AWS re:Invent

NOV. 29 – DEC. 3, 2021 | LAS VEGAS, NV

ANT322

Amazon EMR on EKS

Vincent Gromakowski
Principal SA, Analytics
AWS

Agenda

- Overview of Amazon EMR
- Overview of Amazon EMR on EKS
- Walkthrough of how a job runs
- Advanced Amazon EMR on EKS usage
- Discussion ☺

aws

Amazon EMR

Easily run Spark, Hive, Presto, HBase, Flink, and more big data apps on AWS

Latest versions

Updated with latest open-source frameworks within 30 days

Support for popular OSS like Flink, Hudi

Fast performance at low cost

Spark workloads run 2.4x faster compared to open source

50%–80% reduction in costs with Amazon EC2 Spot and Reserved Instances

Per-second billing for flexibility

Use Amazon S3 storage

Process data in Amazon S3 securely with high performance using the EMRFS connector

Scale compute and storage independent of each other

Easy and scalable

Fully managed; no cluster setup, node provisioning, or cluster tuning

Vertical and horizontal Auto Scaling to suit workload demands

aws

Amazon EMR on EKS

Run Apache Spark jobs on demand on **Amazon Elastic Kubernetes Service (EKS)** – without provisioning EMR clusters – to improve resource utilization and simplify infrastructure management

How Amazon EMR on EKS solves the problem

DevOps team

- Auto scaling considerations
- Patching and upgrading
- Integration with the rest of the platform

Amazon EMR on EKS as a managed service is integrated with AWS natively

Data engineers/ data scientists

- Performance optimization
- Cold startup time
- Data isolation

Amazon EMR on EKS provides 2.4x better performance; k8s multi-tenancy design

C-level

- Cost optimization
- Security
- Access control

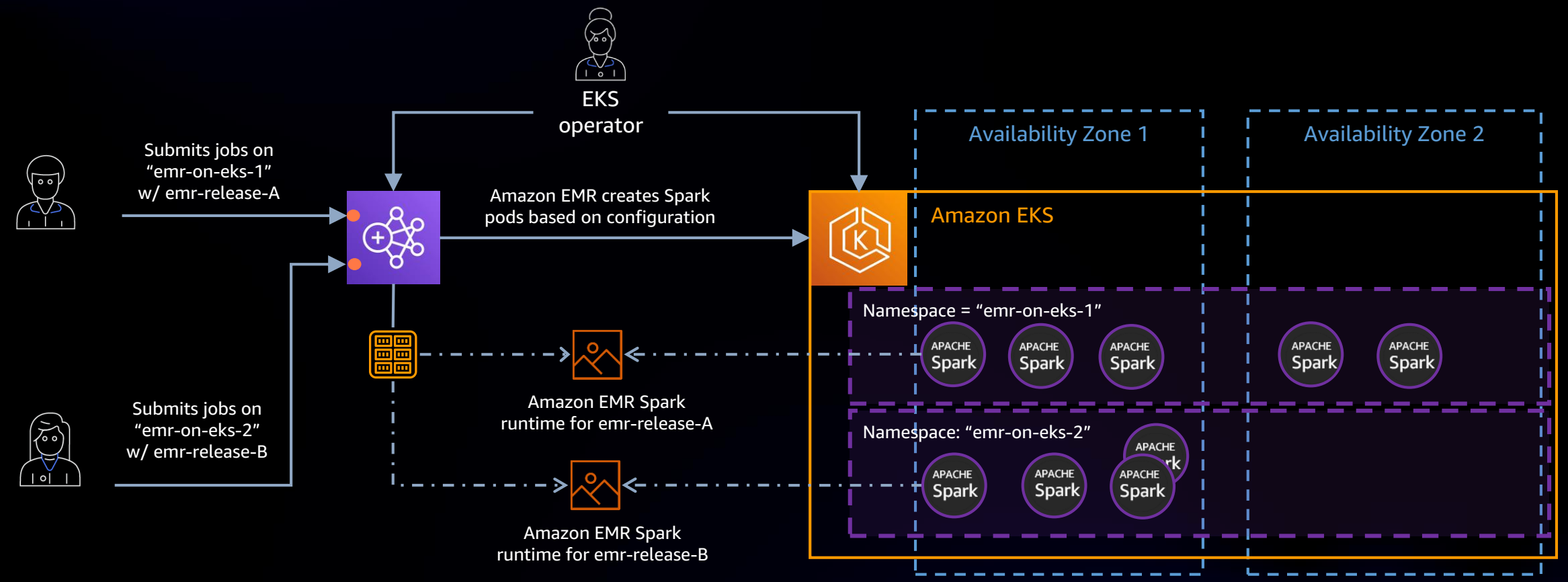
Amazon EMR on EKS charges per job and achieves higher resource utilization with other microservices

aws

Amazon EMR helps accelerate move to EKS

- Provide managed distribution Spark on Kubernetes (2.4 and 3.1)
- Manage job execution on your behalf
- Simplify secure execution using granular access control
- Native integration with Amazon S3, AWS Glue Data Catalog, and more
- Simplify debugging with Spark History Server
- Support integration with Apache Airflow
- Differentiated performance with Amazon EMR runtime for Apache Spark

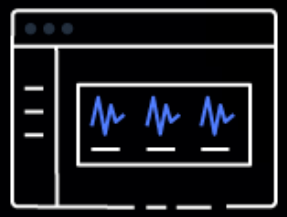
Job-centric workflow



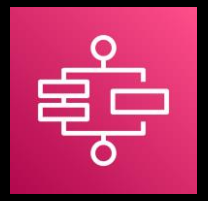
Job submission options

AWS Command Line Interface (AWS CLI)

AWS Tools and AWS SDK



Apache Airflow

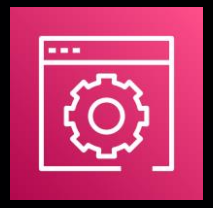


AWS Step Functions

Job debugging options

Amazon CloudWatch

Amazon Simple Storage Service (Amazon S3)



AWS Management Console

Amazon EMR Studio

How to run a Spark job on Amazon EMR on EKS

```
aws emr-containers start-job-run \
    --virtual-cluster-id cluster_id \
    --name sample-job-name \
    --execution-role-arn execution-role-arn \
    --release-label emr-6.3.0-latest \
    --job-driver '{
        "sparkSubmitJobDriver": {
            "entryPoint": "local:///usr/lib/spark/examples/src/main/python/pi.py",
            "sparkSubmitParameters": "--conf spark.executor.instances=2 --conf spark.executor.memory=2G --conf
spark.executor.cores=2 --conf spark.driver.cores=1"
        }
    }'
```

Pod templates

- Schedule Spark executors to run on Amazon EC2 Spot Instances
- Run a separate “sidecar” container next to the Spark driver or executor – logging, additional monitoring
- Run an “init” container that prepares the environment, e.g., downloading jars

```yaml
apiVersion: v1
kind: Pod
spec:
  volumes:
    - name: source-data-volume
      emptyDir: {}
    - name: metrics-files-volume
      emptyDir: {}
  nodeSelector:
    eks.amazonaws.com/nodegroup: emr-containers-nodegroup
  containers:
  - name: custom-side-car-container # Sidecar container
    image: <side_car_container_image>
    env:
      - name: RANDOM_SIDECAR
        value: random
    volumeMounts:
      - name: metrics-files-volume
        mountPath: /var/metrics/data
    command:
      - /bin/sh
      - '-c'
      -  <command-to-upload-metrics-files>
  initContainers:
  - name: spark-init-container-driver # Init container
    image: <spark-pre-step-image>
    volumeMounts:
      - name: source-data-volume # Use EMR predefined volumes
        mountPath: /var/data
    command:
      - /bin/sh
      - '-c'
      -  <command-to-download-dependency-jars>
```

Custom containers

- Install and configure packages specific to your workload
- Set environment variables
- Incorporate data pipeline into CI/CD

```
FROM 895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.3.0-latest

USER root

# Install Chrome
RUN curl https://intoli.com/install-google-chrome.sh \
    | bash && \
    mv /usr/bin/google-chrome-stable /usr/bin/chrome

# Install bokeh and sampledata
RUN pip3 install \
    bokeh>=2.3.2 \
    chromedriver-py>=91.0.4472.19.0 \
    selenium>=3.141.0
RUN bokeh sampledata

USER hadoop:hadoop
```

Resources

EMR Containers Best Practices Guide
https://aws.github.io/aws-emr-containers-best-practices/

Amazon EMR on Amazon EKS Demos
https://youtube.com/playlist?list=PLUe6KRx8LhLpJ8CyNHewFYukWm7sQyQrM

EMR on EKS Workshop (Advanced)
https://www.eksworkshop.com/advanced/430_emr_on_eks/

EMR on EKS Workshop
https://emr-on-eks.workshop.aws/

GitHub: AWS CDK Example
https://github.com/aws-samples/aws-cdk-for-emr-on-eks

GitHub: Custom Image Validation Tool
https://github.com/awslabs/amazon-emr-on-eks-custom-image-cli

aws

Workload isolation

- Spark workloads are IO intensive (disk and network) but Kubernetes/Amazon EKS doesn’t provide IO isolation

- How to isolate Amazon EMR workloads?
- Label nodes dedicated to Amazon EMR
- Use node selectors with these labels
- Use Taints and Tolerations to explicitely reject other workloads

aws

Workload isolation code example

Eksctl nodegroup definition

```yaml
- name: spot-4xl-arm-nvme-1b
  availabilityZones: ["us-east-1b"]
  minSize: 0
  maxSize: 4
  privateNetworking: true
  instancesDistribution:
    instanceTypes: ["r6gd.4xlarge"]
    onDemandBaseCapacity: 0
    onDemandPercentageAboveBaseCapacity: 0
    spotAllocationStrategy: capacity-optimized
  labels:
    arch: arm
    disk: nvme
    noderole: spark
  tags:
    k8s.io/cluster-autoscaler/node-template/label/arch: arm
    k8s.io/cluster-autoscaler/node-template/label/kubernetes.io/os: linux
    k8s.io/cluster-autoscaler/node-template/label/noderole: spark
    k8s.io/cluster-autoscaler/node-template/label/disk: nvme
    k8s.io/cluster-autoscaler/node-template/label/node-lifecycle: spot
    k8s.io/cluster-autoscaler/node-template/taint/spot: "true:NoSchedule"
    k8s.io/cluster-autoscaler/node-template/label/topology.kubernetes.io/zone: us-east-1b
    k8s.io/cluster-autoscaler/experiments: owned
    k8s.io/cluster-autoscaler/enabled: "true"
  taints:
    noderole: "spark:NoSchedule"
  iam:
    withAddonPolicies:
      ebs: true
```

Amazon EMR job definition

```json
"name": "nyTaxiAnalytics",
"virtualClusterId": "awlrqupqxuy8ijws3c0q2ik81",
"executionRoleArn": "arn:aws:iam::152324983118:role/gromav",
"releaseLabel": "emr-6.3.0-latest",
"jobDriver": {
  "sparkSubmitJobDriver": {
    "entryPoint": "s3://aws-data-lake-workshop/emr-eks-demo/emr-eks-demo-assembly-6.3.0.jar",
    "sparkSubmitParameters": "--class ValueZones"
  }
},
"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "spark-defaults",
      "properties": {
        "spark.kubernetes.node.selector.noderole": "spark",
        "spark.kubernetes.driver.podTemplateFile": "s3://myBucket/emr-eks-demo/driver-pod-template.yml",
        "spark.kubernetes.executor.podTemplateFile": "s3://myBucket/emr-eks-demo/executor-pod-template.yml"
      }
```

Amazon EMR job pod template

```yaml
apiVersion: v1
kind: Pod

spec:
  tolerations:
  - key: noderole
    operator: Equal
    value: spark
    effect: NoSchedule
```

Single AZ deployments

- Spark workloads generate lots of network transfer in shuffle phases. Cross AZ network can be costly and degrade performance.

- How to deploy Amazon EMR jobs in single AZ?
  - Amazon EKS managed nodegroups automatically add AZ labels (topology.kubernetes.io/zone: us-east-1a)
  - Amazon EKS Cluster Autoscaler choose one AZ based on the costs for the Spark driver
  - Use AZ level affinity to co-locate Spark executors in the same AZ

aws

Single AZ code example

Eksctl nodegroup definition

```yaml
- name: spot-4xl-arm-nvme-1b
  availabilityZones: ["us-east-1b"]
  minSize: 0
  maxSize: 4
  privateNetworking: true
  instancesDistribution:
    instanceTypes: ["r6gd.4xlarge"]
    onDemandBaseCapacity: 0
    onDemandPercentageAboveBaseCapacity: 0
    spotAllocationStrategy: capacity-optimized
  labels:
    arch: arm
    disk: nvme
    noderole: spark
  tags:
    k8s.io/cluster-autoscaler/node-template/label/arch: arm
    k8s.io/cluster-autoscaler/node-template/label/kubernetes.io/os: linux
    k8s.io/cluster-autoscaler/node-template/label/noderole: spark
    k8s.io/cluster-autoscaler/node-template/label/disk: nvme
    k8s.io/cluster-autoscaler/node-template/label/node-lifecycle: spot
    k8s.io/cluster-autoscaler/node-template/taint/spot: "true:NoSchedule"
    k8s.io/cluster-autoscaler/node-template/label/topology.kubernetes.io/zone: us-east-1b
    k8s.io/cluster-autoscaler/experiments: owned
    k8s.io/cluster-autoscaler/enabled: "true"
  taints:
    noderole: "spark:NoSchedule"
  iam:
    withAddonPolicies:
      ebs: true
```

Amazon EMR job definition

```json
"name": "nyTaxiAnalytics",
"virtualClusterId": "awlrqupqxuy8ijws3c0q2ik81",
"executionRoleArn": "arn:aws:iam::152324983118:role/gromav",
"releaseLabel": "emr-6.3.0-latest",
"jobDriver": {
  "sparkSubmitJobDriver": {
    "entryPoint": "s3://aws-data-lake-workshop/emr-eks-demo/emr-eks-demo-assembly-6.3.0.jar",
    "sparkSubmitParameters": "--class ValueZones"
  }
},
"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "spark-defaults",
      "properties": {
        "spark.kubernetes.node.selector.noderole": "spark",
        "spark.kubernetes.node.selector.arch": "arm",
        "spark.kubernetes.driver.label.spark/app": "emr-eks-optimized",
        "spark.kubernetes.executor.label.spark/app": "emr-eks-optimized",
        "spark.kubernetes.driver.podTemplateFile": "s3://my-bucket/emr-eks-demo/driver-pod-template.yml",
        "spark.kubernetes.executor.podTemplateFile": "s3://my-bucket/emr-eks-demo/executor-pod-template.yml"
      }
```

Amazon EMR job pod template

```yaml
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: spark/app
            operator: In
            values:
              - emr-eks-optimized
        topologyKey: topology.kubernetes.io/zone
```

Kubernetes local volumes for tmp data

- Spark exchanges data between nodes (shuffle) and write this tmp data to disk before sending over the network
- Default configuration is using Amazon EC2 root volume used by the kubelet and limited in space and performance
- For shuffle intensive workloads, recommended to use fast local storage

- How to leverage fast local storage?
- Use nodegroups with NVMe instance store (r5d, r5ad, r5dn…) and label them (disk: nvme)
- Configure the bootstrap action to initialize a RAID0 on top of instance stores
- Use node selector with the NVME disk label
- Configure Amazon EMR job to mount the local storage as HostPath volume
- Set the *spark.tmp.dir* to this volume

aws

Kubernetes local volumes code example

Eksctl nodegroup definition

```yaml
- name: spot-4xl-arm-nvme-1b
  availabilityZones: ["us-east-1b"]
  minSize: 0
  maxSize: 4
  privateNetworking: true
  instancesDistribution:
    instanceTypes: ["r6gd.4xlarge"]
    onDemandBaseCapacity: 0
    onDemandPercentageAboveBaseCapacity: 0
    spotAllocationStrategy: capacity-optimized
  labels:
    arch: arm
    disk: nvme
    noderole: spark
  iam:
    withAddonPolicies:
      autoScaler: true
      cloudWatch: true
    attachPolicyARNs:
      - arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
      - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
      - arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
  preBootstrapCommands:
    - "yum install nvme-cli mdadm -y"
    - "mkdir -p /pv-disks/local"
    - "instance_stores=$(nvme list | awk '/Instance Storage/ {print $1}') && count=$(echo
    - "mdadm --wait /dev/md0"
    - "mkfs.ext4 /dev/md0"
    - "mdadm --detail --scan >> /etc/mdadm.conf"
    - "echo /dev/md0 /pv-disks/local ext4 defaults,noatime 0 2 >> /etc/fstab"
    - "mount -a"
```

Amazon EMR job definition

```json
"name": "nyTaxiAnalytics",
"virtualClusterId": "awlrqupqxuy8ijws3c0q2ik81",
"executionRoleArn": "arn:aws:iam::152324983118:role/gromav",
"releaseLabel": "emr-6.3.0-latest",
"jobDriver": {
  "sparkSubmitJobDriver": {
    "entryPoint": "s3://aws-data-lake-workshop/emr-eks-demo/emr-eks-demo-assembly-6.3.0.jar",
    "sparkSubmitParameters": "--class ValueZones"
  }
},
"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "spark-defaults",
      "properties": {
        "spark.kubernetes.executor.volumes.hostPath.spark-local-dir-1.mount.path": "/pv/tmp",
        "spark.kubernetes.executor.volumes.hostPath.spark-local-dir-1.options.path": "/pv-disks/local",
        "spark.local.dir": "/pv/tmp",
        "spark.kubernetes.driver.podTemplateFile": "s3://my-bucket/emr-eks-demo/driver-pod-template.yml",
        "spark.kubernetes.executor.podTemplateFile": "s3://my-bucket/emr-eks-demo/executor-pod-template.yml"
      }
```

Amazon EMR job pod template

```yaml
apiVersion: v1
kind: Pod

spec:
  initContainers:
  - name: volume-permissions
    image: busybox
    command: ['/bin/sh', '-c', 'chown -R 999 /pv/tmp']
    volumeMounts:
    - mountPath: /pv/tmp
      name: spark-local-dir-1
```

aws

Autoscaling

Amazon EMR Spark application can autoscale using Dynamic Ressource Allocation but requires Amazon EKS resources to start new Pods

Amazon EKS is compatible with Kubernetes Cluster Autoscaler

Scaling in can impact Amazon EMR job if shuffle data are lost

How to autoscale Amazon EMR Spark?

- Install the K8S Cluster Autoscaler
- Configure the nodegroups to be managed by the Cluster Autoscaler
- Configure Dynamic Resource Allocation in EMR job
- Enable beta feature to avoid removing executors with shuffle data

  spark.dynamicAllocation.shuffleTracking.enabled
  spark.dynamicAllocation.shuffleTracking.timeout

Optional: Amazon FSX Lustre can be used to persist shuffle data outside executors

aws

Autoscaling code example

Eksctl nodegroup definition

```yaml
- name: spot-4xl-arm-nvme-1b
  availabilityZones: ["us-east-1b"]
  minSize: 0
  maxSize: 4
  privateNetworking: true
  instancesDistribution:
    instanceTypes: ["r6gd.4xlarge"]
    onDemandBaseCapacity: 0
    onDemandPercentageAboveBaseCapacity: 0
    spotAllocationStrategy: capacity-optimized
  tags:
    k8s.io/cluster-autoscaler/node-template/label/arch: arm
    k8s.io/cluster-autoscaler/node-template/label/kubernetes.io/os: linux
    k8s.io/cluster-autoscaler/node-template/label/noderole: spark
    k8s.io/cluster-autoscaler/node-template/label/disk: nvme
    k8s.io/cluster-autoscaler/node-template/label/node-lifecycle: spot
    k8s.io/cluster-autoscaler/node-template/taint/spot: "true:NoSchedule"
    k8s.io/cluster-autoscaler/node-template/label/topology.kubernetes.io/zone: us-east-1b
    k8s.io/cluster-autoscaler/experiments: owned
    k8s.io/cluster-autoscaler/enabled: "true"
  iam:
    withAddonPolicies:
      ebs: true
      fsx: true
      efs: true
      autoScaler: true
      cloudWatch: true
```

Amazon EMR job definition

```json
"name": "nyTaxiAnalytics",
"virtualClusterId": "awlrqupqxuy8ijws3c0q2ik81",
"executionRoleArn": "arn:aws:iam::152324983118:role/gromav",
"releaseLabel": "emr-6.3.0-latest",
"jobDriver": {
  "sparkSubmitJobDriver": {
    "entryPoint": "s3://aws-data-lake-workshop/emr-eks-demo/emr-eks-demo-assembly-6.3.0.jar",
    "sparkSubmitParameters": "--class ValueZones"
  }
},
"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "spark-defaults",
      "properties": {
        "spark.dynamicAllocation.enabled":"true",
        "spark.dynamicAllocation.minExecutors": "10",
        "spark.dynamicAllocation.maxExecutors": "20",
        "spark.kubernetes.allocation.batch.size": "10",
        "spark.dynamicAllocation.executorAllocationRatio": "1",
        "spark.dynamicAllocation.shuffleTracking.enabled": "true",
        "spark.dynamicAllocation.shuffleTracking.timeout": "300s",
        "spark.executor.memory": "12G",
        "spark.driver.memory": "4G",
        "spark.executor.cores": "3",
```

Using Amazon EC2 Spot instances

- Amazon EC2 Spot is a good fit for Spark workloads because Spark has internal mechanism to recover an executor loss
- The Spark driver is the critical component containing the processing logic to recover, missing data is recomputed

- How to use EC2 Spot with EMR?
- Amazon EKS managed nodegroups automatically add lifecycle labels (eksCapacity=SPOT)
- Schedule driver on on-demand and executor on Spot
- Use Taints and Tolerations to explicity reject non-tolerant workload from EC2 Spot
- External shuffle service to improve resiliency

aws

Using Amazon EC2 Spot code example

Eksctl nodegroup definition

```yaml
- name: spot-4xl-arm-nvme-1b
  availabilityZones: ["us-east-1b"]
  minSize: 0
  maxSize: 4
  privateNetworking: true
  instancesDistribution:
    instanceTypes: ["r6gd.4xlarge", "r6g.4xlarge"]
    onDemandBaseCapacity: 0
    onDemandPercentageAboveBaseCapacity: 0
    spotAllocationStrategy: capacity-optimized
  tags:
    k8s.io/cluster-autoscaler/node-template/label/node-lifecycle: spot
    k8s.io/cluster-autoscaler/node-template/taint/spot: "true:NoSchedule"
    k8s.io/cluster-autoscaler/experiments: owned
    k8s.io/cluster-autoscaler/enabled: "true"
  taints:
    spot: "true:NoSchedule"
```

Amazon EMR job definition

```json
"name": "nyTaxiAnalytics",
"virtualClusterId": "awlrqupqxuy8ijws3c0q2ik81",
"executionRoleArn": "arn:aws:iam::152324983118:role/gromav",
"releaseLabel": "emr-6.3.0-latest",
"jobDriver": {
  "sparkSubmitJobDriver": {
    "entryPoint": "s3://aws-data-lake-workshop/emr-eks-demo/emr-eks-demo-assembly-6.3.0.jar",
    "sparkSubmitParameters": "--class ValueZones"
  }
},
"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "spark-defaults",
      "properties": {
        "spark.kubernetes.driver.podTemplateFile": "s3://my-bucket/emr-eks-demo/driver-pod-template.yml",
        "spark.kubernetes.executor.podTemplateFile": "s3://my-bucket/emr-eks-demo/executor-pod-template.yml"
      }
```

Amazon EMR job pod template

```yaml
apiVersion: v1
kind: Pod

spec:
  tolerations:
  - key: spot
    operator: Equal
    value: true
    effect: NoSchedule
  nodeSelector:
    node-lifecycle: spot
```

Using AWS Fargate

- AWS Fargate provides serverless compute engine for Amazon EKS
- EMR on EKS can leverage Fargate Pods for both Spark drivers and executors

- How to use Fargate Pods with EMR?
- Create a Fargate profile for the EMR Virtual Cluster namespace
- The Fargate profile can use executors labels to schedule executors only (speed up the scheduling)
- Limitations:
- Approximately 4 minutes to launch Fargate Pods
- Limited disk space for tmp data (20 GB)

aws

Using AWS Fargate code example

Eksctl fargate profile definition

```yaml
fargateProfiles:
  - name: emr-serverless
    selectors:
      # All workloads in the "emr-serverless" Kubernetes namespace matching the following
      # label selectors will be scheduled onto Fargate:
      - namespace: emr-serverless
        # Only Spark executors (Pods with this label) will run on Fargate
        labels:
          spark-role: executor
```

Amazon EMR job definition

```json
"name": "nyTaxiAnalytics",
"virtualClusterId": "lurw626szf8516399slt8ayc8",
"executionRoleArn": "arn:aws:iam::152324983118:role/gromav",
"releaseLabel": "emr-6.3.0-latest",
"jobDriver": {
  "sparkSubmitJobDriver": {
    "entryPoint": "s3://aws-data-lake-workshop/emr-eks-demo/emr-eks-demo-assembly-6.3.0.jar",
    "entryPointArguments": ["s3://nyc-tlc/trip data", "2017","s3://nyc-tlc/misc/taxi _zone_looku
    "sparkSubmitParameters": "--class ValueZones"
  }
},
"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "spark-defaults",
      "properties": {
        "spark.hadoop.hive.metastore.client.factory.class":"com.amazonaws.glue.catalog.metastore
        "spark.dynamicAllocation.enabled":"true",
        "spark.dynamicAllocation.minExecutors": "10",
        "spark.dynamicAllocation.maxExecutors": "20",
        "spark.kubernetes.allocation.batch.size": "10",
        "spark.dynamicAllocation.executorAllocationRatio": "1",
        "spark.dynamicAllocation.shuffleTracking.enabled": "true",
        "spark.dynamicAllocation.shuffleTracking.timeout": "300s",
        "spark.executorEnv.AWS_REGION": "us-east-1",
        "spark.kubernetes.driverEnv.AWS_REGION": "us-east-1",
        "spark.executor.memory": "12G",
        "spark.driver.memory": "4G",
        "spark.executor.cores": "2",
        "spark.kubernetes.driver.label.spark/role": "driver",
        "spark.kubernetes.executor.label.spark/role": "executor"
      }
```

Thank you!

Vincent Gromakowski

gromav@amazon.com

aws