Amazon Neptune

DB Cluster

Enterprise Architecture & Production Guide

Interview Prep • SOP • POC • Cost Analysis • Architecture

Graph Database | Serverless | SPARQL / Gremlin | HA Cluster | AWS Managed

Production Ready | Enterprise Grade | AWS Certified Reference

February 2025 | Version 1.0

■ What is Amazon Neptune?

Amazon Neptune is a **fully managed, high-performance graph database service** built by AWS for workloads that have highly connected datasets. It supports two popular graph data models — **Property Graph** (via Apache TinkerPop Gremlin) and **RDF (Resource Description Framework)** (via SPARQL) — in a single engine.

> **■ Interview Answer: Neptune is NOT a relational DB. It stores data as NODES (entities) and EDGES (relationships), making it ideal for social networks, knowledge graphs, fraud detection, and recommendation engines where relationships are first-class citizens.**

■ Why We Use Amazon Neptune

Traditional relational databases struggle when the relationships between data are as important as the data itself. Neptune solves this with native graph storage and traversal.

#	Reason	Business Problem Solved
1	Complex Relationship Queries	JOIN-heavy SQL fails at 5+ hops; Neptune handles millions of hops in milliseconds
2	Fraud Detection	Detect circular transactions and suspicious entity networks in real time
3	Knowledge Graphs	Link products, concepts, and metadata across domains (e.g., Google Knowledge Graph)
4	Social Networks	Friends-of-friends, influence analysis, community detection
5	Recommendation Engines	Co-purchase, co-view, collaborative filtering via graph traversal
6	Identity Resolution	Link customer identities across platforms and data sources
7	Supply Chain	Map and traverse complex supplier-to-delivery dependency chains
8	Compliance & Lineage	Track data provenance and regulatory relationships (GDPR, HIPAA)

■■ How Amazon Neptune Works

Neptune separates **compute** from **storage**. The cluster consists of one Primary (writer) instance and up to 15 Read Replicas. All instances share a single distributed, fault-tolerant storage volume that automatically replicates across 3 Availability Zones.

Component	Role	Details
Primary Instance	Read + Write	Processes all DDL/DML, sends logs to shared storage
Read Replicas (up to 15)	Read Only	Low-latency reads, sub-10ms replica lag
Cluster Volume	Shared Storage	Auto-grows up to 128 TiB, replicated 6 ways across 3 AZs
Cluster Endpoint	Writer Entry	Always points to Primary, auto-failover on failure
Reader Endpoint	Load-Balanced Read	Distributes read traffic across all replicas
Neptune Streams	Change Data Capture	Ordered stream of graph changes for downstream processing
Neptune Serverless	Auto-Scaling Compute	Scale Neptune Capacity Units (NCUs) from 1 to 128 automatically

Query Languages Supported

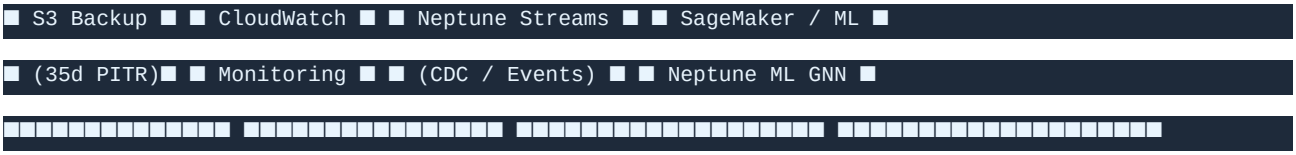
Language	Model	Use Case	Example Query Style
`Gremlin`	Property Graph	Traversal-heavy apps, social graphs	g.V().has('name','Alice').out('knows')
`SPARQL`	RDF / Semantic Web	Knowledge graphs, ontologies	SELECT ?s WHERE { ?s :knows :Bob }
`openCypher`	Property Graph	Familiar for Neo4j users	MATCH (a)-[:KNOWS]->(b) RETURN b

■ Features & Benefits

	VPC isolation, IAM auth, TLS encryption in-transit, KMS encryption at-rest, fine-grained access
■ Performance	Purpose-built graph engine, optimized index-free adjacency traversal, in-memory caching
	Multi-AZ by default, 6-way storage replication, automated failover < 30 seconds
■ Scalability	Read: add up to 15 replicas. Serverless: auto-scale compute 1–128 NCUs. Storage: auto-grows
	Continuous incremental backups to S3 (up to 35 days PITR), no I/O freeze
■ Integration	Native AWS integration: Lambda, SageMaker, Glue, S3, IAM, CloudWatch, CloudTrail
	CloudWatch metrics, Performance Insights, Neptune Streams for real-time change capture
■ Multi-model	Single cluster supports both RDF and Property Graph — no need for separate databases
	Neptune ML uses Graph Neural Networks (GNN) via SageMaker for graph-based predictions
■ Global Replication	Neptune Global Database for cross-region low-latency reads and DR

■■ Enterprise Architecture Diagram

Architecture: Multi-AZ Neptune Cluster with Read Replicas, VPC, and AWS Service Integration


S3 Backup
CloudWatch
Neptune Streams
SageMaker / ML
(35d PITR)
Monitoring
(CDC / Events)
Neptune ML GNN

■ POC — Proof of Concept

The following POC demonstrates creating a Neptune Cluster using AWS CLI and running a Gremlin traversal query to find fraud patterns in a financial graph.

Step 1: Create Neptune Subnet Group

```
aws neptune create-db-subnet-group \ --db-subnet-group-name neptune-subnet-group \
--db-subnet-group-description "Neptune Multi-AZ Subnet Group" \ --subnet-ids subnet-abc123
subnet-def456 subnet-ghi789
```

Step 2: Create Neptune DB Cluster

```
aws neptune create-db-cluster \ --db-cluster-identifier my-neptune-cluster \ --engine neptune \
--engine-version 1.3.0.0 \ --db-subnet-group-name neptune-subnet-group \ --vpc-security-group-ids
sg-0abc123def456789 \ --availability-zones us-east-1a us-east-1b us-east-1c \ --storage-encrypted \
--kms-key-id arn:aws:kms:us-east-1:123456789:key/my-key \ --iam-database-authentication-enabled \
--backup-retention-period 7 \ --tags Key=Environment,Value=Production Key=Team,Value=DataEngineering
```

Step 3: Add Primary Instance

```
aws neptune create-db-instance \ --db-instance-identifier neptune-primary \ --db-instance-class
db.r6g.xlarge \ --engine neptune \ --db-cluster-identifier my-neptune-cluster \ --availability-zone
us-east-1a
```

Step 4: Add Read Replica

```
aws neptune create-db-instance \ --db-instance-identifier neptune-replica-1 \ --db-instance-class
db.r6g.large \ --engine neptune \ --db-cluster-identifier my-neptune-cluster \ --availability-zone
us-east-1b
```

Step 5: Gremlin Fraud Detection Query (Python SDK)

```
from gremlin_python.driver import client as gremlin_client from
gremlin_python.driver.driver_remote_connection import DriverRemoteConnection from
gremlin_python.process.anonymous_traversal import traversal # Connect to Neptune cluster endpoint
NEPTUNE_ENDPOINT =
"wss://my-neptune-cluster.cluster-xyz.us-east-1.neptune.amazonaws.com:8182/gremlin" conn =
DriverRemoteConnection(NEPTUNE_ENDPOINT, 'g') g = traversal().with_remote(conn) # Load sample
financial data g.addV('account').property('id','ACC001').property('balance',5000).next()
g.addV('account').property('id','ACC002').property('balance',100).next()
g.addV('account').property('id','ACC003').property('balance',9999).next()
g.V().has('id','ACC001').addE('TRANSFER').to(g.V().has('id','ACC002')).property('amount',500).next()
g.V().has('id','ACC002').addE('TRANSFER').to(g.V().has('id','ACC003')).property('amount',490).next()
g.V().has('id','ACC003').addE('TRANSFER').to(g.V().has('id','ACC001')).property('amount',480).next()
```

```
# Detect circular fraud pattern (cycle of length 3) fraud_cycles = g.V().as_('start') \
.out('TRANSFER').out('TRANSFER').out('TRANSFER') \ .where(eq('start')) \ .path().toList()
print(f"Fraud cycles detected: {len(fraud_cycles)}") for cycle in fraud_cycles: print(f" Cycle path:
{cycle}") conn.close()
```

SOP — Standard Operating Procedure

SOP-01	Cluster Provisioning	Run IaC (Terraform/CDK), validate VPC, subnet, SG, KMS, IAM roles before CREATE
SOP-02	Access Control	Enable IAM DB Auth. No password auth in production. Use instance profiles
SOP-03	Monitoring Setup	Enable Enhanced Monitoring (60s granularity), set CloudWatch alarms for CPU > 80%, Fre
SOP-04	Backup Verification	After cluster creation, verify automated backups enabled (7–35 days), test PITR monthly
SOP-05	Failover Testing	Quarterly: trigger manual failover via Console/CLI, verify app reconnects within 30s
SOP-06	Patching	Apply minor engine versions automatically. Schedule major version upgrades in maintenan
SOP-07	Scaling	Add replicas for read scaling. Use Serverless for unpredictable workloads. Never vertically
SOP-08	Security Audit	Monthly: review IAM policies, VPC flow logs, CloudTrail audit, rotate KMS keys annually
SOP-09	Query Optimization	Profile slow queries via Neptune DFE (Dataflow Execution). Add property indices as neede
SOP-10	DR Drill	Semi-annual: restore from S3 snapshot to new cluster, validate data integrity, update RTO/
SOP-11	Cost Review	Monthly: review instance hours, I/O, storage in Cost Explorer. Right-size instances
SOP-12	Deprecation / Cleanup	Delete unused clusters, snapshots > 90 days, idle replicas to reduce cost

■ Cost Analysis (us-east-1, 2025 Pricing)

Neptune pricing has 4 dimensions: Instance Hours, I/O Requests, Storage (GB/month), and Backup Storage. Below are estimated monthly costs for 3 common deployment patterns.

Component	Dev/Test	Standard Production	Enterprise HA
Primary Instance	db.t4g.medium ~$52/mo	db.r6g.xlarge ~$438/mo	db.r6g.4xlarge ~$1,752/mo
Read Replicas	None	1 × db.r6g.large ~$219/mo	3 × db.r6g.2xlarge ~$2,628/mo
Storage	10 GB × $0.10 ~$1/mo	500 GB × $0.10 ~$50/mo	5 TB × $0.10 ~$500/mo
I/O Requests	1M × $0.0002 ~$0.20/mo	1B × $0.0002 ~$200/mo	10B × $0.0002 ~$2,000/mo
Backup Storage	Included (1×)	~$5/mo	~$50/mo
ESTIMATED TOTAL	**~$53/month**	**~$912/month**	**~$6,930/month**

■ Cost Tip: Use Neptune Serverless for dev/test workloads that are idle for long periods. You only pay for NCUs consumed. Reserved Instances can save 30–60% for steady-state production clusters.

How to Create — Step-by-Step

Step 1: Pre-requisites

- VPC with private subnets in at least 2 AZs
- Security Group: allow inbound TCP 8182 from application SG
- KMS key for encryption at rest
- IAM Role for enhanced monitoring

Step 2: Create DB Subnet Group

- Navigate to: RDS Console → Subnet Groups → Create DB Subnet Group
- Select your VPC and include subnets from 3 AZs
- Name it: neptune-prod-subnet-group

Step 3: Create Neptune Cluster

- Navigate to: Neptune Console → Databases → Create Database
- Engine: Amazon Neptune | Version: latest stable (1.3.x)
- Instance class: db.r6g.xlarge (Production) or db.t4g.medium (Dev)
- Multi-AZ: Enable | Storage encryption: Enable (select KMS key)
- VPC: Select your VPC | Subnet Group: neptune-prod-subnet-group
- IAM DB Authentication: Enable | Backup retention: 7 days minimum
- Enable Performance Insights | Enable Enhanced Monitoring

Step 4: Add Read Replicas

- Neptune Console → Select Cluster → Add Reader
- Place in a different AZ from Primary
- For high read workloads, add 2–3 replicas across AZs

Step 5: Configure Security

- Restrict Security Group to only allow app-tier SG on port 8182
- Enable VPC Flow Logs for audit trail
- Create IAM policy for neptune:Connect, attach to app role

Step 6: Load Data

- Use Neptune Bulk Loader with S3 source (CSV or Turtle RDF format)
- aws neptune start-loader-job --source s3://bucket/data/ --format csv --iamRoleArn arn:...
- Monitor loader status via Neptune Console or CLI

Step 7: Connect & Test

- Use Gremlin Console or Python gremlinpython SDK
- Connection: wss://:8182/gremlin
- Run: g.V().limit(10).toList() to validate connectivity

Step 8: Set Up Monitoring Alarms

- CloudWatch Alarm: CPUUtilization > 80% → SNS notification
- CloudWatch Alarm: FreeableMemory < 1024 MB → SNS notification
- CloudWatch Alarm: ReplicationLag > 100ms → SNS notification
- Enable Neptune Streams if downstream CDC is required

■ Interview Q&A; — Neptune Deep Dive

Q1: What is the difference between Neptune and DynamoDB?

DynamoDB is a key-value/document NoSQL DB optimized for simple lookups at massive scale. Neptune is a graph database optimized for traversing deeply connected relationships. Use DynamoDB for session data, caching. Use Neptune when relationships between entities matter (fraud, social graph).

Q2: What query languages does Neptune support?

Neptune supports three: (1) Apache TinkerPop Gremlin — property graph traversal, (2) SPARQL 1.1 — RDF/semantic web queries, and (3) openCypher — Cypher-compatible for Property Graph. Gremlin is most widely used.

Q3: How does Neptune achieve High Availability?

Neptune replicates storage 6 times across 3 Availability Zones. If the Primary fails, Neptune promotes a Read Replica to Primary in under 30 seconds. The cluster endpoint automatically redirects traffic to the new Primary.

Q4: What is Neptune Serverless?

Neptune Serverless automatically scales the compute capacity (measured in Neptune Capacity Units, NCUs) based on actual workload demand. Minimum 1 NCU, maximum 128 NCU. Ideal for intermittent or unpredictable workloads where you want to avoid paying for idle instances.

Q5: How would you detect fraud using Neptune?

Model accounts as Vertices and transactions as Edges. Run Gremlin traversals to detect: (1) Circular transaction patterns (cycles), (2) Accounts with abnormally high out-degree (money mules), (3) Shared PII across accounts. Neptune ML can predict fraudulent accounts using Graph Neural Networks.

Q6: What is Neptune Streams?

Neptune Streams is a change data capture (CDC) feature that provides a complete, ordered sequence of every change made to the graph. Consumers (Lambda, Kinesis) can read from the stream to keep downstream systems in sync in real time.

Q7: How do you load bulk data into Neptune?

Use the Neptune Bulk Loader — load data from S3 in CSV (Property Graph) or Turtle/N-Quads (RDF) format. The bulk loader is fast (parallel S3 reads), bypasses transaction overhead, and is the recommended approach for initial data loads of millions of records.

Q8: Neptune vs Neo4j — when would you choose Neptune?

Choose Neptune when: you are already on AWS and want managed infrastructure, need AWS IAM/KMS/VPC integration, require Multi-AZ HA out of the box, need SPARQL for RDF workloads. Choose Neo4j for advanced graph algorithms (built-in GDS library), richer Cypher tooling, or on-prem deployments.

■■ When NOT to Use Neptune

Neptune is not the right tool for every use case. Avoid Neptune when:

- ■ Your data has no meaningful relationships — use DynamoDB or RDS instead
- ■ You need ACID transactions across large batches — Neptune has per-request atomicity, not distributed transactions
- ■ You require full SQL analytics — use Redshift or Athena for analytical queries
- ■ Your team has no graph query language (Gremlin/SPARQL) expertise — consider the learning curve
- ■ Ultra-low-cost storage of flat records — graph storage overhead is higher than tabular storage
- ■ Real-time streaming ingestion at millions of events/sec — use Kinesis + DynamoDB for raw event storage

■ Quick Reference Summary

Attribute	Value

Engine Type	Managed Graph Database (Property Graph + RDF)
Query Languages	Gremlin, SPARQL 1.1, openCypher
Storage	Auto-grow, up to 128 TiB, 6-way replicated, 3 AZs
Replication Lag	< 10ms typical read replica lag
Failover Time	< 30 seconds automated failover
Max Replicas	15 Read Replicas per cluster
Backup	Continuous to S3, 1–35 day PITR
Encryption	KMS at-rest, TLS in-transit
Auth	IAM DB Authentication (token-based)
Serverless Range	1 to 128 Neptune Capacity Units (NCUs)
Best Use Cases	Fraud, Social Graphs, Knowledge Graphs, Recommendation, Supply Chain
Not Ideal For	Flat tabular data, pure analytical workloads, high-volume raw event streams
Pricing Model	Instance Hours + I/O + Storage + Backup

