

Neo4j Enterprise on AWS EC2

David Pecollet
Customer Success Architect

neo4j

Table of content

Official Neo4j offering

Images, Marketplace, Cloudformation

Architecture Recommendations

Sizing, Instance type, Storage, Network

Operations

Backup, Cluster self-healing

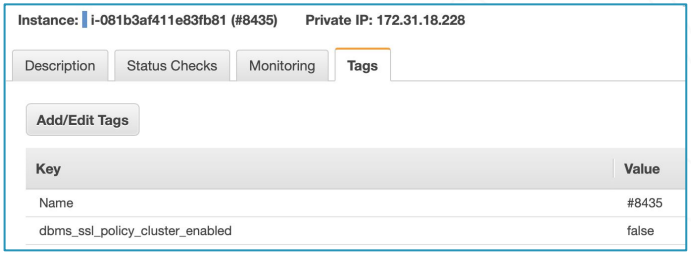
Neo4j Offering

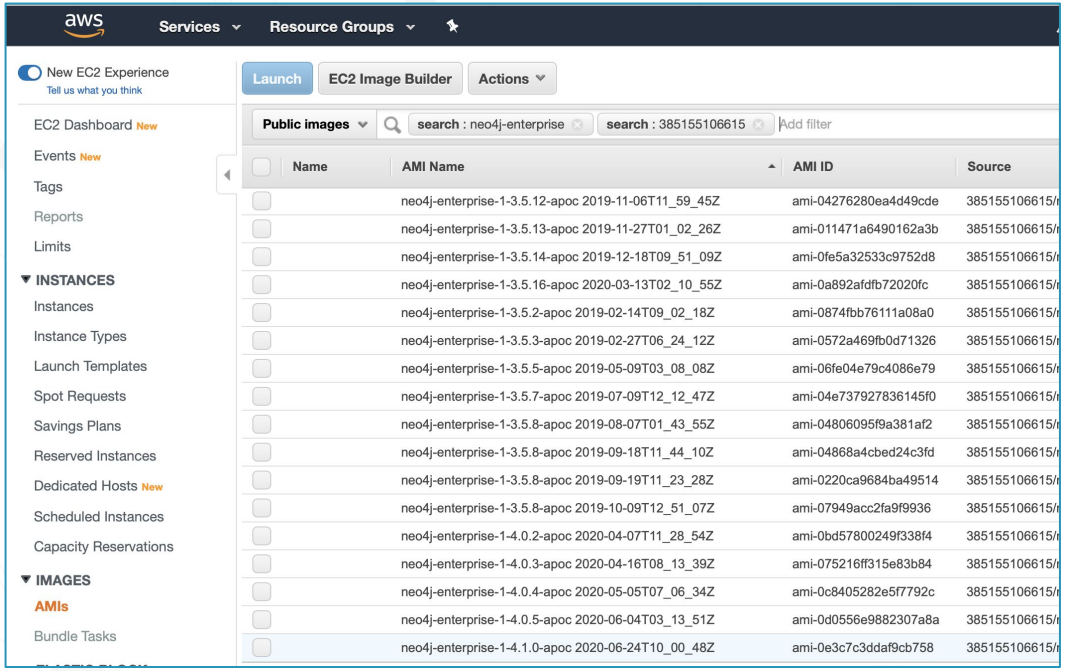


Official AMIs

- Ubuntu based

  file locations: */var/lib/neo4j/data*
  /usr/bin
  /etc/neo4j/neo4j.conf
  /var/log/neo4j/
- APOC included
- Config via tags





AWS Marketplace



Neo4j Enterprise Causal Cluster

By: Neo4j Latest Version: 4.1.1

Neo4j is an open-source, NoSQL, native graph database that provides an ACID-compliant transactional backend for your applications. The Enterprise Edition includes all that Community

Show more

Linux/Unix ☆☆☆☆☆ 0 AWS reviews | 39 external reviews

BYOL

Continue to Subscribe

Save to List

Typical Total Price

$0.133/hr

Total pricing per instance for services hosted on r4.large in US East (N. Virginia). View Details

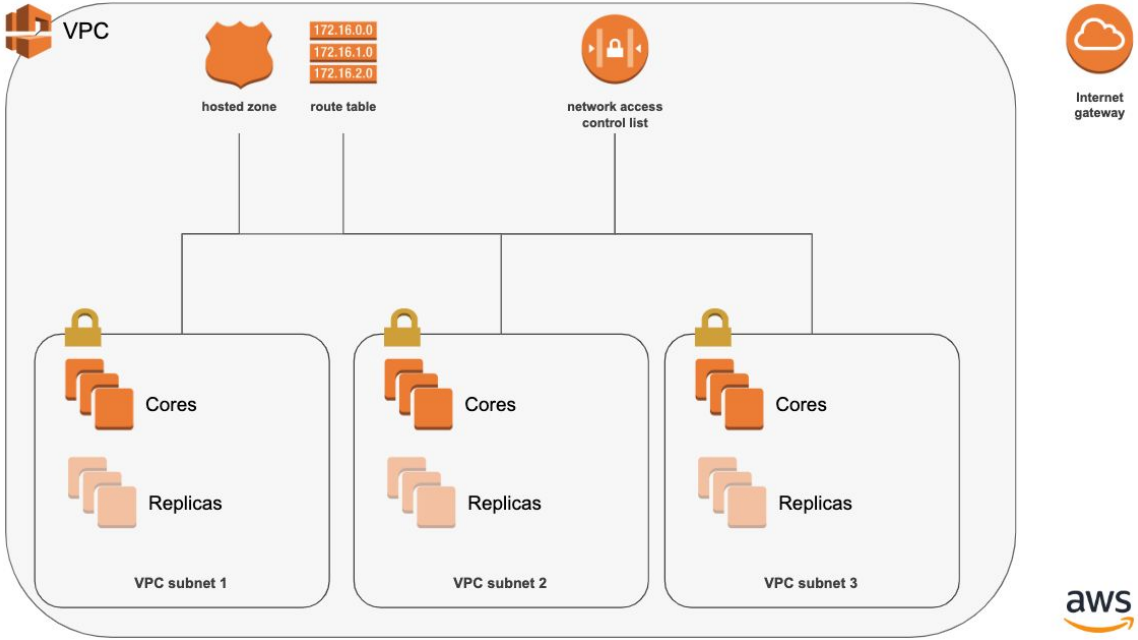
CloudFormation template

CloudFormation templates

```
https://neo4j.com/developer/kb/deploy-aws-cloudformation/

http://neo4j-cloudformation.s3.amazonaws.com/neo4j-enterprise
-stack-$VERSION.json
```

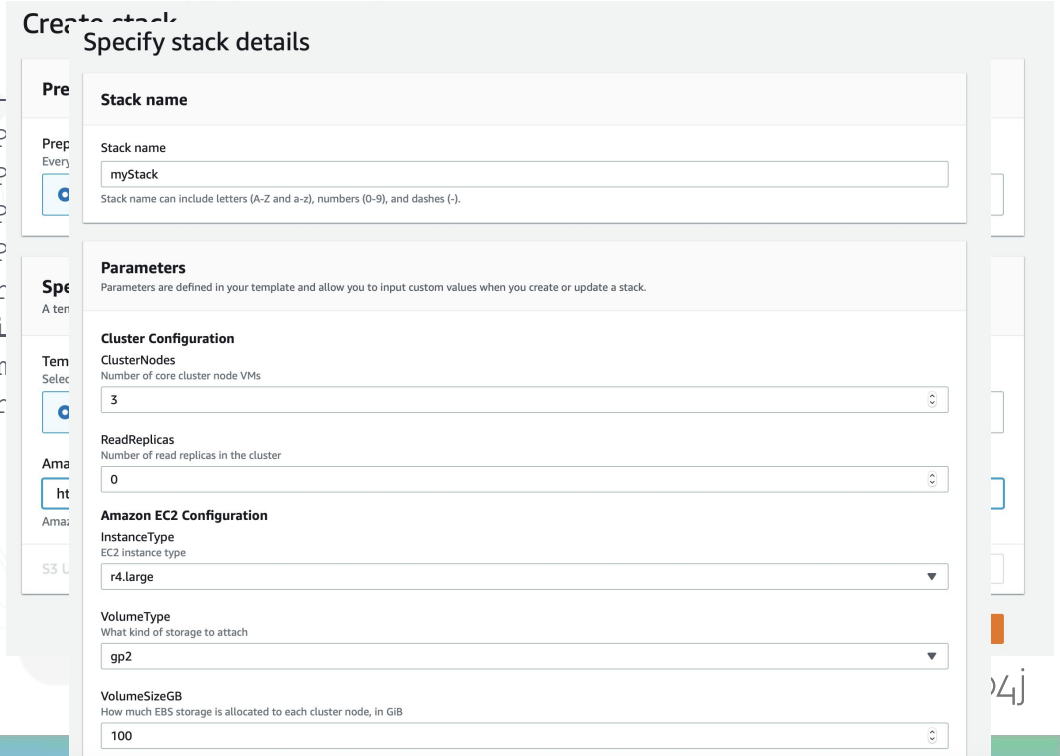
- New virtual private cloud (VPC)
- Up to 7 Cores & 5 Read Replicas
- Spread across availability zones



neo4j

CloudFormation - Stack creation

```
aws cloudformation create-stack \
   --stack-name myStack \
   --template-body file://neo4j-enterprise-
   --parameters ParameterKey=ClusterNodes,P
                ParameterKey=ReadReplicas,P
                ParameterKey=InstanceType,P
                ParameterKey=VolumeSizeGB,P
                ParameterKey=VolumeType,Par
                ParameterKey=NetworkWhiteli
                ParameterKey=Password,Param
                ParameterKey=SSHKeyName,Par
    --capabilities CAPABILITY_NAMED_IAM
```

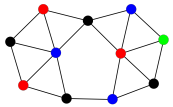


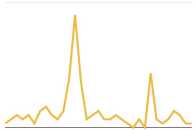
Architecture Recommendations



Sizing

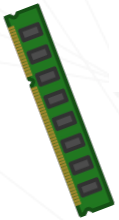
data size

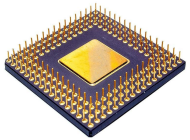




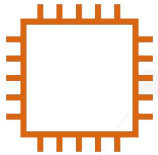
workload

hardware requirements

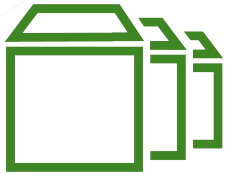








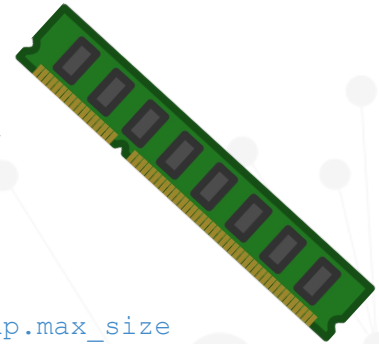
AWS instance type



storage

neo4j

Sizing memory



JVM heap `dbms.memory.heap.max_size`

[4GB - 30GB]

concurrent transactions, large writes, aggregations?

`dbms.memory.transaction.global_max_size` [4.1]

Page cache `dbms.memory.pagecache.size`

[30% - 100%] of store size (ideally 100%)

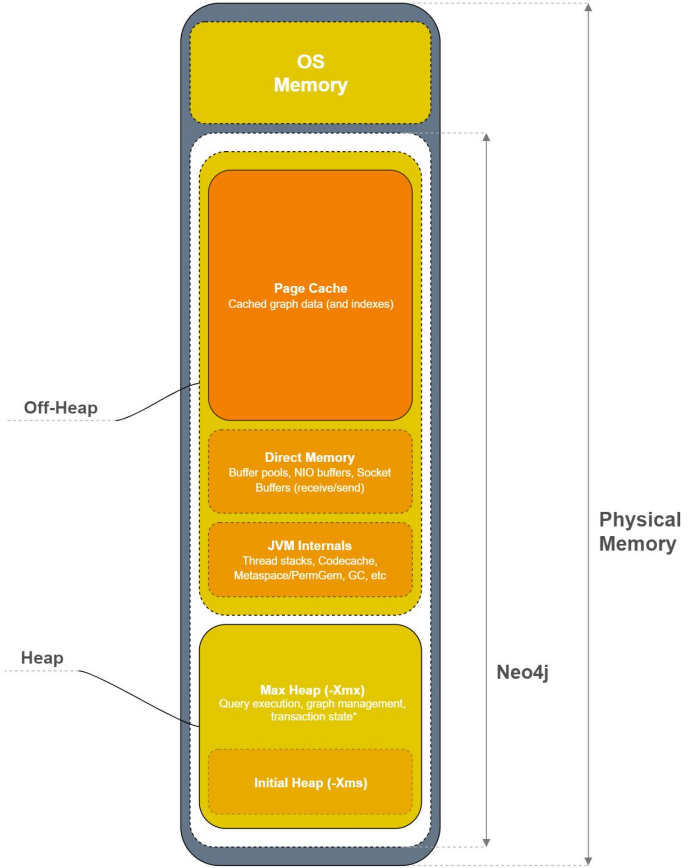
hot set?

Other

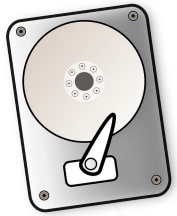
network buffers : concurrent connections

lucene indexes

OS headroom



Sizing disk



Store size

of nodes, relationships, properties

# of indexed properties

IOPS

access patterns

sequential writes & random reads

important if

- pagecache < data (cache misses on reads)
- data changes a lot (heavy writes)

`dbms.checkpoint.iops.limit`

https://neo4j.com/hardware-sizing-calculator



Estimated Graph Size

The calculator can estimate your memory requirements with the following input:

You can use suffixes k, m, g or b to denote larger values, e.g. 10m.

Nodes

5,000,000

Relationships

25,000,000

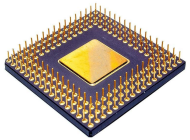
Properties per Node

5

Properties per Relationship

5

Other sizing Considerations



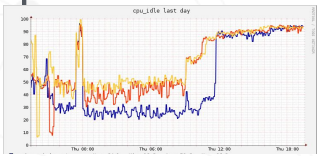
CPU

concurrent transactions

- each transaction runs in a thread
- number of cores determines parallelism

irregular traffic patterns?

- burstable instance



tied to Neo4j licence



Network

amount of data read (or written)

backups

⚠ Intra-Region Data Transfer

Choosing Storage

Instance Store

- ephemeral
  - only for clusters where data is replicated
  - new instances need to be seeded from backup
- fastest
- price included with instance

EBS

- persistent
- speed
  - provisioned IOPS (io1/io2) = next fastest
  - gp2 (100-16000 IOPS depending on size)
- price

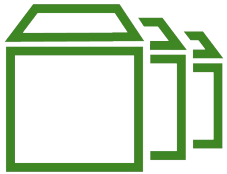
gp2 : 50$/month*
io1/io2 : 62.5$/month + 65$ per 1000 IOPS

** current prices for 500GB in region US-East*

~~*EFS, NFS*~~

Tips

- SSD, NVMe
- several drives
  - *separate drives for transaction logs vs data store ; RAID striping*



Choosing Instance types

(x86 only)

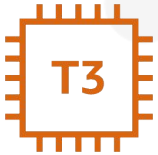
General purpose

for high transaction loads



or m5d for Instance Storage

for irregular traffic patterns



burstable CPU
only up to 32GB

Memory-optimized

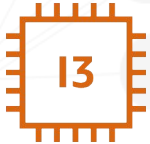
for large data sets



or r5d for Instance Storage



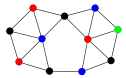
high memory/CPU ratios
up to 4TB RAM
large pagecache



very large disks
smaller page cache

neo4j

Example 1

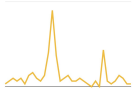


Very Small data set

10M nodes / 50M relationships / 200M properties

↓ https://neo4j.com/hardware-sizing-calculator

size on disk ~10GB



Medium concurrency, with spikes

↓

JVM heap 8GB

~10GB data store

↓fully in cache +20% for growth

page cache 12GB

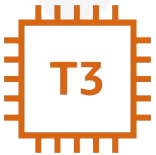
↓+JVM heap 8GB

↓+OS overhead 20%

requires 25GB RAM

↓

t3.2xlarge + EBS



8 vCPUs

32GB RAM

~253$/month

neo4j

Example 2



Small data set

30M nodes / 150M relationships / 600M properties

↓ https://neo4j.com/hardware-sizing-calculator

size on disk ~30GB



Medium concurrency

↓

JVM heap 8GB

~30GB data store

↓ fully in cache +20% for growth

page cache 36GB

↓ +JVM heap 8GB

↓ +OS overhead 20%

requires 55GB RAM

↓

r5.2xlarge + EBS



8 vCPUs

64GB RAM

~388$/month

neo4j

Example 3



Medium data set

100M nodes / 500M relationships / 2000M properties

↓ https://neo4j.com/hardware-sizing-calculator

size on disk ~100GB



Low concurrency

↓

JVM heap 8GB

~100GB data store

↓3/4 in cache

page cache 75GB

↓+JVM heap 8GB

↓+OS overhead 20%

requires 104GB RAM

↓

x1e.xlarge + EBS



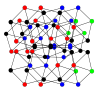
4 vCPUs

122 GB RAM

~634$/month

neo4j

Example 4



Large data set

300M nodes / 1500M relationships / 6000M properties

↓ https://neo4j.com/hardware-sizing-calculator

size on disk ~300GB



Very high concurrency

↓

JVM heap 30GB

~300GB data store

↓50% in cache

page cache 150GB

↓+JVM heap 30GB

↓+OS overhead 20% + network buffers 5GB

requires 230GB RAM

↓

m5d.16xlarge

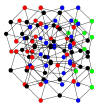


64 vCPUs ~2639$/month

256 GB RAM

4 x 600 NVMe SSD

Example 5



Very Large data set

3000M nodes / 15000M relationships / 60000M properties

↓ https://neo4j.com/hardware-sizing-calculator

size on disk ~3TB

↓

at least 4TB of disk

disk must be fast : high cache miss ratio likely



Very high concurrency

↓

JVM heap 30GB

~3TB data store

↓

page cache 350GB

↓ +JVM heap 30GB

↓ +OS overhead 20% + network buffers 5GB

requires 480GB RAM

↓

i3.16xlarge



64 vCPUs ~3644$/month

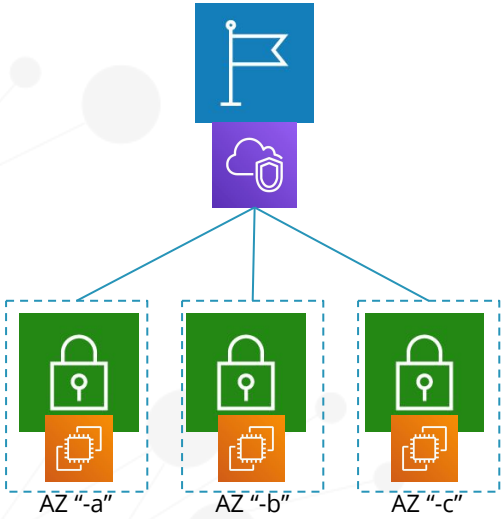
488 GB RAM

8 x 1,900 NVMe SSD

Network Configuration

VPC in 1 region

- 1 Subnet in each AZ
- instances spread over AZs for High Availability



Security Groups

- intra-cluster traffic : TCP 5000, 6000, 7000, 6362, 7688
- external traffic: 7687, 7474/7473 => IP whitelisting

DNS

- cluster name : 1 ‘A record’ for each core instance
- client URL `neo4j://clustername:7687`

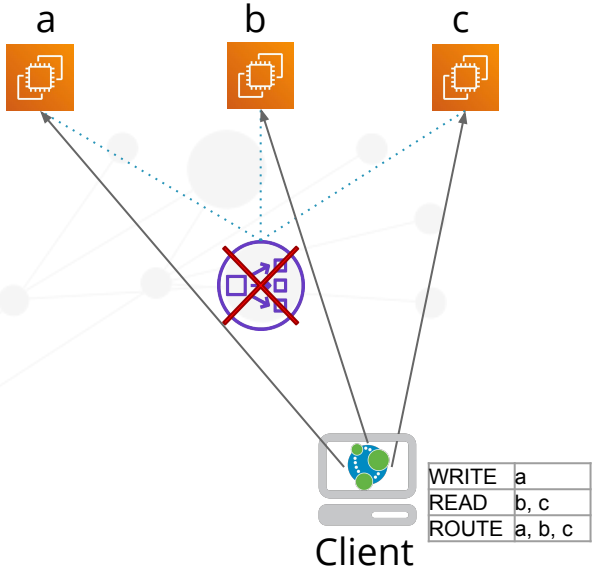
Network - BOLT Client routing

Use client routing bolt+routing:// or neo4j:// schemes

BOLT Client/driver :

- maintains routing table
- routes writes to leader
- routes reads to follower & replicas
- uses least-connected load balancing strategy

⚠ No Load Balancer - direct access client => servers



Operations



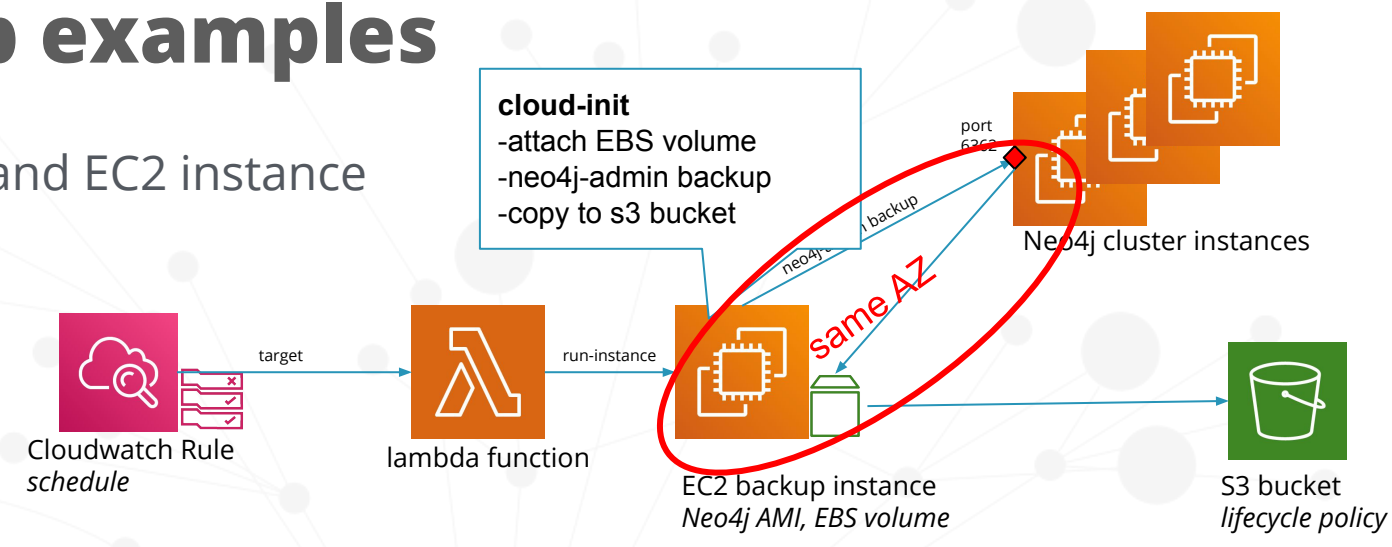


Backup

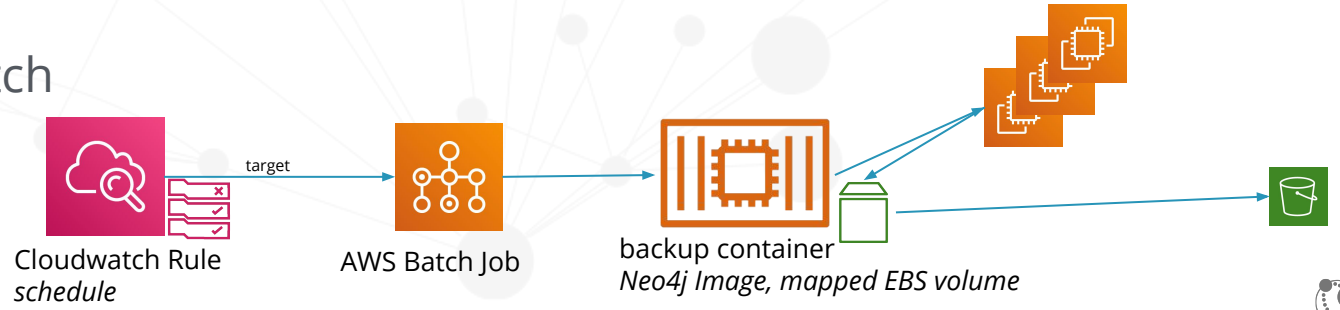
- use *neo4j-admin backup*
  - volume snapshots can't guarantee consistent backups
  - ~~AWS Backup~~
- avoid running on core instances
  - run on a separate Neo4j instance,
    - *& hit the backup endpoint (of a RR preferably, or follower…)*
  - or on a Read Replica
    - *& hit the local backup endpoint*
- storage
  - S3 (lifecycle for retention policy)
  - EBS volume (required for incremental backup)

Backup examples

- on-demand EC2 instance

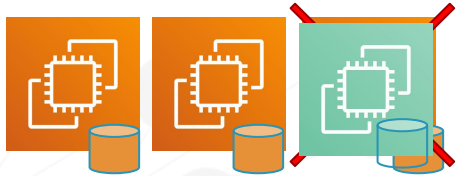


- AWS Batch



Self healing

Auto Scaling Groups



👍 Use to maintain cluster size.
👎 **NOT** scaling up/down.

- 1 ASG for core instances desired capacity=min=max=3
  => Core launch template
  `dbms.mode=CORE`

- 1 ASG for Read Replicas desired capacity=min=max=N
  => RR launch template
  `dbms.mode=READ_REPLICA`

Self healing

Stable Network identity

- required for cluster discovery

```
causal_clustering.initial_discovery_members=core0:5000,core1:5000,core2:5000
```

- Dynamic DNS

  Dynamic DNS provider
  or update DNS Records in a Route53 Hosted Zone on instance startup (cloud-init)
  Low TTL

Data seeding for new instances

- new empty instance needs to catch up

  ~= DoS attack on cluster for large databases

- restore from a backup instead (cloud-init)

  catch up from the cluster only for the remainder

