
AWS
re:Invent
NOV. 28 – DEC. 2, 2022 | LAS VEGAS, NV

CON319

Running large, compute-intensive workloads on Amazon EKS with AWS Batch

Angel Pizarro

Principal Developer Advocate
AWS

Jason Rupard

Principal Software Development Engineer
AWS

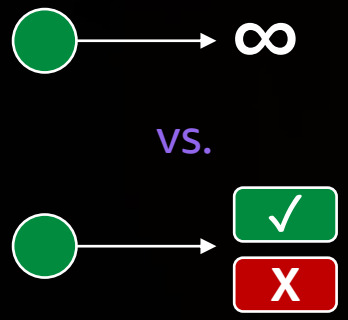
Kubernetes excels at managing microservices

Batch workloads, like ML training, differ from microservices in important ways

Stable vs. finite

Microservices are assumed to not stop once started.

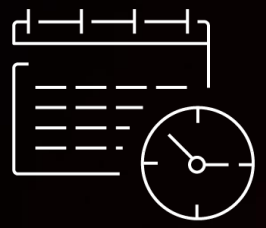
Batch workloads by definition run to completion (succeed or fail).



Response times

Microservices expect millisecond response times.

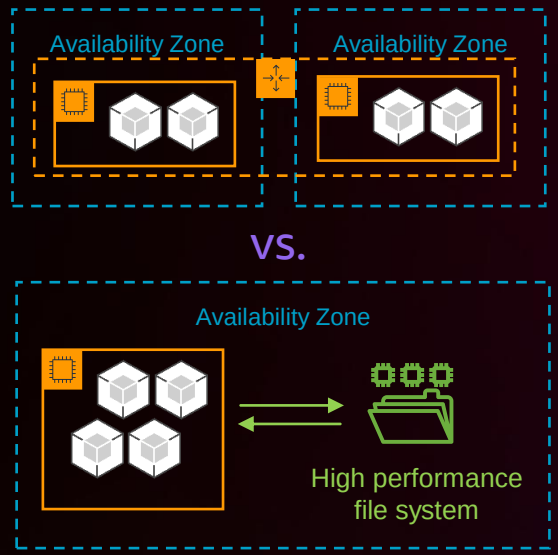
Batch results return of result expectations are at least minutes, sometimes days.



Replicated vs. not

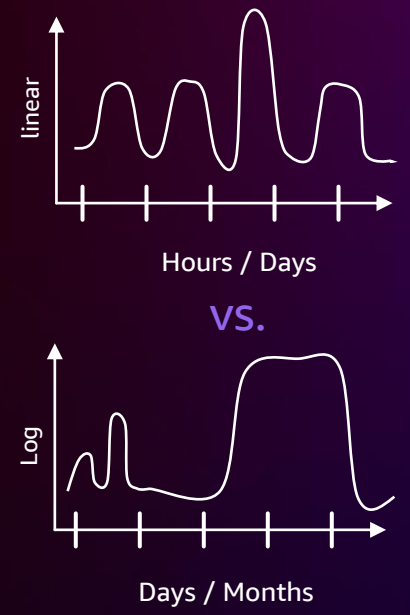
High-availability is not a thing for batch.

Fast access to Zone-level resources



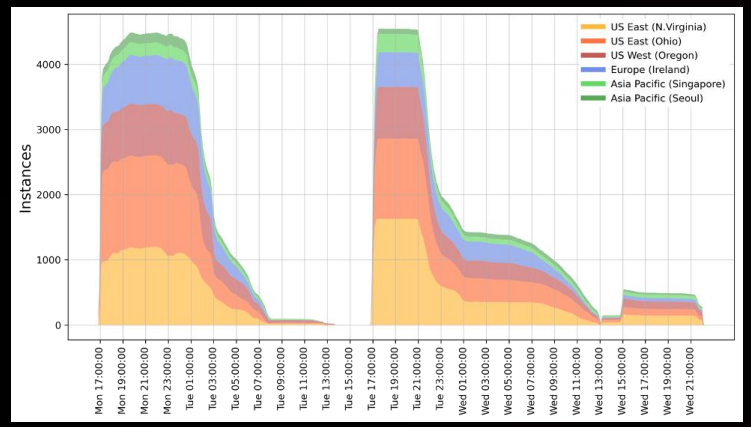
Cyclical vs. Episodic

Both time and capacity scales diverge



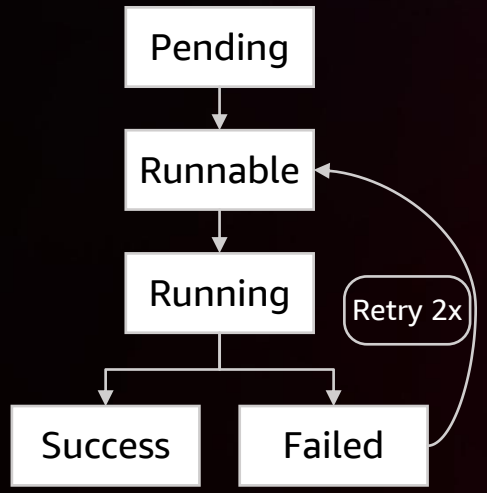
Additional batch workload requirements

Extreme scaling!



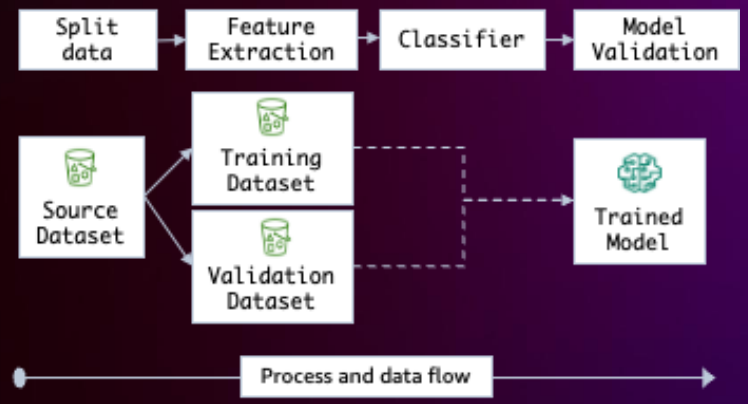
Launch 4500+ instances for a few hours, then scale to zero for months.

Job lifecycle, retry logic



Retry logic can be modified by the exit status. E.g. only retry in the case of Spot reclaim

Job dependencies



Dependency across jobs can be defined *a priori* or at runtime.

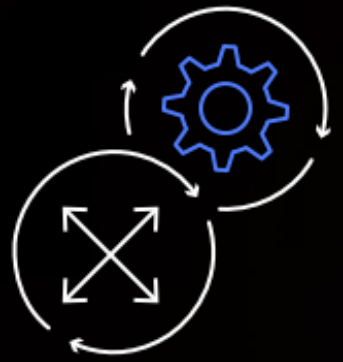
If there is a failure early, *it makes no sense to keep going.*

Solutions for batch computing on Kubernetes

- Several open source frameworks exists for batch computing
  - Apache YuniKorn, Volcano, Kueue
- Ability to roll-your-own solution with general purpose scheduler + proper batch-style node scaling
  - Karpenter, Hashicorp Nomad
  - Leverage the Kubernetes Jobs API

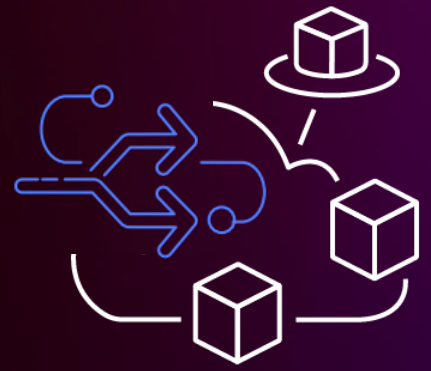
All of these solutions strive to meet the challenges of batch workloads on Kubernetes, but …

We heard from customers that running high-scale, compute-intensive workloads on Kubernetes is challenging



Running and scaling batch workloads on Kubernetes is difficult and requires significant investment

Operational and cost optimizations are different for batch workloads

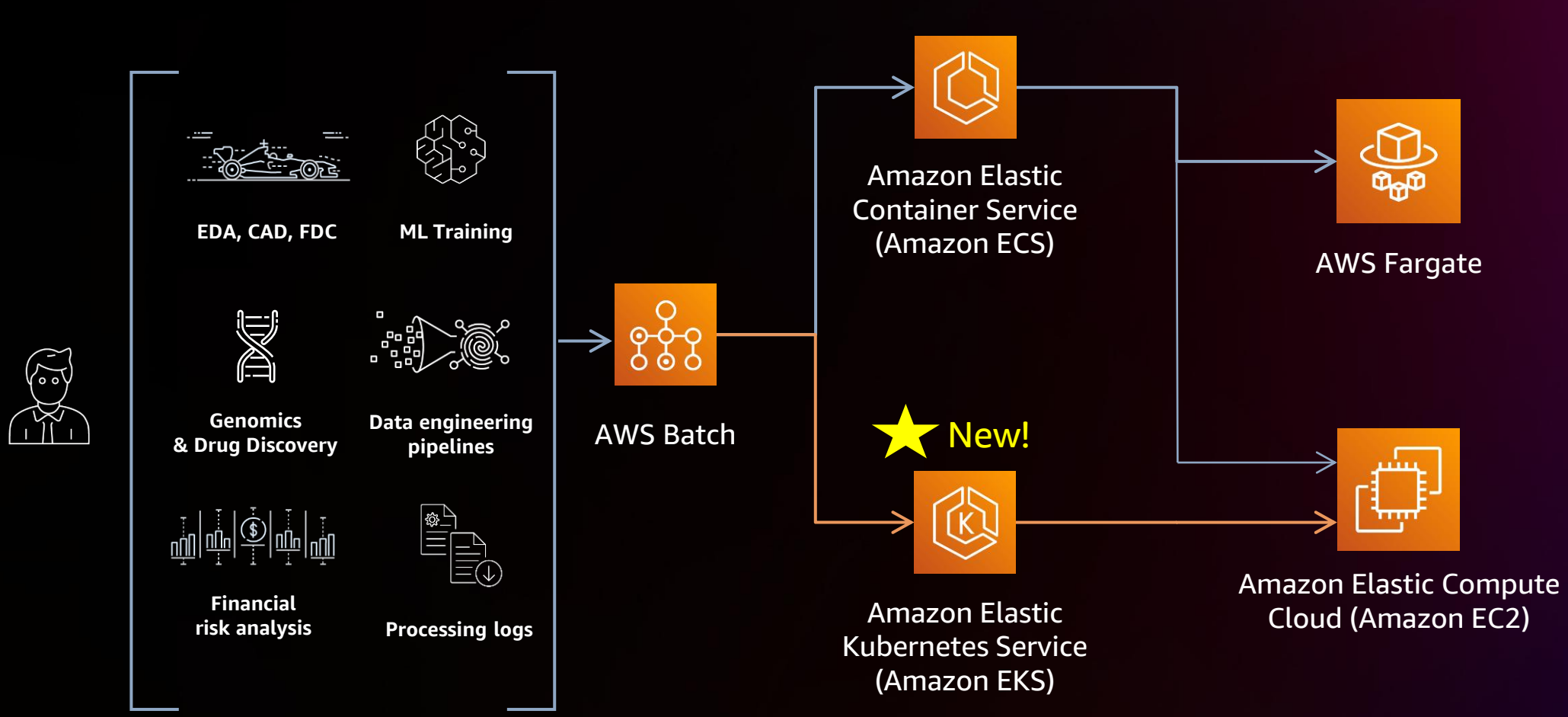


Scheduling pods quickly and efficiently is a challenge for the k8s native scheduler

All of this extra work was leading to a lot of undifferentiated heavy lifting.

AWS Batch

FULLY-MANAGED BATCH SCHEDULER FOR CONTAINERIZED WORKLOADS



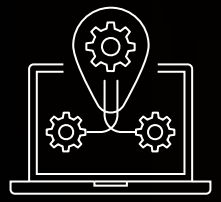
What is AWS Batch?

Job scheduler

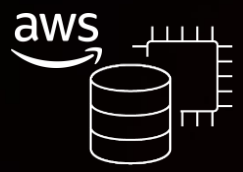
- Schedules and runs jobs asynchronously
- Manages dependencies

Resource orchestrator

- Manages and optimizes compute resources
- Scales up/down as needed
- Utilizes the right compute resources for the job



Fully managed



Integrated with AWS services

Massive scalability

Optimized resource provisioning

Cost-efficient

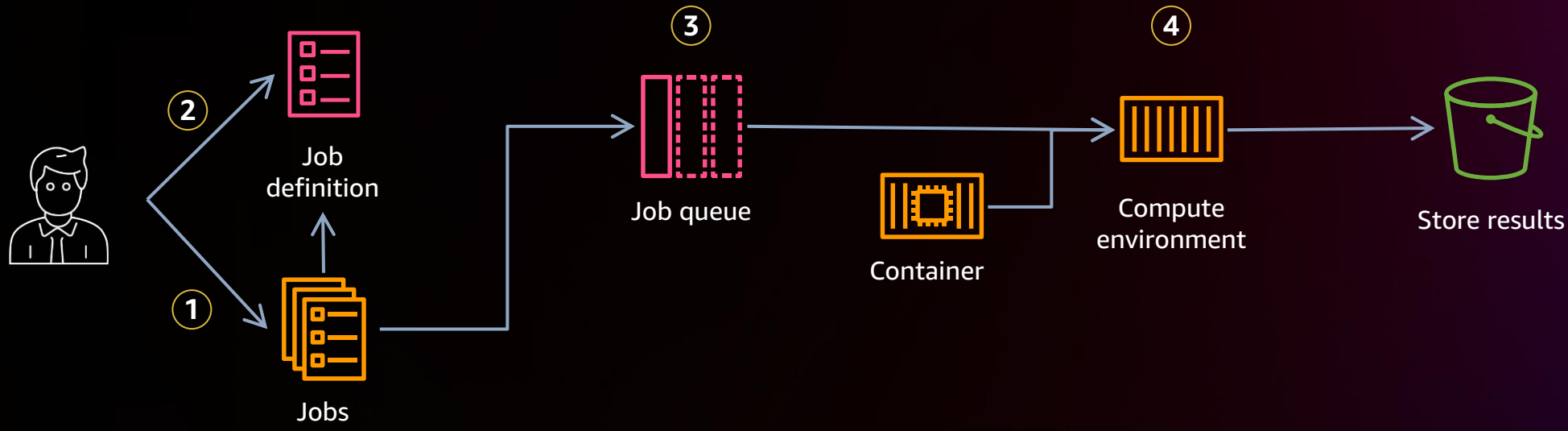
AWS Batch overview

1. **Job**
   Actual work request. Each job must reference a job definition, but many parameters may be overridden when submitted

2. **Job definition**
   Template that has common attributes (such as container image, IAM role, vCPU and memory requirements)

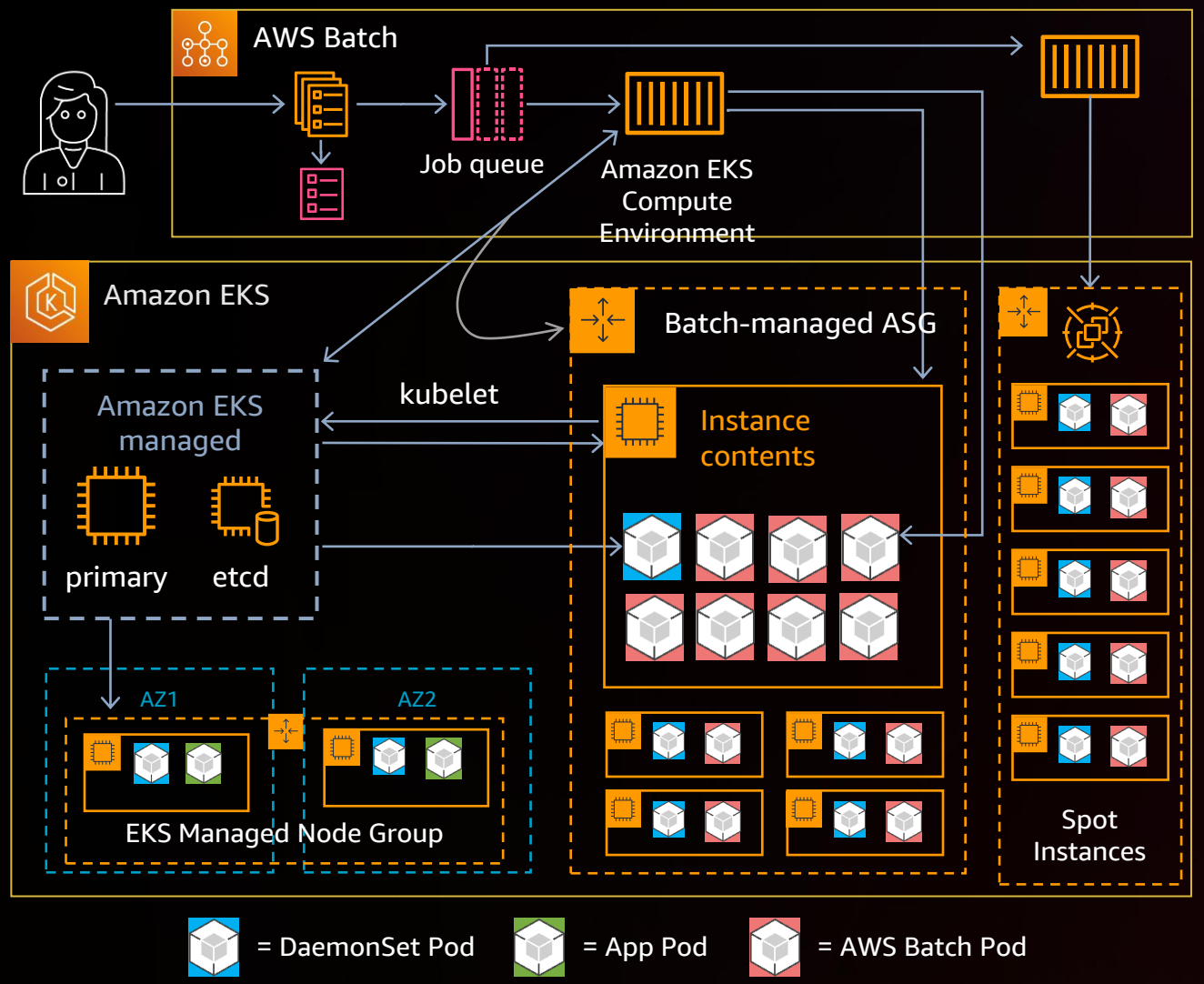
3. **Job queue (JQ)**
   Queue determines priorities. Each JQ is connected to one or more CE

4. **Compute environment (CE)**
   Resource mix (defines on-demand vs. Spot and other instance types); CE can be connected to more than one JQ



Diving into the details

AWS Batch for Amazon EKS – Under the hood



- Batch is the main entry point for batch workload requests
- Batch launches worker nodes based on the jobs queued
- Batch nodes are separate from other EKS node groups
  - Using Batch's scaling knowledge for capacity pools and Spot interruption likelihood
  - Taints prevent placement of other pods on Batch-managed nodes
- Batch handles the pod placements via nodename
- Multiple Compute Environments per cluster

Amazon EKS specific API parameters

- Amazon EKS compute environments and job definitions have **separate parameters from Amazon ECS**
- AWS Batch supports EKS versions **1.21+, but 1.22+ is recommended**
- You can scope a DaemonSet to Batch nodes using **tolerations**

```
{
    "jobDefinitionName": "",
    "type": "container",
    "eksProperties": {
        "podProperties": {
            "serviceAccountName": "myBatchEksServiceAccount",
            "hostNetwork": true,
            "containers": [
                {
                    "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
                    "command": [
                        "echo",
                        "Hello KubeCon!"
                    ],
                    "resources": {
```

```
tolerations:
- key: "batch.amazonaws.com/batch-node"
  operator: "Exists"
```

```
                    "securityContext": {}
                }
            ],
            "volumes": []
        }
    }
    "retryStrategy": {},
    "propagateTags": true,
    "timeout": {
        "attemptDurationSeconds": 0
    },
    "tags": {
        "KeyName": ""
    },
    "schedulingPriority": 0,
}
```

vCPU and memory considerations for EKS jobs

- Batch on EKS supports setting requests and **limits** configuration in container resources for jobs
- Batch also has a unique set of constraints for `cpu` and `memory` specifications
  - `cpu` can be specified only in whole cpu values (ie., 2) or in fraction forms
  - When cpu is specified in fraction forms, it must be in increments of 0.25
  - cpu cannot be specified in millCPU form, i.e. 100m
  - cpu must be >= 0.25 (the smallest cpu size of a job)
- `memory` can only be specified in Mebibytes (Mi) form
- When Batch translates an EKS Job into a pod, it sets `request` equal to `limits`

```
{
    "jobDefinitionName": "MyJobOnEks_Sleep",
    "type": "container",
    "eksProperties": {
        "podProperties": {
            "containers": [
                {
                    "image": "public.ecr.aws/amazonlinux/amazonlinux:2",
                    "command": ["sleep", "60"],
                    "resources": {
                        "limits": {
                            "cpu": "1",
                            "memory": "1024Mi"
                        }
                    }
                }
            ]
        }
    }
}
```

```
---
apiVersion: v1
kind: Pod
...
spec:
  ...
  containers:
    - command:
        - sleep
        - 60
      image: public.ecr.aws/amazonlinux/amazonlinux:2
      resources:
        limits:
          cpu: 1
          memory: 1024Mi
        requests:
          cpu: 1
          memory: 1024Mi
      ...
```

GPU workloads

- Batch supports P2, P3, P4, G3 and G4 instance families
- By default, Batch uses the Amazon EKS Accelerated AMI with Kubernetes version matching your Amazon EKS cluster control-plane version
- Batch **does not** manage the NVIDIA GPU device plugin on your behalf. You must install this plugin into your Amazon EKS cluster as a DaemonSet!

```
# pull nvidia daemonset spec
curl -O "https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v0.12.2
  /nvidia-device-plugin.yml"

# Add these tolrations to your config
tolerations:
  - key: "batch.amazonaws.com/batch-node"
    operator: "Exists"
    effect: "NoSchedule"
  - key: "batch.amazonaws.com/batch-node"
    operator: "Exists"
    effect: "NoExecute"

# apply the changes
kubectl apply -f nvidia-device-plugin.yml
```

```
{
    "jobDefinitionName": "MyGPUJobOnEks_Smi",
    "type": "container",
    "eksProperties": {
        "podProperties": {
            "hostNetwork": true,
            "containers": [
                {
                    "image": "nvcr.io/nvidia/cuda:10.2-runtime-centos7",
                    "command": ["nvidia-smi"],
                    "resources": {
                        "limits": {
                            "cpu": "1",
                            "memory": "1024Mi",
                            "nvidia.com/gpu": "1"
                        }
                    }
                }
            ]
```

Shared responsibility – The infrastructure edition

This is your cluster! AWS Batch is a good tenant, but will need some permissions defined.

OIDC, RBAC, namespace, AWS Identity and Access Management (IAM) service account identity mapping

Register the Amazon EKS cluster with AWS Batch as a CE

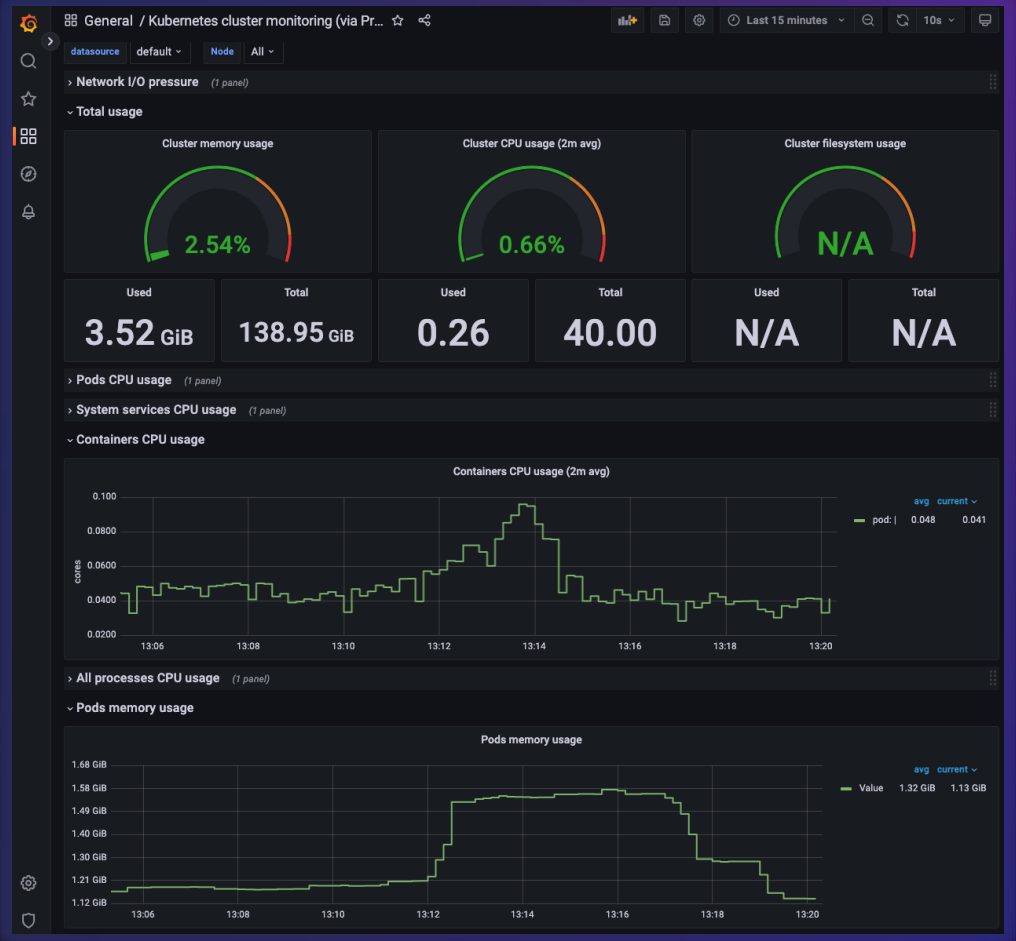
CE validate cluster for the proper permissions and version

If the check fails, the CE is marked as INVALID

Why leverage AWS Batch for workloads on Amazon EKS?

- Managed services **reduce your operational and optimization overhead.** Lets you focus on business requirements
  - Benefit from improved operations and cost optimizations as they are developed.
- **Advanced batch features** like Fair-Share scheduling, job dependencies, and compute allocation strategies
- **Integration** with other AWS services (e.g. IAM, Amazon EventBridge, AWS Step Functions)
- Take advantage of the **Kubernetes ecosystem** of partners and tools

Monitoring and visualization with Prometheus and Grafana



Thank you!

Angel Pizarro
Email: pizarroa@amazon.com
Twitter: @delagoya

Jason Rupard
Email: rupard@amazon.com

Please complete the session survey in the **mobile app**