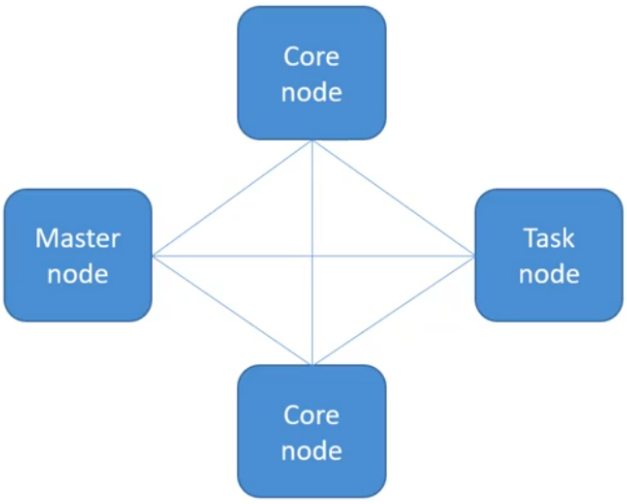
Emr

What is EMR?

- Elastic MapReduce
- Managed Hadoop framework on EC2 instances
- Includes Spark, HBase, Presto, Flink, Hive & more
- EMR Notebooks
- Several integration points with AWS

Amazon EMR

An EMR Cluster



- **Master node**: manages the cluster
  - Tracks status of tasks, monitors cluster health
  - Single EC2 instance (it can be a single node cluster even)
  - AKA "leader node"
- **Core node**: Hosts HDFS data and runs tasks
  - Can be scaled up & down, but with some risk
  - Multi-node clusters have at least one
- **Task node**: Runs tasks, does not host data
  - Optional
  - No risk of data loss when removing
  - Good use of **spot instances**

EMR Usage

- Transient vs Long-Running Clusters
  - Transient clusters terminate once all steps are complete
    - Loading data, processing, storing – then shut down
    - Saves money
  - Long-running clusters must be manually terminated
    - Basically a data warehouse with periodic processing on large datasets
    - Can spin up task nodes using Spot instances for temporary capacity
    - Can use reserved instances on long-running clusters to save $
    - Termination protection on by default, auto-termination off

EMR Usage

- Frameworks and applications are specified at cluster launch
- Connect directly to master to run jobs directly
- Or, submit ordered steps via the console
  - Process data in S3 or HDFS
  - Output data to S3 or somewhere
  - Once defined, steps can be invoked via the console

EMR / AWS Integration

- Amazon EC2 for the instances that comprise the nodes in the cluster
- Amazon VPC to configure the virtual network in which you launch your instances
- Amazon S3 to store input and output data
- Amazon CloudWatch to monitor cluster performance and configure alarms
- AWS IAM to configure permissions
- AWS CloudTrail to audit requests made to the service
- AWS Data Pipeline to schedule and start your clusters

EMR Storage

- HDFS
  - Hadoop Distributed File System
  - Multiple copies stored across cluster instances for redundancy
  - Files stored as blocks (128MB default size)
  - Ephemeral – HDFS data is lost when cluster is terminated!
  - But, useful for caching intermediate results or workloads with significant random I/O
  - Hadoop tries to process data where it is stored on HDFS
- EMRFS: access S3 as if it were HDFS
  - Allows persistent storage after cluster termination
  - **EMRFS Consistent View** – Optional for S3 consistency
    - Uses DynamoDB to track consistency
    - May need to tinker with read/write capacity on DynamoDB
  - **New in 2021: S3 is Now Strongly Consistent!**

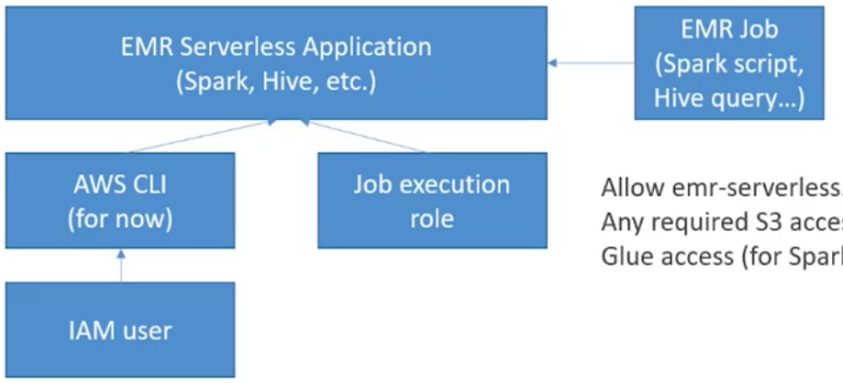
EMR Storage

- Local file system
  - Suitable only for temporary data (buffers, caches, etc)
- EBS for HDFS
  - Allows use of EMR on EBS-only types (M4, C4)
  - Deleted when cluster is terminated
  - EBS volumes can only be attached when launching a cluster
  - If you manually detach an EBS volume, EMR treats that as a failure and replaces it

EMR Serverless

- Coming in 2022
- Choose an EMR Release and Runtime (Spark, Hive, Presto)
- Submit queries / scripts via job run requests
- EMR manages underlying capacity
  - But you can specify default worker sizes & pre-initialized capacity
  - EMR computes resources needed for your job & schedules workers accordingly
  - All within one region (across multiple AZ's)
- Why is this a big deal?
  - You no longer have to estimate how many workers are needed for your workloads – they are provisioned as needed, automatically.
- Serverless? Really?
  - TBH you still need to think about worker nodes and how they are configured.

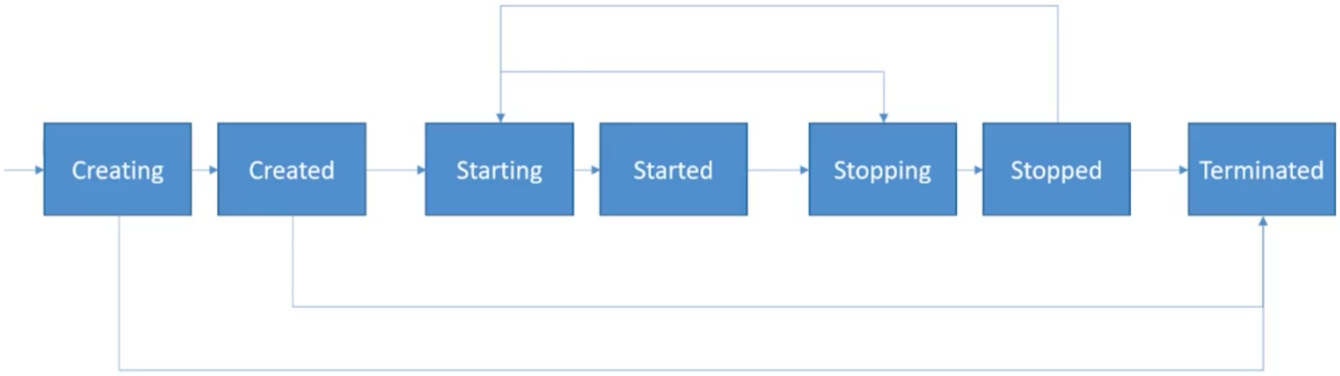
Using EMR Serverless




```
aws emr-serverless start-job-run \
  --application-id <application_id> \
  --execution-role-arn <execution_role_arn> \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "s3://us-east-1.elasticmapreduce/emr-containers/samples/wordcount/scripts/wordcount.py",
      "entryPointArguments": ["s3://DOC-EXAMPLE-BUCKET/output"],
      "sparkSubmitParameters": "--conf spark.executor.cores=1 --conf spark.executor.memory=4g --conf spark.driver.cores=1 --conf spark.driver.memory=4g --conf spark.executor.instances=1"
    }
  }' \
  --configuration-overrides '{
    "monitoringConfiguration": {
      "s3MonitoringConfiguration": {
        "logUri": "s3://DOC-EXAMPLE-BUCKET/logs"
      }
    }
  }'
```

Allow emr-serverless.amazonaws.com service
Any required S3 access for scripts & data
Glue access (for SparkSQL), KMS keys

EMR Serverless Application Lifecycle



This isn't all automatic!
Must call API's such as CreateApplication, StartApplication, StopApplication, and importantly DeleteApplication to avoid excess charges.

Pre-Initialized Capacity

- Spark adds 10% overhead to memory requested for drivers & executors
- Be sure initial capacity is at least 10% more than requested by the job

```
aws emr-serverless create-application \
 --type "SPARK" \
 --name <"my_application_name"> \
 --release-label "emr-6.5.0-preview" \
 --initial-capacity '{
  "DRIVER": {
     "workerCount": 5,
     "resourceConfiguration": {
        "cpu": "2vCPU",
        "memory": "4GB"
     }
  },
  "EXECUTOR": {
     "workerCount": 50,
     "resourceConfiguration": {
        "cpu": "4vCPU",
        "memory": "8GB"
     }
  }
 }' \
 --maximum-capacity '{
  "cpu": "400vCPU",
  "memory": "1024GB"
 }'
```

EMR Serverless Security

- Basically the same as EMR
- EMRFS
  - S3 encryption (SSE or CSE) at rest
  - TLS in transit between EMR nodes and S3
- S3
  - SSE-S3, SSE-KMS
- Local disk encryption
- Spark communication between drivers & executors is encrypted
- Hive communication between Glue Metastore and EMR uses TLS
- Force HTTPS (TLS) on S3 policies with aws:SecureTransport