PRJ-DOP-026: High-Performance Computing (HPC) & Batch Processing

Certification: AWS Certified DevOps Engineer – Professional
Domain: High-Performance Computing and Batch Processing

---

1. Project Overview

This project explores two primary methods for running large-scale High-Performance Computing (HPC) and batch processing workloads on AWS. These workloads, common in scientific research, financial modeling, and media rendering, require massive computational power for a limited time. The challenge is to provision and manage this power cost-effectively without maintaining a large, expensive, and often idle on-premises cluster.

We will first use **AWS ParallelCluster**, an open-source cluster management tool, to quickly set up a traditional HPC environment with a familiar job scheduler like **Slurm**. This is ideal for users accustomed to classic HPC environments. We will then explore **AWS Batch**, a fully managed service that provides a more cloud-native approach to running batch jobs, abstracting away the underlying cluster management and integrating directly with Docker containers.

Key Objectives

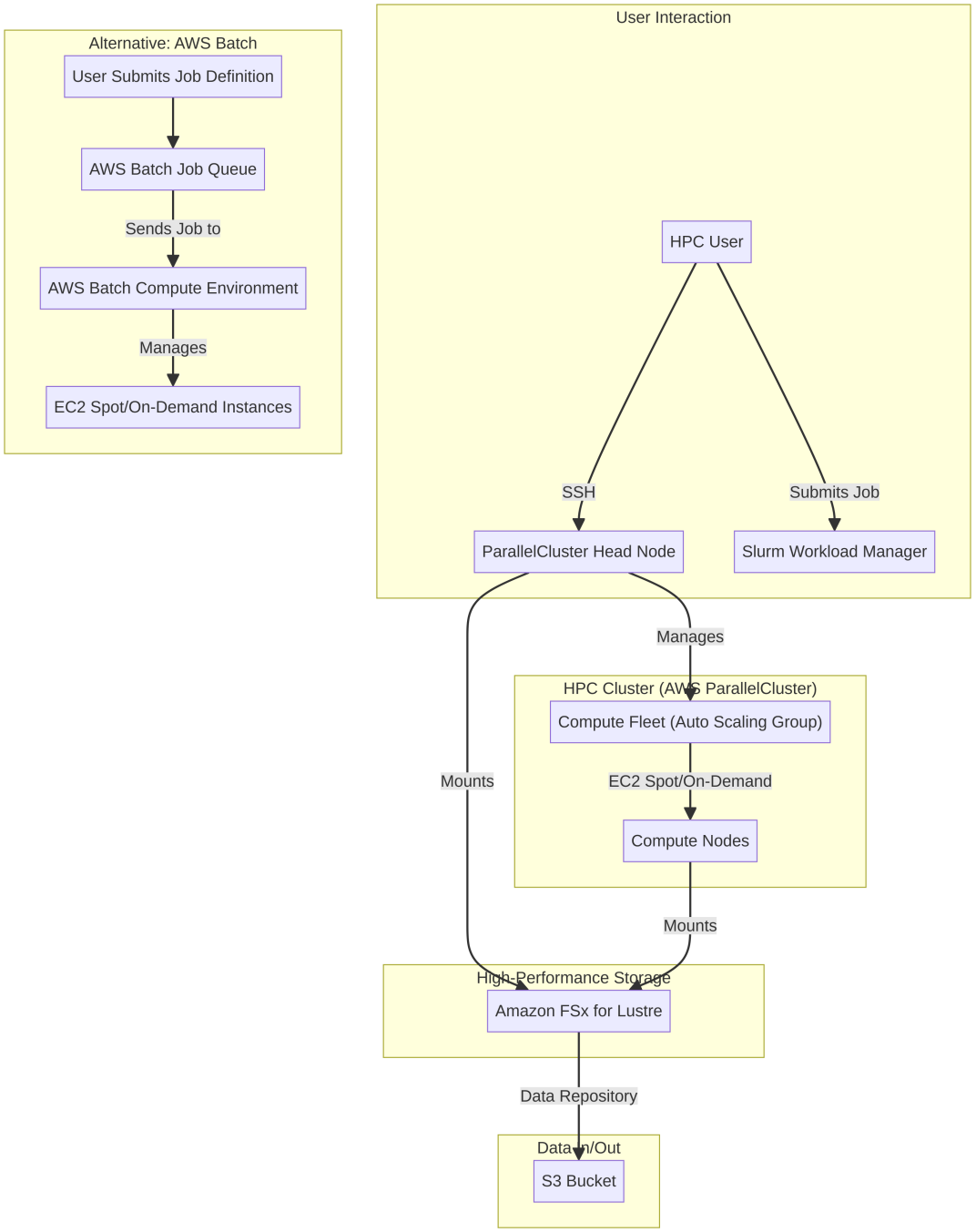
- Understand the use cases for HPC and batch processing on AWS.
- Deploy a flexible, auto-scaling HPC cluster using AWS ParallelCluster.
- Submit and manage jobs on the cluster using the Slurm workload manager.
- Integrate a high-performance parallel file system using **Amazon FSx for Lustre**.
- Set up a cloud-native batch processing environment using AWS Batch.
- Define job definitions, compute environments, and job queues in AWS Batch.

- Leverage **EC2 Spot Instances** in both scenarios to dramatically reduce computational costs.

---

2. Architecture

We will explore two distinct architectures for this project.



Architecture 1: AWS ParallelCluster

- **User Interaction:** A user (e.g., a scientist or researcher) connects via SSH to a single **Head Node**. They submit their computational jobs using standard Slurm commands (`sbatch`, `squeue`).
- **Cluster Management:** The Head Node runs the Slurm workload manager. It manages a **Compute Fleet**, which is an Auto Scaling Group of EC2 instances.
- **Auto Scaling:** When jobs are submitted, ParallelCluster automatically scales out the compute fleet, adding nodes to run the jobs. When the jobs are complete, it scales the fleet back down to zero (or a configured minimum), ensuring you only pay for the compute you use.
- **Storage:** All nodes in the cluster, including the head node, mount a shared, high-performance file system provided by **Amazon FSx for Lustre**. This provides a POSIX-compliant file system with extremely high throughput and low latency, essential for many HPC applications. The FSx file system can be linked to an S3 bucket as a durable data repository.

Architecture 2: AWS Batch

- **User Interaction:** A user defines a **Job Definition**, which is a blueprint for a job (essentially a Docker container image, command, and resource requirements like vCPU and memory).
- **Job Submission:** The user submits a job to a **Job Queue**. AWS Batch holds the job in the queue.
- **Compute Environment:** The job queue is mapped to a **Compute Environment**. This is a managed pool of EC2 instances (which can be Spot, On-Demand, or a mix). AWS Batch automatically manages the scaling of this compute environment based on the number and requirements of the jobs in the queue.
- **Job Execution:** When capacity is available in the compute environment, AWS Batch takes a job from the queue, provisions an EC2 instance, runs the job (as a Docker container), and terminates the instance when the job is complete.

---

3. Prerequisites

- An AWS account with administrative permissions.
- The AWS CLI installed and configured.
- A key pair for SSH access.

---

4. Step-by-Step Guide: AWS ParallelCluster

Step 4.1: Install AWS ParallelCluster CLI

```
sudo pip3 install aws-parallelcluster
```

Step 4.2: Create a Cluster Configuration File

Create a file named `hpc-cluster-config.yaml`.

```
Region: <YOUR_AWS_REGION>
Image:
  Os: alinux2
HeadNode:
  InstanceType: t2.xlarge
  Networking:
    SubnetId: <YOUR_PUBLIC_SUBNET_ID>
  Ssh:
    KeyName: <YOUR_KEY_PAIR_NAME>
Scheduling:
  Scheduler: slurm
  SlurmQueues:
    - Name: spot-queue
      ComputeResources:
        - Name: spot-compute-nodes
          InstanceType: c5.large
          MinCount: 0
          MaxCount: 10
          SpotPrice: 0.05 # Optional: Set a max Spot price
      Networking:
        SubnetIds:
          - <YOUR_PRIVATE_SUBNET_ID>
SharedStorage:
  - Name: FsxLustreStorage
    StorageType: FsxLustre
    FsxLustreSettings:
      StorageCapacity: 1200 # Minimum size is 1200 GiB
```

Important: Replace the placeholders for region, subnet IDs, and key pair name.

Step 4.3: Create the Cluster

```
pcluster create-cluster --cluster-name my-hpc-cluster --cluster-
configuration hpc-cluster-config.yaml
```

This command will use CloudFormation to provision the entire cluster. It can take 15-20 minutes.

Step 4.4: Connect and Submit a Job

1. **Connect to the Head Node:**

```
pcluster ssh --cluster-name my-hpc-cluster
```

2. **Create a Sample Job Script:** Create a file named `sleep.sh` on the head node.

```
#!/bin/bash
#SBATCH --comment="My first Slurm job"
#SBATCH --nodes=1
#SBATCH --output="slurm.out"


echo "Hello from an HPC node!"
sleep 60
echo "Job finished."
```

3. **Submit the Job:**

```
sbatch sleep.sh
```

4. **Monitor the Cluster:**

   - Run `squeue` to see your job in the queue (it will be in a `PENDING` state).
   - In the EC2 console, you will see ParallelCluster automatically launching a new `c5.large` Spot Instance to run your job.
   - Once the instance is ready, the job will run (check `squeue` again, it will be `RUNNING`).
   - After 60 seconds, the job will finish. The output will be in `slurm.out`.
   - A few minutes after the queue is empty, ParallelCluster will automatically terminate the compute node to save costs.

Step 4.5: Cleanup

```
pcluster delete-cluster --cluster-name my-hpc-cluster
```

---

5. Step-by-Step Guide: AWS Batch

Step 5.1: Create a Docker Image for the Job

- Create a simple application that performs a task (e.g., a Python script that processes a file).
- Create a `Dockerfile` and push the image to Amazon ECR.

Step 5.2: Create the AWS Batch Environment

1. **Create Compute Environment:**
    - Go to the **AWS Batch console -> Compute environments -> Create**.
    - **Name:** `spot-compute-env`
    - **Service role / Instance role:** Let Batch create new roles for you.
    - **Provisioning model:** Spot
    - **Allowed instance types:** `optimal` (lets Batch choose the best instance type).
    - **Minimum vCPUs:** 0
    - **Maximum vCPUs:** 16
    - Configure networking (VPC and subnets).
2. **Create Job Queue:**
    - Go to **Job queues -> Create**.
    - **Name:** `high-priority-queue`
    - Connect it to the `spot-compute-env` you just created.
3. **Create Job Definition:**

- Go to **Job definitions -> Create**.
- **Name:** `my-batch-job`
- **Platform:** EC2
- **Container image:** Enter the URI of the Docker image you pushed to ECR.
- **Command:** The command to run inside the container.
- **vCPUs / Memory:** Specify the resources your job requires.

Step 5.3: Submit a Job

1. Go to **Jobs -> Submit new job**.
2. **Name:** `test-job-01`
3. **Job definition:** Select `my-batch-job`.
4. **Job queue:** Select `high-priority-queue`.
5. Click **Submit**.

Step 5.4: Monitor the Job

- In the **Dashboard**, you will see your job move from `SUBMITTED` to `RUNNABLE` to `STARTING` and finally `RUNNING`.
- AWS Batch will automatically provision an EC2 instance in the background, run your containerized job, and then terminate the instance.
- You can view the logs for your job directly in the AWS Batch console, which pulls them from CloudWatch Logs.

---

6. Comparison: ParallelCluster vs. AWS Batch

Feature	AWS ParallelCluster	AWS Batch
Model	Cluster-centric	Job-centric
User Experience	Traditional HPC (SSH, scheduler commands)	Cloud-native (API, SDK, Console)
Workload Scheduler	Slurm, Grid Engine (User manages)	AWS Managed (User has no access)
Application Format	Binaries, scripts on a shared file system	Docker Containers
Best For…	Tightly-coupled MPI jobs, users migrating from on-premises clusters.	Loosely-coupled/embarrassingly parallel jobs, container-based workflows.

---

7. Cleanup

1. **Delete the Job Definition.**
2. **Delete the Job Queue.**
3. **Delete the Compute Environment.**
4. **Delete the ECR repository** and the IAM roles created by Batch.

Business Context

The Problem

Development teams face slow deployment cycles, manual testing bottlenecks, and security vulnerabilities introduced during the CI/CD process. Traditional deployment methods lack security scanning, leading to vulnerabilities reaching production. Manual processes create inconsistency and human error.

The Solution

Secure CI/CD pipeline with integrated security scanning, automated testing, and infrastructure as code. Implements shift-left security with SAST, DAST, and dependency scanning. Automates deployments while enforcing security gates and compliance checks at every stage.

Business Value

- **Faster Time to Market:** Automated pipelines reduce deployment time from weeks to hours
- **Improved Security Posture:** Catches vulnerabilities before production deployment
- **Reduced Manual Effort:** Eliminates 80%+ of manual deployment tasks
- **Audit Trail:** Complete deployment history and security scan results for compliance

Risk Mitigation

Prevents deployment of vulnerable code, hardcoded secrets, misconfigured infrastructure, and non-compliant resources to production environments.

GRC Mapping

Compliance Frameworks

- **NIST CSF:** PR.IP-1 (Baseline configuration), PR.DS-6 (Integrity checking), DE.CM-4 (Code analysis)
- **ISO 27001:** A.12.1 (Operational procedures), A.14.2 (Security in development), A.12.4 (Logging)
- **CIS Controls:** Control 16 (Application Software Security), Control 11 (Secure Configuration)
- **OWASP DevSecOps:** Security testing integration, Secret management, Dependency checking

Security Controls Implemented

- Static Application Security Testing (SAST)
- Dynamic Application Security Testing (DAST)
- Software Composition Analysis (SCA)
- Secrets scanning and prevention
- Infrastructure as Code security validation

Audit Evidence

- Pipeline execution logs with security scan results
- Code commit and approval records
- Security test reports (SAST/DAST/SCA)
- Deployment approval and rollback records

Regulatory Alignment

- **SOX:** Section 404 (Change management controls)
- **PCI DSS:** Requirement 6.3 (Secure development), Requirement 6.5 (Common vulnerabilities)
- **GDPR:** Article 25 (Data protection by design), Article 32 (Security of processing)
- **SOC 2:** CC8.1 (Change management), CC7.2 (System monitoring)