AWS Batch Deep Dive

Sean Smith, Sr. HPC SA
seaam@amazon.com

What is AWS Batch?

Job Scheduler

Orchestrator

Batch Core Components

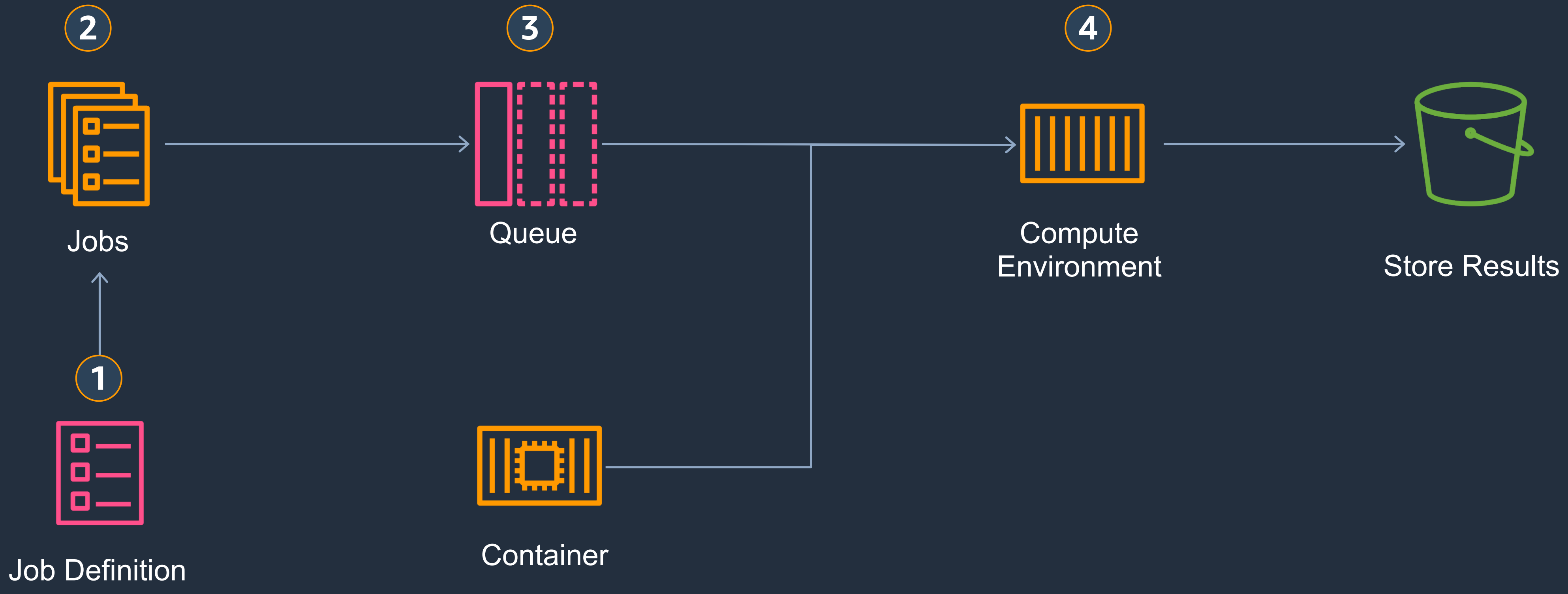
AWS Batch overview

1 **Job Definition**
Template that has common attributes (container image, IAM role, vCPU & memory requirements, ,…

2 **Job**
Each job must reference a job definition, but many parameters may be overridden when submited

3 **Job Queue (JQ)**
Queue determines priorities. Each JQ is connected to 1 or more CE

4 **Compute Environment (CE)**
Resource Mix (defines On-demand vs. Spot and instance types. CE can be connected to more than one JQ



Job Definitions

AWS Batch **job definitions** specify how jobs are to be run.

Some attributes in a job definition:

Container Image	Amazon ECR, DockerHub, private registry or regular storage
IAM role associated with the job	Actions permitted/forbidden on services and resources
vCPU and memory requirements	Memory, swap memory, shared memory
Volumes	Mount points, docker volumes, tmpfs
Environment variables	Shell variable transmitted to the job (parameters)
Retry strategy	# of retries in case of failure, custom retries New

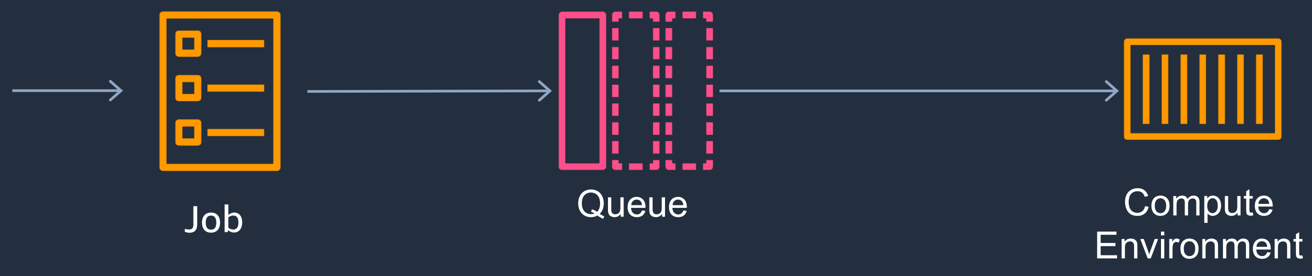
Job definitions are templates, parameters can be overridden

https://docs.aws.amazon.com/batch/latest/APIReference/API_RegisterJobDefinition.html

Jobs

A job is the unit of work that will be processed by Amazon EC2 through AWS Batch.

Parameters through Job Definition or defined at submission time. Instances are selected based on CPU, Mem*, GPU.



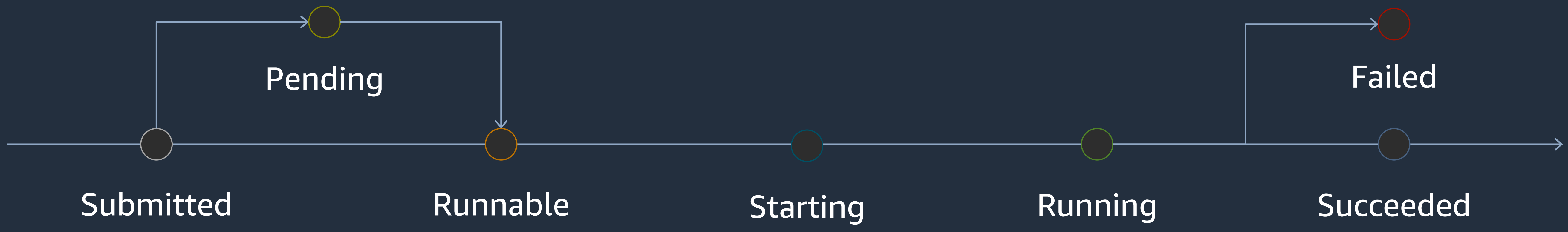
- Types of jobs:
  - Atomic: 1 or multiple jobs
  - Array: group of jobs with shared parameters (max 10k child jobs)
  - Multi-Node Parallel (MNP): MPI or NCCL
- Job dependencies: wait for a another job to complete

```
"dependsOn": [
      {
          "jobId": "IDJobA",
          "type": "SEQUENTIAL|N_TO_N"
      }
]
```

Expressing dependencies

* 32MB of memory reserved with ECS_RESERVED_MEMORY
https://docs.aws.amazon.com/batch/latest/userguide/memory-management.html#ecs-reserved-memory
https://docs.aws.amazon.com/batch/latest/userguide/job_definitions.html

Jobs states

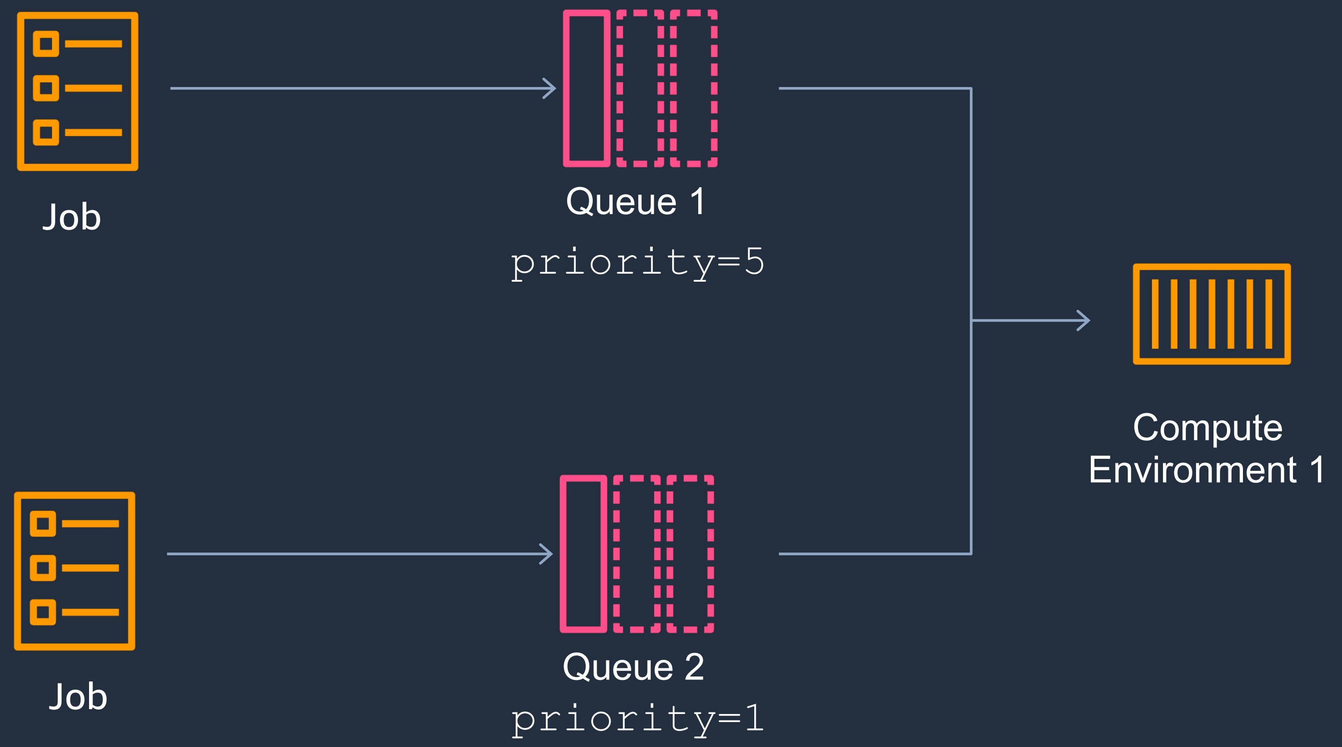


- SUBMITTED: accepted into the queue, but not yet evaluated by the scheduler for execution
- PENDING: the job has dependencies on other jobs which have not yet completed
- RUNNABLE: the job is evaluated by the scheduler and is ready to run
- STARTING: the job is in the process of being scheduled to a compute resource
- RUNNING: the job is currently running
- SUCCEEDED: the job has finished with exit code 0
- FAILED: the job finished with a non-zero exit code, was cancelled or terminated

Job Queues (JQ)

Job Queues are were jobs are submitted and reside throughout their lifetime

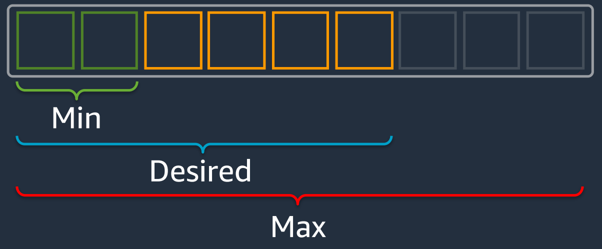
- A single queue can connect to 1 or a set of Compute Environments (CEs) and can share a CE with other queues
- Some parameters
  - Priority: scheduling priority to assign a job to a CE shared with multiple JQs in ascending order
  - CE Order: placement in descending order (0 first)



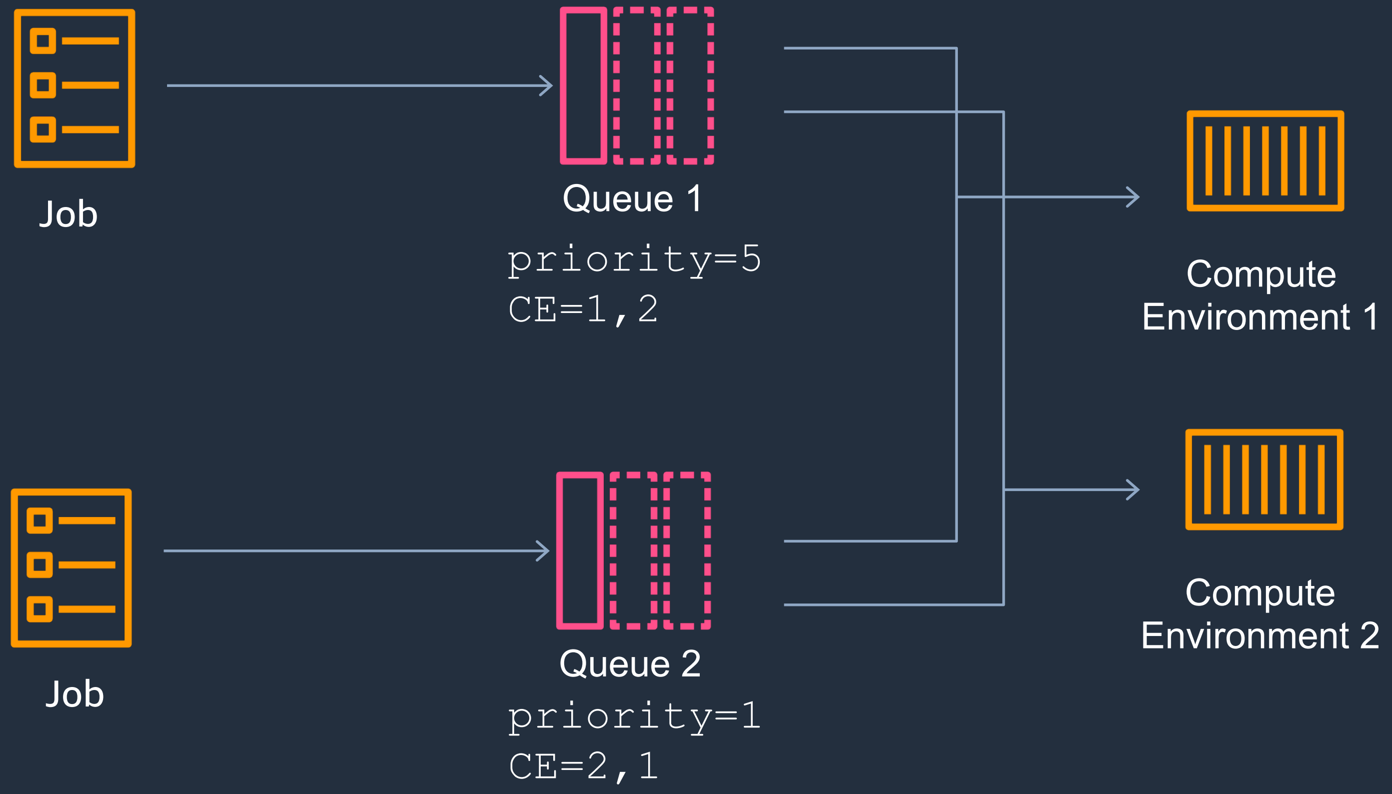
Compute Environments (CEs)

Compute Environments contain the underlying resources that are used to run jobs

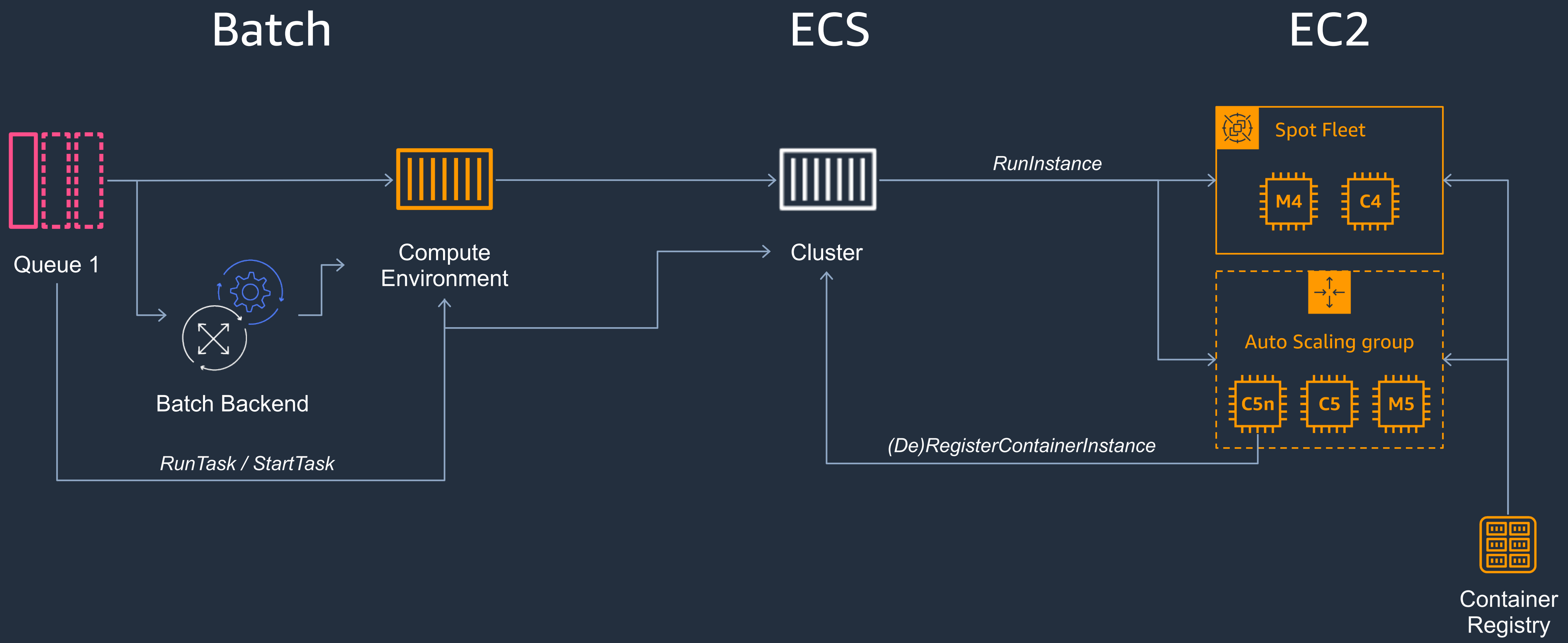
- Types
  - Managed: AWS scales and configures underlying instances (recommended)
  - Unamanaged: Customers control and manage instance configuration, provisioning and scaling
- Parameters
  - Scaling:



  - Instances types: instance families or specific instances on which jobs will be running



Compute high level structure



How AWS Batch scales

Allocation strategies

- `BEST_FIT` (default in the CLI)

  Pick the least number of instances that can fit the jobs requirements at the lowest cost regardless of the type and size within your selection of instances. Will diversify across instance families.

- `BEST_FIT_PROGRESSIVE` (default for OD in the console)

  Same as best fit but will select instances of the same family in priority, then look at other families if $/vCPUs & requirements cannot be met.

- `SPOT_CAPACITY_OPTIMIZED` (default for Spot in the console)

  Pick instances in chosen families and focus on pools with lower chances of interruptions based on historical data.

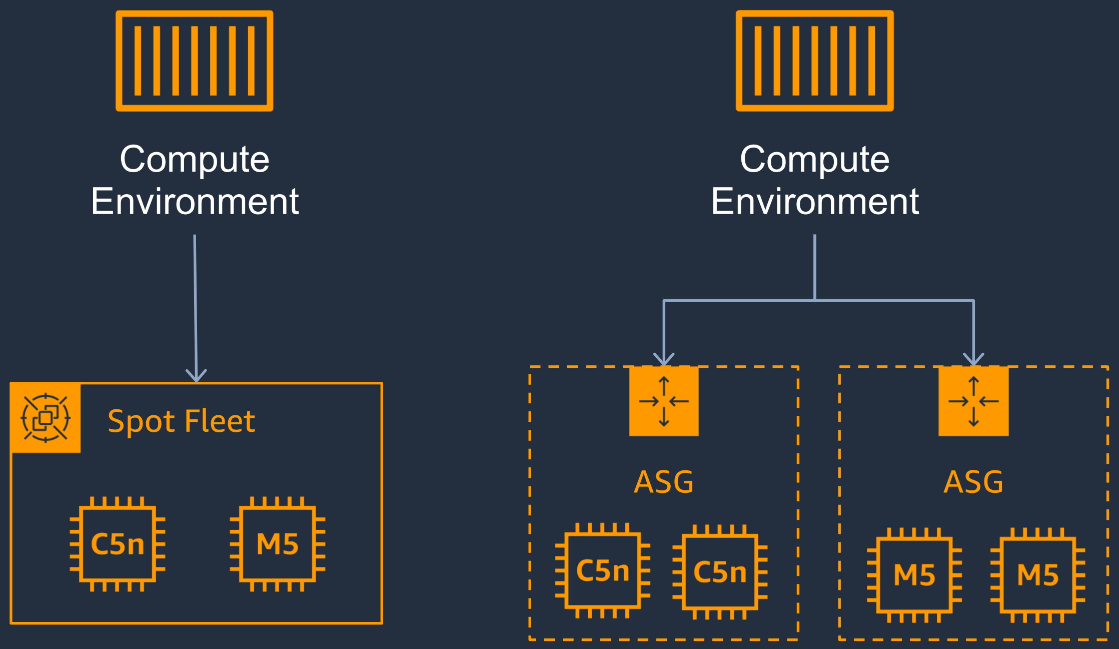
https://aws.amazon.com/blogs/compute/optimizing-for-cost-availability-and-throughput-by-selecting-your-aws-batch-allocation-strategy/

Allocation strategies underneath

BEST_FIT

Spot

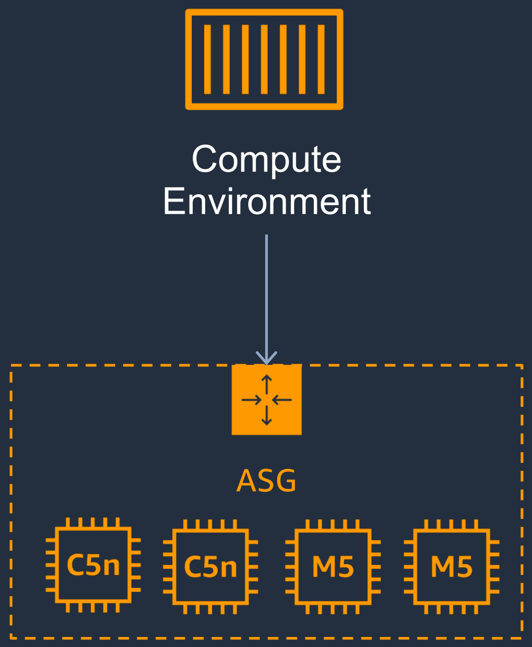
OnDemand



Each instance type has its own ASG

BEST_FIT_PROGRESSIVE

SPOT_CAPACITY_OPTIMIZED

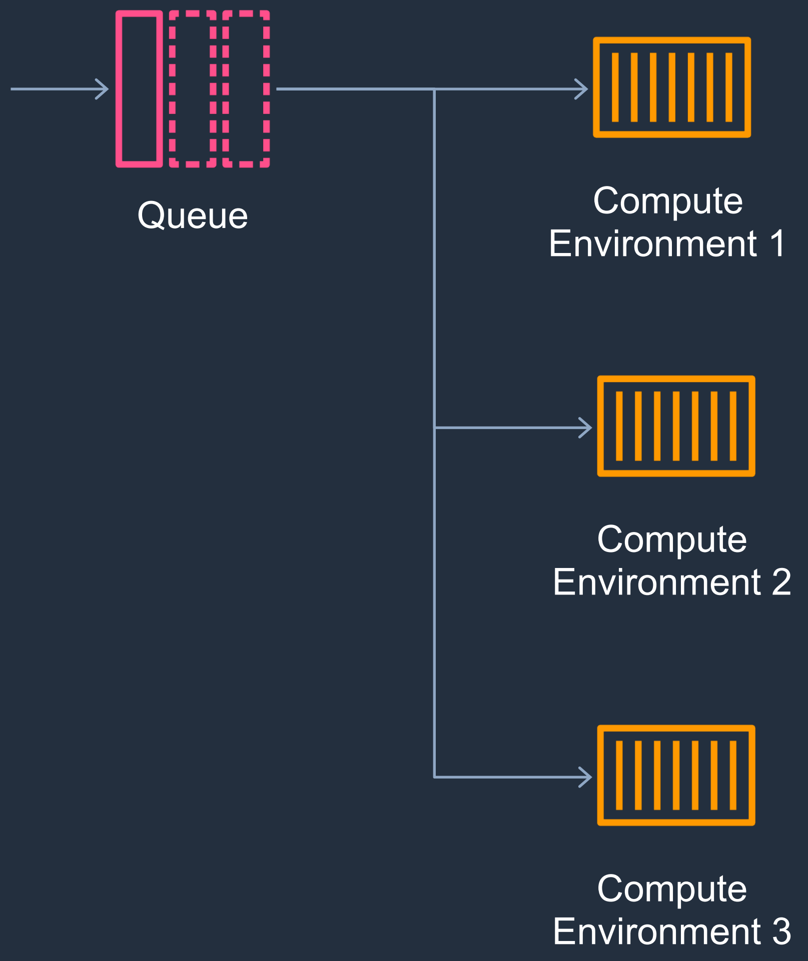


How batch scales CEs

- When is scaling triggered
  - The first time a job is submitted to a queue
  - Every 10 minutes
  - When a MNP job is submitted
  - User calls Create/Update/Delete CE
  - Backend action

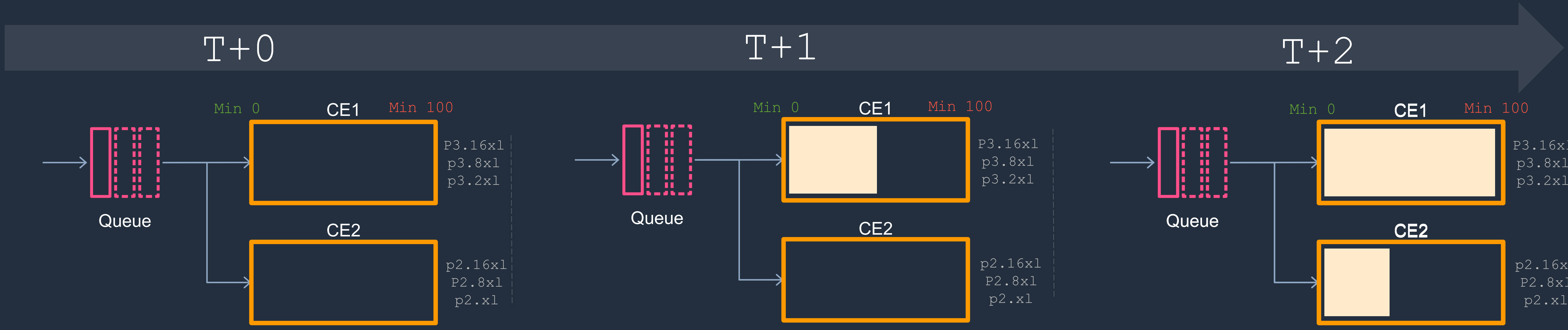
- How is scale up is conducted
  1. For each queue
  2. Consolidate view runnable jobs by properties
  3. Pack jobs in resources chunks to maximize vCPUs packing
  4. Select instances type(s) by lowest $ and vCPU/Memory/GPU packing, use larger instances if possible
  5. Provide list of instances ordered by $ to the ASG

- For scale down, AWS Batch explicitly terminate instances, not the ASG



CEs in-order scaling example

- CEs are scaled in-order by the JQ
- Switching to the next CE occurs when capacity is met in the actual CE



Jobs submitted, those in runnable stage are evaluated

CE1 is scale up until being fulfilled

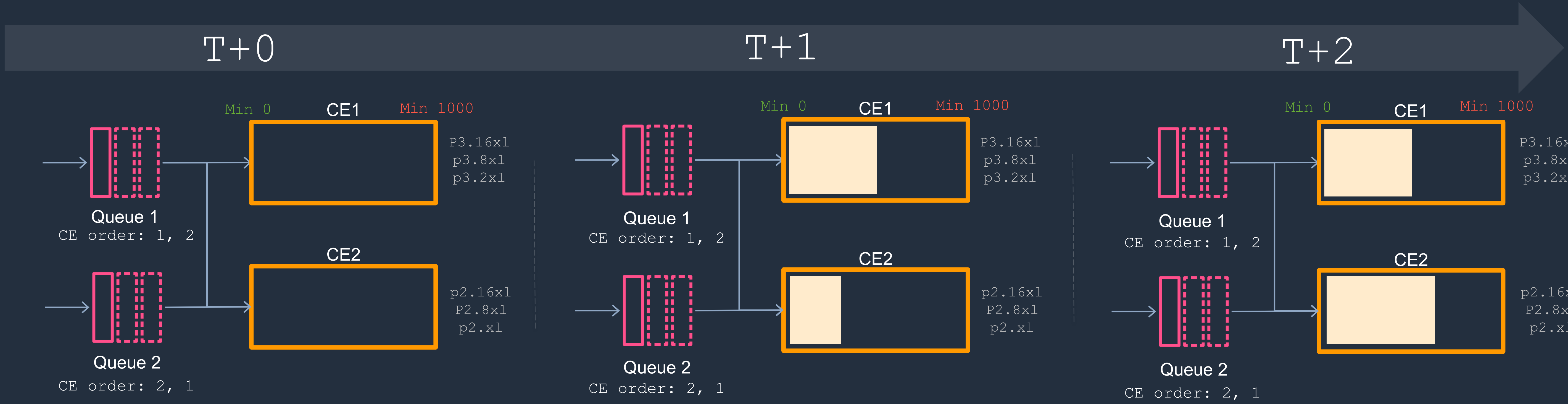
When CE1 is fulfilled, we spill up to CE2

No fulfillment of CE1, we spilling to CE2

What if our CE cannot be fulfilled?

CE / JQ Interleaving technique

- All JQs attached to each CEs (jobs can be schedules on each CE)
- JQ to CE order defined in a rolling fashion, each JQ scales a CE
- This technique helps for large capacity acquisition and fast scale-up



Jobs are distributed to both queues at the same time

Each queue scale the CEs in order, order varies by queue

Jobs can be scheduled on both CEs to be executed

Best Practices

Disabling Hyperthreading

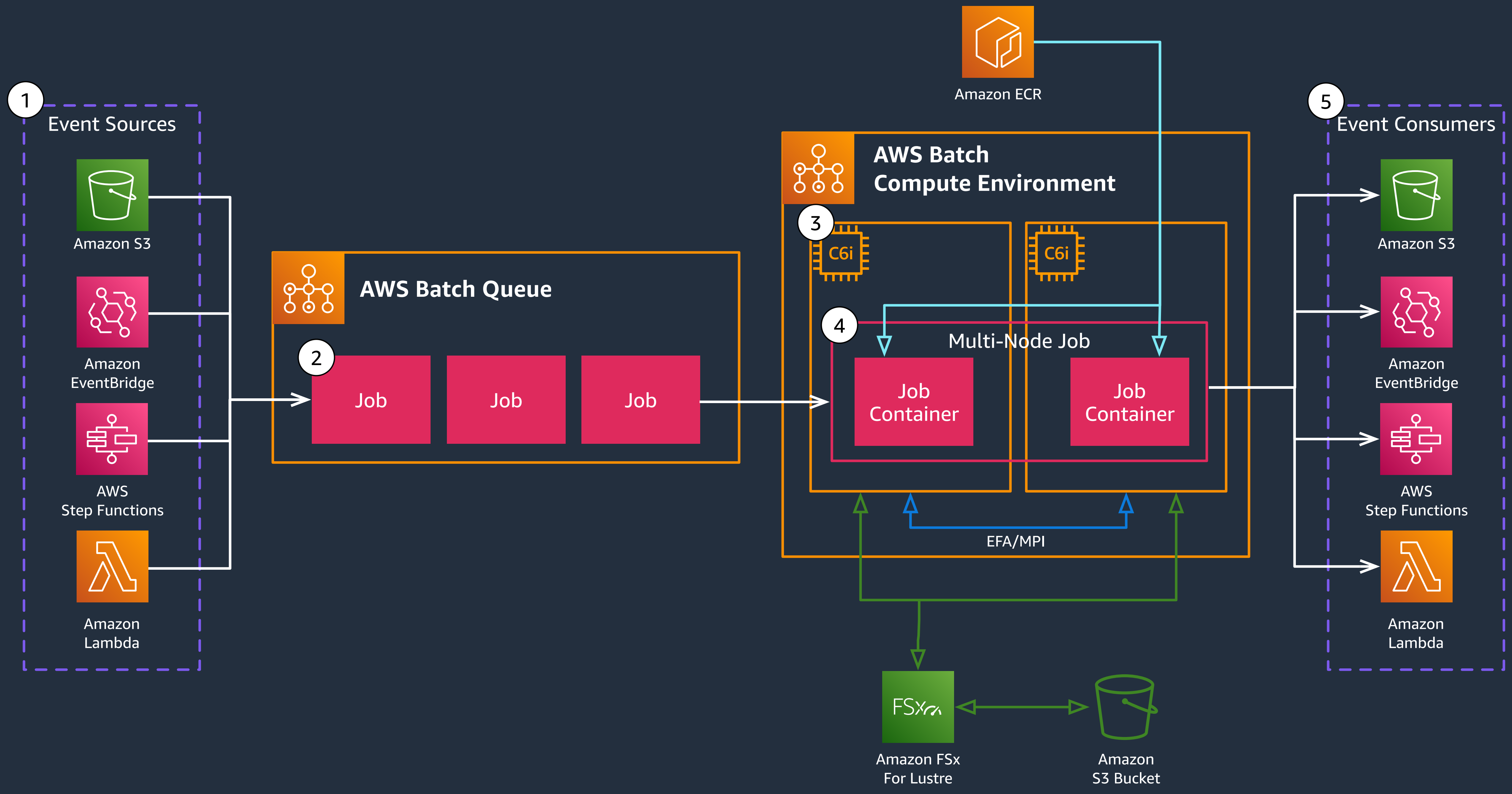
Many MPI applications will perform best with hyperthreading **disabled**. AWS Batch does not natively have the ability to disable hyperthreading on instances. Hyperthreading can be disabled by using a launch template or a modified AMI to disable SMT in Linux.

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="==DISABLE-HYPERTHREADING=="

--==DISABLE-HYPERTHREADING==
Content-Type: text/cloud-config; charset="us-ascii"

runcmd:
- echo off > /sys/devices/system/cpu/smt/control
```

Aws Batch Multi-Node Parallel



Intel MPI /dev/shm issue

By default, Intel MPI uses shared memory (/dev/shm) for local communication. Docker defaults to setting very low limits (64M) on /dev/shm for containers.

Batch supports setting the size of /dev/shm. As a general rule, 2GB/node-rank should be sufficient. For instance, if running 4 ranks on a node, set /dev/shm to 8GB. This may still be insufficient. If unexpected crashes still happen, try increasing that size. In a job definition, this looks like:

```
{
  "linuxParameters": {
    "sharedMemorySize": 8192
  }
}
```

AMI vs Docker Images

AMI’s are pulled by EC2 as a requirement before the instance starts, docker images are pulled by the ECS agent once the **instance is running.**

When a docker image becomes too big, it’s time to start creating a custom AMI, or putting software on a shared directory:

- System software, i.e. Lustre drivers, EFA, ect should be on the AMI
- Application software is typically kept on the docker image
- Programming environment i.e. /apps is usually kept on EFS/ZFS filesystem

Docker Image Optimization

- Update /etc/docker/daemon.json on the underlying AMI and bump up the threads that pull the container image down
    - { "max-concurrent-uploads": N, "max-concurrent-downloads": N }
  - Make sure the docker image layers are around the same size
    - https://github.com/wagoodman/dive
  - Thin down the image: https://github.com/docker-slim/docker-slim
  - Cache the image on the AMI so it can be used between jobs:
    - https://docs.aws.amazon.com/AmazonECS/latest/bestpracticesguide/pull-behavior.html

AWS Batch – Large Scale (>250k vcpus) Architecture

Things you may know

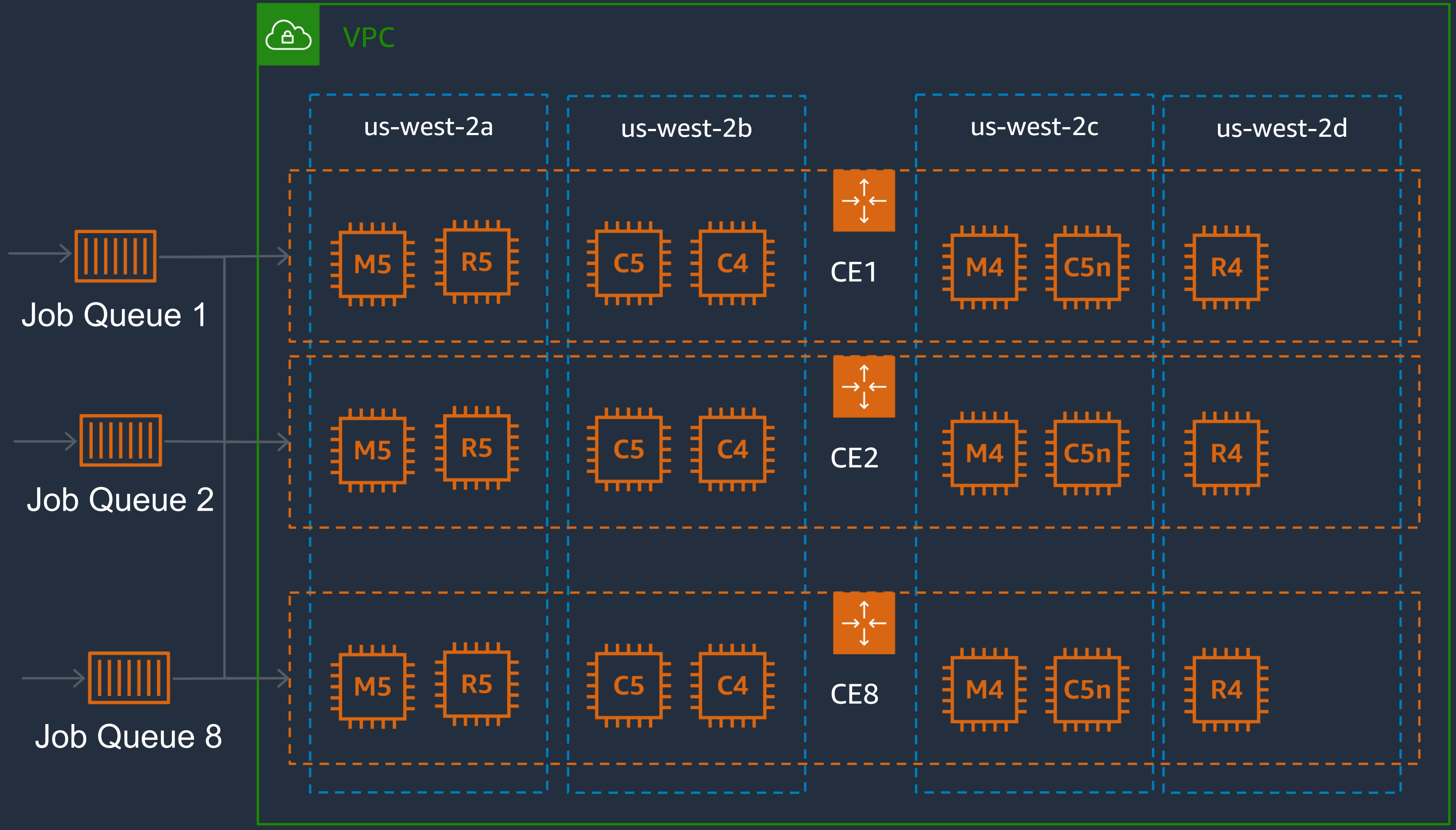
- A Job Queue can be attached to multiple CEs
- CEs will be scaled automatically by the JQ
- Jobs will be distributed across CEs

Things you wish you knew

- Batch will scale CEs/ASGs in order until capacity is met
- One solution is to interleave JQs / CEs

⇢ JQ1 linked to CE1, CE2, CE3, CE4…
JQ2 linked to CE2, CE3, CE4, CE1

⇢ each JQ will scale its first CE, jobs will be dispatched across CEs



AWS Batch – JQ / CE Interleaving

Things you may know

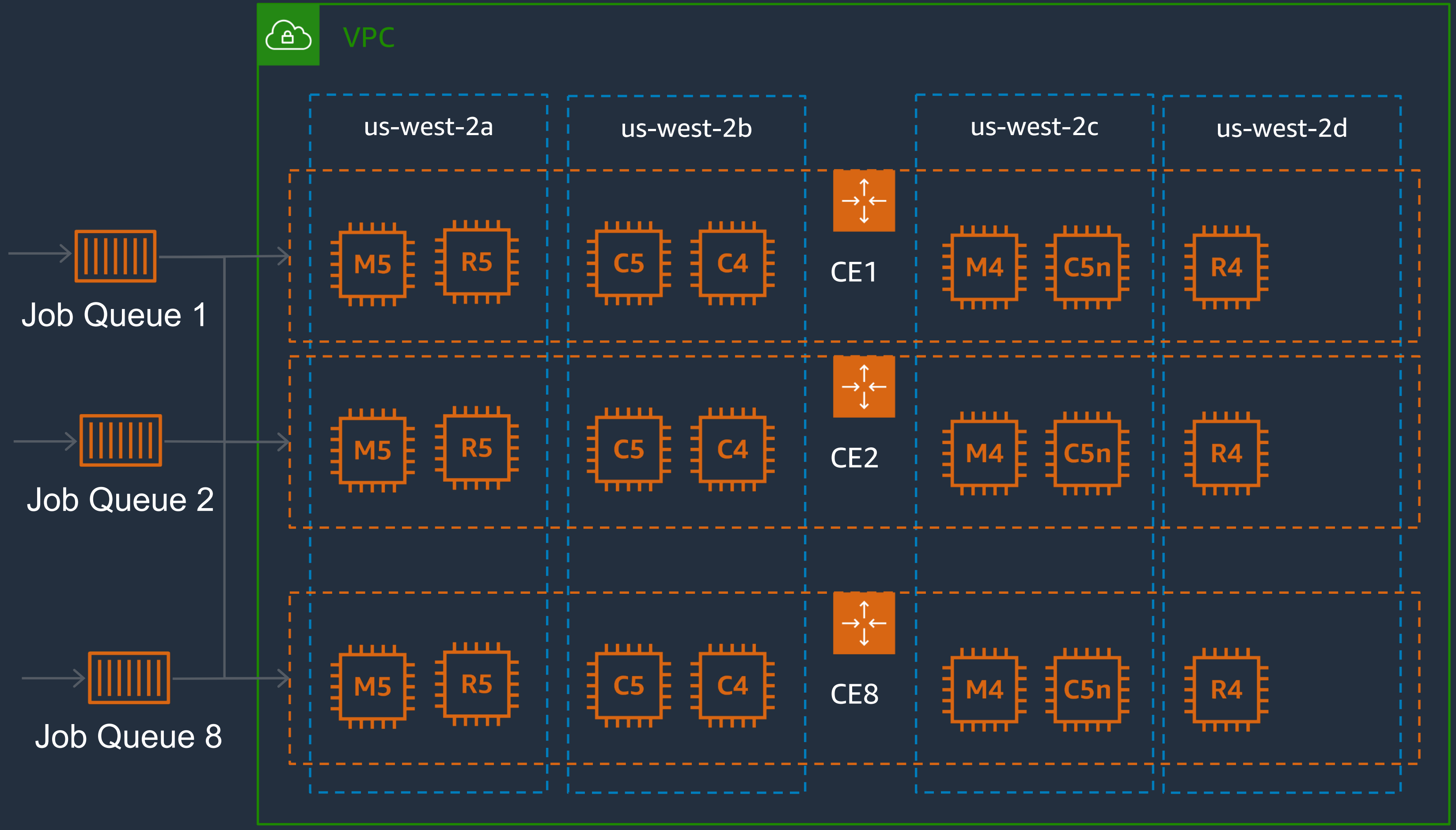
- A Job Queue can be attached to multiple CEs
- CEs will be scaled automatically by the JQ
- Jobs will be distributed across CEs

Things you wish you knew

- Batch will scale CEs/ASGs in order until capacity is met
- One solution is to interleave JQs / CEs

⇢ JQ1 linked to CE1, CE2, CE3, CE4…
JQ2 linked to CE2, CE3, CE4, CE1

⇢ each JQ will scale its first CE, jobs will be dispatched across CEs



AWS Batch – Simple Scaling Optimization

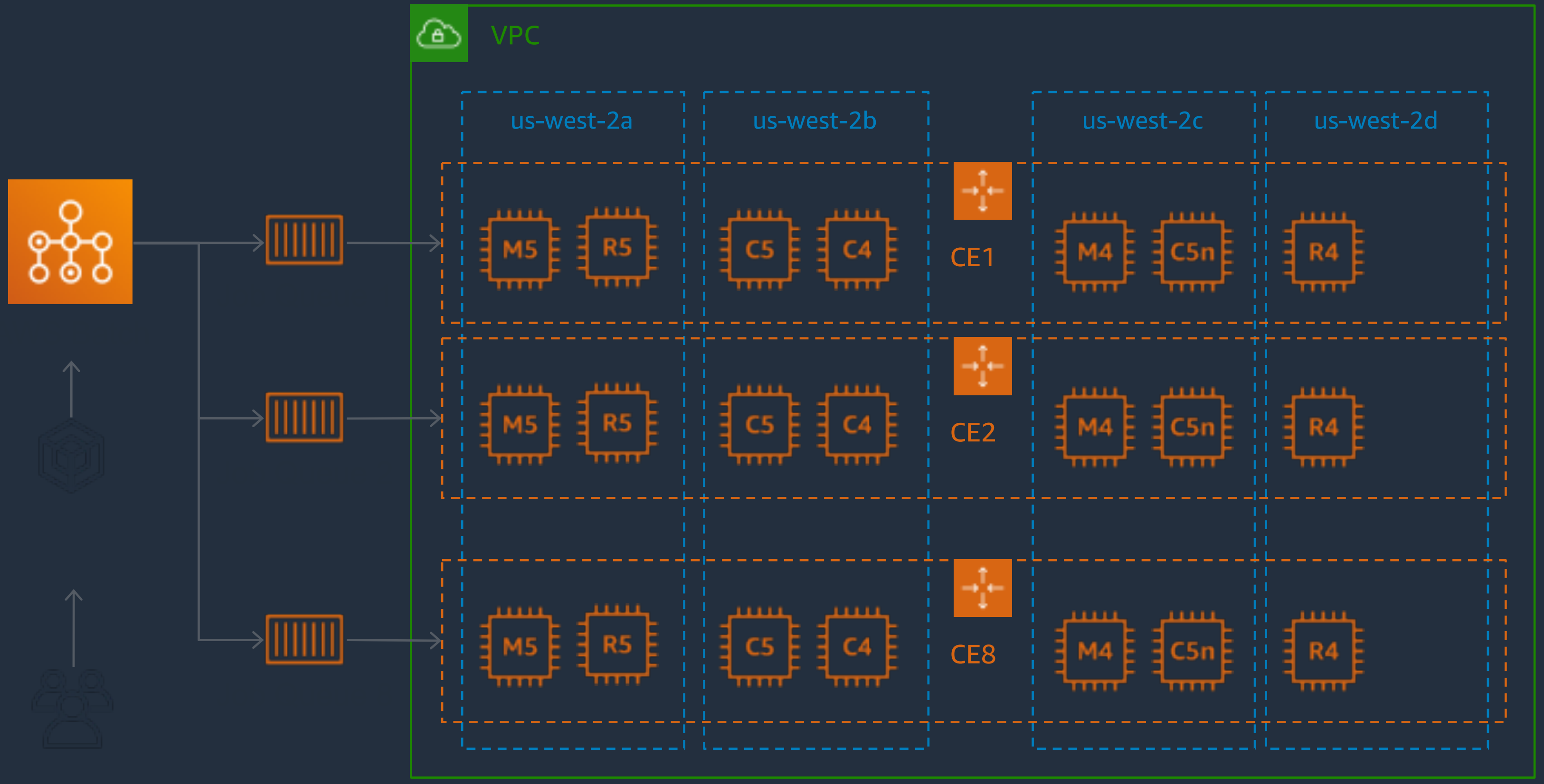
Limits to keep in mind

- ASG provides 1200 instances / 30 s *
- Batch Run Task is 150 TPS*
- Resource scale-up evaluated every 2 min*
- 3,000 Instances / CE

Summarized

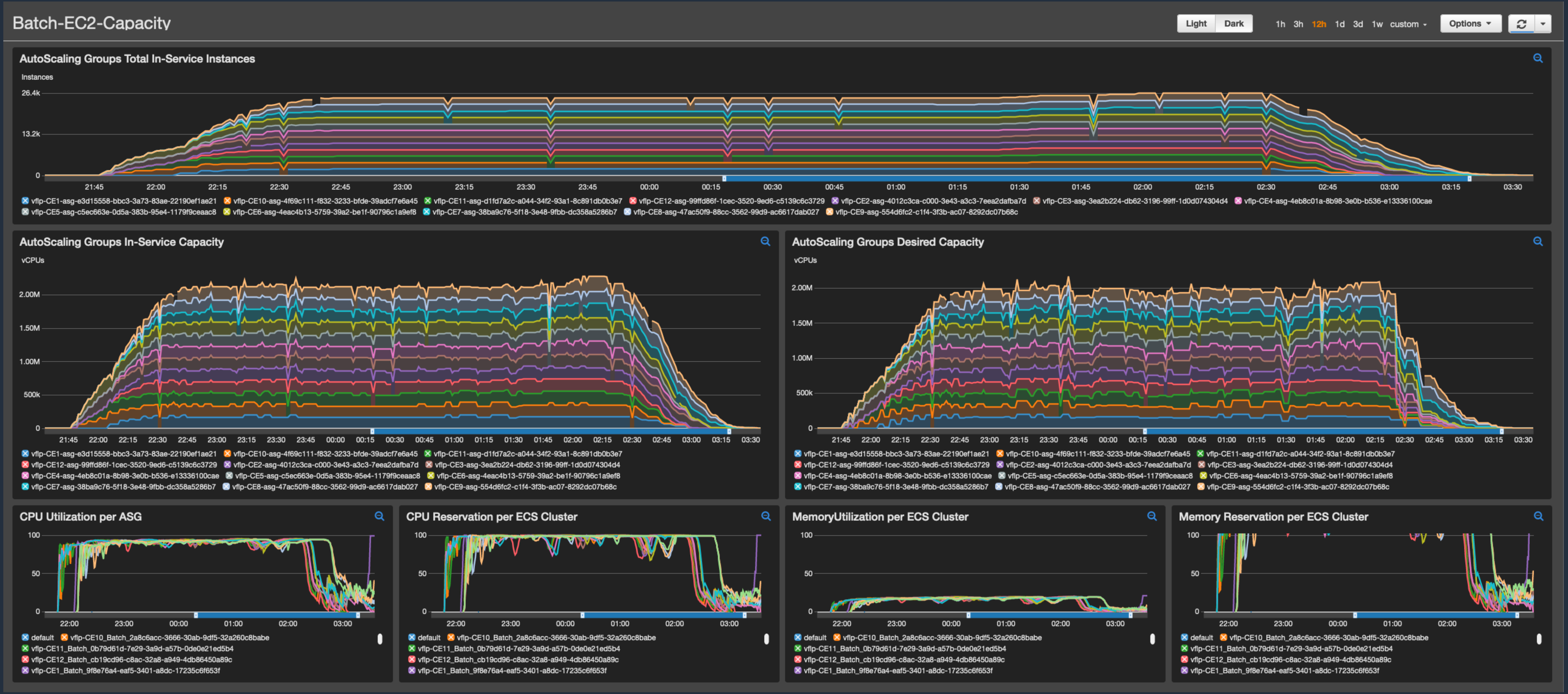
- 8 CEs / ASGs ⇢ 1600 instances / min
- Array jobs ⇢ up to 10,000 jobs / transaction
- Run Task ⇢ 9,000 jobs executed / min

Not fully balanced but ensures that scaling is initiated in <2 min



* This is on a per-account level, each CE is limited to 80 TPS

Metrics - Create CloudWatch Dashboard


Batch-EC2-Capacity
Light
Dark
1h
3h
12h
1d
3d
1w
custom
Options
AutoScaling Groups Total In-Service Instances
Instances
26.4k
13.2k
0
21:45
22:00
22:15
22:30
22:45
23:00
23:15
23:30
23:45
00:00
00:15
00:30
00:45
01:00
01:15
01:30
01:45
02:00
02:15
02:30
02:45
03:00
03:15
03:30
vflp-CE1-asg-e3d15558-bbc3-3a73-83ae-22190ef1ae21
vflp-CE10-asg-4f69c111-f832-3233-bfde-39adcf7e6a45
vflp-CE11-asg-d1fd7a2c-a044-34f2-93a1-8c891db0b3e7
vflp-CE12-asg-99ffd86f-1cec-3520-9ed6-c5139c6c3729
vflp-CE2-asg-4012c3ca-c000-3e43-a3c3-7eea2dafba7d
vflp-CE3-asg-3ea2b224-db62-3196-99ff-1d0d074304d4
vflp-CE4-asg-4eb8c01a-8b98-3e0b-b536-e13336100cae
vflp-CE5-asg-c5ec663e-0d5a-383b-95e4-1179f9ceaac8
vflp-CE6-asg-4eac4b13-5759-39a2-be1f-90796c1a9ef8
vflp-CE7-asg-38ba9c76-5f18-3e48-9fbb-dc358a5286b7
vflp-CE8-asg-47ac50f9-88cc-3562-99d9-ac6617dab027
vflp-CE9-asg-554d6fc2-c1f4-3f3b-ac07-8292dc07b68c
AutoScaling Groups In-Service Capacity
vCPUs
2.00M
1.50M
1.00M
500k
0
AutoScaling Groups Desired Capacity
vCPUs
2.00M
1.50M
1.00M
500k
0
CPU Utilization per ASG
100
50
0
22:00
23:00
00:00
01:00
02:00
03:00
default
vflp-CE10_Batch_2a8c6acc-3666-30ab-9df5-32a260c8babe
vflp-CE11_Batch_0b79d61d-7e29-3a9d-a57b-0de0e21ed5b4
vflp-CE12_Batch_cb19cd96-c8ac-32a8-a949-4db86450a89c
vflp-CE1_Batch_9f8e76a4-eaf5-3401-a8dc-17235c6f653f
CPU Reservation per ECS Cluster
MemoryUtilization per ECS Cluster
Memory Reservation per ECS Cluster

https://github.com/aws-samples/aws-batch-runtime-monitoring

Thank You