted by a static Amazon S3 page.
2. User submits a job through client-side call to API Gateway.
3. API Gateway calls Lambda with submitted job and job parameters.

4. Lambda saves job and parameters in an Amazon DynamoDB table.
5. DynamoDB triggers a Lambda function to process the incoming job and determines job segmentation based on data indexes.
6. Lambda submits each segment as a message to Amazon Simple Notification Service (Amazon SNS).
7. Amazon SNS triggers a Lambda function to process each segment.
8. Lambda segments pull required data from Amazon S3 for processing.
9. Results are saved back to a DynamoDB table once each segment finishes.
10. User is updated with results during client-side polling to API Gateway.

The Pillars of the Well-Architected Framework

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

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.

Design Principles

In the cloud, there are a number of principles that drive operational excellence:

- **Traditional versus cloud-native**: HPC architectures typically take one of two forms. The first is a traditional cluster configuration with a head login instance, compute nodes, scheduler, and queue. The second is considered to be cloud-native with automated deployments, the use of serverless capability, managed services, and a single workload per (ephemeral) cluster. While the best approach depends on the environment sought for HPC users, cloud-native architectures can further optimize operational considerations.

- **Perform operations with code**: Automation offers operational excellence when there are repetitive processes or procedures. For example, when provisioning an HPC cluster consider automating the configuration management of compute nodes such as specifying shared storage, authentication, libraries, paths, and other common attributes. This can be automated by using user data, startup scripts, or infrastructure-as-code templates. An added benefit is the reproducibility of infrastructure. Also, consider automating responses to events such as job start, completion, or failure. Consider automating the job submission process. In HPC, it is quite common that users are expected to repeat multiple steps with every job including, for example, uploading case files, submitting a job to a scheduler, and moving result files. These repetitive steps can be automated with scripts or by event-driven code to maximize usability and minimize costs and failures.

Definition

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

- Prepare
- Operate
- Evolve

The **prepare** and **evolve** areas are described in the AWS Well-Architected Framework whitepaper. They will not be described here as the practices in the AWS Well-Architected Framework paper do not require modification for HPC workloads.

Best Practices

Prepare

There are no operational practices unique to HPC for the **prepare** practice area, so review the corresponding section in the AWS Well-Architected Framework whitepaper.

Operate

Operations should be standardized and managed on a routine basis. The focus should be 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, they should not require scheduled downtime, and they should not require manual execution. A wide range of logs and metrics that are based on key operational indicators for a workload should be collected and reviewed to ensure continuous operations.

When choosing AWS, you have 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, should be based on small incremental changes and tracked versions. You should be able to revert releases that introduce operational issues without causing operational impact. Tracked changes can vary from version control of source code, such as AWS CodeCommit, or infrastructure configuration, such as AWS CloudFormation templates.

HPCOPS 1: How are you evolving your workload while minimizing the impact of change?

HPCOPS 2: How do monitor your workload to ensure it's 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 demands. At the same time, cloud-native architectures for HPC operate differently, for example, using different mechanisms for job submission and provisioning resources on-demand as jobs arrive. Consider adopting operational procedures that benefit from and accommodate the elasticity of the cloud and the dynamic nature of cloud-native architectures.

Evolve

There are no operational practices unique to HPC for the **evolve** practice area, so review 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.

Design Principles

In the cloud, there are a number of principles that can help you strengthen your system security. HPC workloads often contain and create confidential and sensitive information. AWS security best practices are designed to protect your data.

- **Implement a principle of least privilege**: Ensure that authorization is appropriate for each interaction with your AWS resources and implement strong logical access controls directly on resources.
- **Focus on securing your system**: With the AWS Shared Responsibility Model you can focus on securing your application, data, and operating systems, while AWS provides secure infrastructure and services.
- **Automate security best practices**: Software-based security mechanisms improve your ability to securely scale more rapidly and cost-effectively. Create and save a patched, hardened image of a virtual server, and then use that image automatically on each new server you launch. Create an entire trust zone architecture that is defined and managed in a template via revision control. Automate the response to both routine and anomalous security events.
- **Limit exposure of sensitive data**: HPC data is typically produced within a limited time, allowing for migration of the data from the server to high-availability storage options such as on Amazon S3. This minimizes the possibility of unauthorized access of the data.
- **Enable traceability**: Log and audit all actions and changes to your environment.

Definition

There are 5 best practice areas for security in the cloud:

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

Before architecting any system, you need to put in place practices that influence security. You will want to control who can do what. In addition, you want to be able to 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 also provides greater access to security data and an automated approach to responding to security events.

The **detective controls**, **infrastructure protection**, and **incident response** categories are vital and well described in the AWS Well-Architected Framework whitepaper. They will not be described in this paper as the practices in the AWS Well-Architected Framework paper do not require modification for HPC workloads.

Best Practices

Identity and Access Management

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

In addition, consider running HPC workloads autonomously and ephemerally to limit exposure of sensitive data. Autonomous deployments require minimal user access to instances and thus minimize exposure of the resources. HPC data is typically produced within a limited time, minimizing the possibility of potential unauthorized data access.

HPCSEC 1: Are managed services, autonomous methods, and ephemeral clusters used to minimize user 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. SSH keys should be treated as private data and rotated regularly.

HPCSEC 2: What methods are you using to manage and rotate your SSH authentication keys?

Detective Controls

You can use detective controls to identify a potential security incident. These controls are an essential part of governance frameworks and can be used to support a quality process and legal and compliance obligations. They can also be used for threat identification and response efforts.

Infrastructure Protection

Infrastructure protection includes control methodologies, such as defense-in-depth and multi-factor authentication, which are necessary to meet best practices and industry and regulatory obligations. Use of these methodologies is critical for successful, ongoing operations in either the cloud or on-premises.

Data Protection

Before architecting any system, foundational practices that influence security should be in place. 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.

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.

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

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

Even with extremely mature preventive and detective controls, organizations should still put processes in place to respond to and mitigate the potential impact of security incidents. The architecture of your workload will strongly affect the ability of your teams to operate effectively during an incident to isolate or contain systems and to restore operations to a known-good state. Putting in place the tools and access ahead of a security incident, then routinely practicing incident response, will make sure the architecture is updated to accommodate timely investigation and recovery.

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.

Design Principles

In the cloud, a number of principles can help you increase reliability. In particular, the following are emphasized for HPC workloads. See also 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 from multiple cases, consider creating multiple clusters across the Amazon infrastructure to further isolate your risk of potential failures. Since infrastructure can be treated as code, not only can you horizontally scale resources inside a single cluster, but you can also 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 either manually or automatically to meet increases or decreases in demand. Compute nodes need not be idle when not in use and computations need not have long wait times because of limited resources.
- **Manage change in automation**: Changes to your infrastructure should be done using automation. This allows you to place a cluster infrastructure under version control and make exact duplicates of a previously created cluster. The changes that need to be managed are changes to the automation.

Definition

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

- Foundations
- Change management
- Failure management

The **change management** category will not be described in this paper as the best practices in the AWS Well-Architected Framework whitepaper do not require modification for HPC workloads. Please refer to the AWS Well-Architected Framework paper for an understanding of best practices.

Best Practices

Foundations

The cloud is designed to be essentially limitless, so it is the responsibility of AWS to satisfy the requirement for sufficient networking and compute capacity. 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, it 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.

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

Service limits often need to be increased from the default values to handle the requirements of a large deployment. You can contact AWS Support to request an increase.

Change Management

Being aware of how change affects a system allows you to plan proactively. Monitoring allows you to quickly identify trends that could lead to capacity issues or SLA breaches. In traditional environments, change-control processes are often manual and must be carefully coordinated with auditing to effectively control who makes changes and when they are made.

Failure Management

In any system of reasonable complexity, it is expected that failures will occur, and it is generally of interest to know how 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.

Failure tolerance can be improved in multiple ways. For long-running cases, incorporating regular checkpoints in your code will allow 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 the running application to periodically write out intermediate results. They can be used to diagnose an application error or to restart the case as needed, while only losing the work done between the last checkpoint and the job failure.

HPCREL 2: Does your application support checkpointing to recover from failures?

Checkpointing is particularly useful on Spot Instances, that is, when using highly cost-effective but pre-emptible nodes that can be interrupted at any time. In addition, some applications may benefit from checkpointing and changing the default Spot interruption behavior (e.g., stopping or hibernating the instance rather than terminating it). Lastly, it is important to consider the durability of the storage option when relying on checkpointing for failure management.

Failure tolerance can potentially be improved by deploying to multiple Availability Zones. The low-latency requirements of tightly coupled HPC applications require that each individual case reside within a single placement group and Availability Zone. Alternatively, HTC 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 outside of the origin 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. Another possibility is to do nothing in response to an Availability Zone interruption and simply wait a short time for it to recover.

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

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.

Design Principles

When designing for HPC in the cloud, a number of principles can 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 needed. When running a variety of applications on AWS, a variety of architectures can be used to meet each application's demands.
- **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 frequently 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.
- **Consider cloud-native architectures**: 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 allows each step in the workload process to be decoupled and optimized.
- **Experiment more often**: With virtual and automatable resources, you can quickly carry out comparative testing using different types of instances, storage, or configurations.

Definition

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

- Selection
- Review
- Monitoring
- Tradeoffs

The **review**, **monitoring**, and **tradeoffs** categories will not be described in this paper because the best practices in the AWS Well-Architected Framework whitepaper do not require modification for HPC workloads. Refer to the AWS Well-Architected Framework paper for an understanding of best practices.

Best Practices

Selection

The optimal solution for a particular system will vary based on the kind of workload you have. Often there are multiple approaches possible. 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

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 support specific workloads, for example, compute-, memory-, or GPU-intensive workloads, while others 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. However, compute-intensive workloads will see significant performance degradation on the T-series instance family, even with Unlimited mode enabled, since its compute resources are designed for applications with moderate CPU usage that experience temporary spikes in usage.

Within each instance family, one or more instance sizes (see the Instance Type Matrix[7]), allow vertical scaling of resources. Some applications require a larger instance type (for example, 16xlarge) while others run on smaller types (for example, large). Many smaller instances may be preferred for an HPC application over fewer larger instances.

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, start 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 are 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.

The following example question focuses on compute resources:

HPCPERF 1: How do you select your compute solution?

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: Hardware virtual machine (HVM) Amazon Machine Images (AMIs) can take advantage of special hardware extensions (CPU, network, and storage) for better performance. The HVM virtualization type is recommended for HPC applications. Previously, AWS also offered the paravirtual (PV) virtualization type. PV is considered a legacy approach and is no longer supported on current generation instance types.

HPCPERF 2: How do you select your operating system and virtualization type?

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

First, HPC applications can greatly benefit from these advanced processor features (for example, Advanced Vector Extensions) and can considerably increase their calculation speeds by simply compiling the software for the Intel architecture.[8] The compiler options for architecture-specific instructions vary by compiler (check the usage guide for your compiler).

Second, 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). Other HPC applications benefit by disabling hyperthreading, and 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.

Third, 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 will depend on your workload and application.

Lastly, AWS enables you to tune the processor state control on certain instance types.[9] You may want to alter 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, consider experimenting with the C-state or P-state settings that are available on select instances.

HPCPERF 3: How do you optimize your compute environment for your application?

There are many compute options available to optimize a compute environment. Choices can severely affect the performance of an application. Cloud deployment allows for experimentation on every level from instance type, operating system, and AMI type. As static clusters are tuned before deployment, time spent experimenting with cloud-based clusters is vital to achieving the desired performance.

Storage

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, AWS Partner solutions, and open-source configurations deployed on EC2 instances. High-performance file systems can be created from Amazon EFS, Amazon EBS volumes, and 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 storage or shared storage. 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. Choosing an EBS-optimized instance provides more consistent performance and is recommended 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, you should choose an instance type that enables EBS optimization by default or choose an instance type that allows enabling EBS optimization at launch. Please refer to the instance type matrix for EBS optimization support.[10]

Lastly, instance-store volumes, including non-volatile memory express (NVMe) SSD volumes (only available on certain instance families), can be used for temporary block-level storage. Shared network storage using instance-store volumes typically implement clustered file systems (e.g., Lustre) across instances to allow for underlying hardware failures and instance lifecycle events (e.g., stop, terminate).

The following example question focuses on storage considerations for performance efficiency:

HPCPERF 4: How do you select your storage solution?

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

The optimal network solution for an HPC workload will vary 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 often possible to select a large instance type with a large number of cores such that the application fits entirely within the instance without crossing the network at all. However, for large tightly coupled workloads, multiple instances are required with low latency between the instances. 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 bi-section bandwidth between instances. Cluster placement groups are recommended for tightly coupled applications and can be created through the AWS Management Console, CLI, or API.

HTC cases are generally not sensitive to very low latency networking and typically do not require the use of a placement group or the need to keep instances in the same Availability Zone or Region. For maximum flexibility, cluster placement groups should not be used unless your application requires consistent, low latency networking between compute instances.

The best network performance may be obtained with the largest instance type in a family. Please refer to the instance type matrix for details.

For optimal network performance, you should select 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 inter-instance latency compared to traditional device virtualization.

Enhanced networking is recommended for HPC applications and is available on all current-generation instance types, with the exception of the T2 instance family. Enhanced networking 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, see the enhanced networking documentation.[11]

The following example question focuses on network considerations for performance efficiency:

HPCPERF 5: How do you select your network solution?

Review

When architecting solutions, there is a finite set of options from which you can choose. However, over time new technologies and approaches become available that can improve the performance of your architecture.

Monitoring

After you have implemented your architecture, you will need to monitor its performance so that you can fine-tune it for maximum performance. Monitoring metrics should be used to gain insight into resource performance and raise alarms when thresholds are breached. The alarm can trigger automated action to work around any poorly performing components.

In addition to the monitoring considerations outlined in the AWS Well-Architected Framework whitepaper, many HPC users find detailed Amazon EC2 monitoring useful when gauging application performance. Detailed monitoring shortens the monitoring interval from five minutes to one minute for your instance metrics. Detailed monitoring can be enabled through the AWS Management Console, CLI, or API.

Trade-offs

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

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 will enable you to build and operate cost-aware systems that achieve business outcomes and minimize costs, thus allowing your business to maximize its return on investment.

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 use a large and expensive server for all tasks, 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 cheaper servers. Costs are reduced by optimizing infrastructure for each task at each step of a workload.
- **Burst workloads in the most efficient way**: With cloud, savings are often obtained for HPC workloads by bursting horizontally. When bursting horizontally, many jobs or iterations of an entire workload are run simultaneously for less total elapsed time. Depending on the application, horizontal scaling can potentially 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 and bursty nature of HPC workloads makes them very well suited to Spot Instances (as opposed to, for example, an uninterruptible workload, such as database hosting). The risk of Spot Instance interruption can be minimized by working with the Spot Advisor, and the interruption impact can be potentially mitigated by changing the default interruption behavior and using Spot Fleet to manage your Spot Instances. The need to occasionally restart a workload is often easily offset by the cost savings of Spot Instances.
- **Assess the tradeoff of cost versus time**: Tightly coupled, massively parallel workloads are often 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 be 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 be run more quickly on more cores, while cost savings can be achieved by running on fewer cores and at maximum efficiency. Workloads can also 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** categories are vital and well described in the AWS Well-Architected Framework whitepaper. They will not be described in this paper because the practices in the AWS Well-Architected Framework paper do not require modification for HPC workloads.

Best Practices

Cost-Effective Resources

Using the appropriate instances and resources 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. Therefore, it is important to consider the potential tradeoff between job turnaround and cost optimization and to consider testing workloads with different instance sizes and storage options to optimize cost.

A well-architected system will use 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.

It is important to note that HPC workloads that are sensitive to low network latency or high network throughput should be run in a cluster placement group to minimize run times and reduce overall costs.

HPCCOST 1: Have you evaluated different instance types and storage options for your workload to assess optimal cost?

HPCCOST 2: Have you considered the trade-off between job turnaround time and cost?

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

Matching Supply and Demand

Optimally matching supply to demand delivers the lowest costs for an HPC system. Demand can be fixed or variable, requiring metrics and automation to ensure optimization. With AWS, you can automatically provision resources to match demand with Auto Scaling. Auto Scaling allows you to add and remove resources as needed.

Expenditure Awareness

The increased flexibility and agility that the cloud enables encourages innovation and fast-paced development and deployment. It eliminates the manual processes and time associated with provisioning on-premises infrastructure, including identifying hardware specifications, negotiating price quotations, managing purchase orders, scheduling shipments, and then deploying the resources. However, the ease of use and virtually unlimited on-demand capacity may require a new way of thinking about expenditures.

Optimizing Over Time

As AWS releases new services and features, it is a best practice to review your existing architectural decisions to ensure they continue to be the most cost-effective. As your requirements change, be aggressive in decommissioning resources, entire services, and systems that you no longer require.

The cloud allows you to perform a hardware refresh much more frequently than on premises, as there is little cost or effort involved in moving up to a newly released instance family. This can be as easy as editing cluster configuration settings and spinning up new compute instances. Hence, it is important to stay up-to-date on launch announcements and to be willing to refresh hardware more aggressively.

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 have covered prototypical HPC architectures and overarching HPC design principles. We've also revisited the five 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 will help you build stable and efficient systems, leaving you to focus on running HPC applications and pushing the boundaries of the field to which you're committed.

Contributors

The following individuals and organizations contributed to this document:

- Aaron Bucher, HPC Specialist Solutions Architect, Amazon Web Services
- Kevin Jorissen, Technical Business Development Manager, Amazon Web Services
- Omar Shorbaji, Solutions Architect, Amazon Web Services
- Linda Hedges, HPC Specialist Solutions Architect, Amazon Web Services
- Philip Fitzsimons, Sr. Manager Well-Architected, Amazon Web Services

Further Reading

For additional information, see the following:

- AWS Well-Architected Framework[12]
- https://aws.amazon.com/hpc
- https://d1.awsstatic.com/whitepapers/Intro_to_HPC_on_AWS.pdf
- https://d1.awsstatic.com/whitepapers/optimizing-electronic-design-automation-eda-workflows-on-aws.pdf
- https://aws.amazon.com/blogs/compute/real-world-aws-scalability/

Notes

1 https://aws.amazon.com/well-architected

2 https://d0.awsstatic.com/whitepapers/architecture/AWS_Well-Architected_Framework.pdf

3 https://aws.amazon.com/batch/

4 https://aws.amazon.com/ec2/

5 https://aws.amazon.com/ec2/spot/

6 https://aws.amazon.com/message-queue

7 https://aws.amazon.com/ec2/instance-types/#instance-type-matrix

8 https://aws.amazon.com/intel/

9 http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/processor_state_control.html

10 http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSOptimized.html#ebs-optimization-support

11 http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/enhanced-networking.html

12 https://aws.amazon.com/well-architected