Table of Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is Neptune?</td>
<td>1</td>
</tr>
<tr>
<td>Amazon Neptune Features</td>
<td>2</td>
</tr>
<tr>
<td>Performance and Scalability</td>
<td>2</td>
</tr>
<tr>
<td>High Availability and Durability</td>
<td>2</td>
</tr>
<tr>
<td>Support for Open Graph APIs</td>
<td>2</td>
</tr>
<tr>
<td>Enhanced Data Security</td>
<td>2</td>
</tr>
<tr>
<td>Fully Managed Service</td>
<td>2</td>
</tr>
<tr>
<td>What is a Graph Database?</td>
<td>3</td>
</tr>
<tr>
<td>Graph Database Uses</td>
<td>4</td>
</tr>
<tr>
<td>Graph Queries and Traversals</td>
<td>7</td>
</tr>
<tr>
<td>Quick Start</td>
<td>10</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>10</td>
</tr>
<tr>
<td>Creating a Neptune Cluster</td>
<td>10</td>
</tr>
<tr>
<td>Accessing the Neptune Graph</td>
<td>11</td>
</tr>
<tr>
<td>Getting Started</td>
<td>13</td>
</tr>
<tr>
<td>Setting Up</td>
<td>13</td>
</tr>
<tr>
<td>Neptune VPC Requirements</td>
<td>13</td>
</tr>
<tr>
<td>Creating a Security Group to Provide Access to the Neptune DB Instance in the VPC</td>
<td>15</td>
</tr>
<tr>
<td>Launching a DB Cluster</td>
<td>16</td>
</tr>
<tr>
<td>Launch a Neptune DB Cluster Using the Console</td>
<td>16</td>
</tr>
<tr>
<td>Accessing a Graph</td>
<td>19</td>
</tr>
<tr>
<td>Finding the Endpoint</td>
<td>19</td>
</tr>
<tr>
<td>Launch an EC2 Instance</td>
<td>20</td>
</tr>
<tr>
<td>Gremlin</td>
<td>21</td>
</tr>
<tr>
<td>Neptune Gremlin Implementation Differences</td>
<td>22</td>
</tr>
<tr>
<td>Loading an Example Graph</td>
<td>27</td>
</tr>
<tr>
<td>Gremlin Console</td>
<td>29</td>
</tr>
<tr>
<td>HTTP REST</td>
<td>31</td>
</tr>
<tr>
<td>Java</td>
<td>31</td>
</tr>
<tr>
<td>Python</td>
<td>34</td>
</tr>
<tr>
<td>.NET</td>
<td>35</td>
</tr>
<tr>
<td>Node.js</td>
<td>37</td>
</tr>
<tr>
<td>Gremlin HTTP and WebSocket API</td>
<td>38</td>
</tr>
<tr>
<td>Next Steps</td>
<td>38</td>
</tr>
<tr>
<td>SPARQL</td>
<td>39</td>
</tr>
<tr>
<td>Loading an Example Graph</td>
<td>39</td>
</tr>
<tr>
<td>RDF4J Console</td>
<td>41</td>
</tr>
<tr>
<td>HTTP REST</td>
<td>42</td>
</tr>
<tr>
<td>Java</td>
<td>43</td>
</tr>
<tr>
<td>SPARQL HTTP API</td>
<td>46</td>
</tr>
<tr>
<td>Next Steps</td>
<td>46</td>
</tr>
<tr>
<td>SSL Settings</td>
<td>47</td>
</tr>
<tr>
<td>Loading Data into Neptune</td>
<td>48</td>
</tr>
<tr>
<td>Prerequisites: IAM and Amazon S3</td>
<td>49</td>
</tr>
<tr>
<td>Creating an IAM Policy for S3 Access</td>
<td>49</td>
</tr>
<tr>
<td>Creating an IAM Role to Access AWS Services</td>
<td>50</td>
</tr>
<tr>
<td>Adding the IAM Role to a Cluster</td>
<td>51</td>
</tr>
<tr>
<td>Load Data Formats</td>
<td>51</td>
</tr>
<tr>
<td>Gremlin Load Data Format</td>
<td>52</td>
</tr>
<tr>
<td>RDF Load Data Formats</td>
<td>56</td>
</tr>
<tr>
<td>Example: Loading Data</td>
<td>57</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>57</td>
</tr>
<tr>
<td>Neptune Loader API Reference</td>
<td>59</td>
</tr>
<tr>
<td>Loader Command</td>
<td>59</td>
</tr>
</tbody>
</table>

API Version 2017-11-29

iii
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loader Get Status</td>
<td>63</td>
</tr>
<tr>
<td>Loader Cancel Job</td>
<td>68</td>
</tr>
<tr>
<td>DB Instance Lifecycle</td>
<td>70</td>
</tr>
<tr>
<td>Backing Up and Restoring</td>
<td>71</td>
</tr>
<tr>
<td>Working with Backups</td>
<td>71</td>
</tr>
<tr>
<td>Creating a Snapshot</td>
<td>74</td>
</tr>
<tr>
<td>DB Parameter Groups</td>
<td>76</td>
</tr>
<tr>
<td>Edit a DB Parameter Group</td>
<td>76</td>
</tr>
<tr>
<td>Create a DB Parameter Group</td>
<td>77</td>
</tr>
<tr>
<td>Modifying a DB Instance</td>
<td>78</td>
</tr>
<tr>
<td>Impact of Apply Immediately</td>
<td>78</td>
</tr>
<tr>
<td>Common Settings and Downtime Notes</td>
<td>78</td>
</tr>
<tr>
<td>Renaming a DB Instance</td>
<td>81</td>
</tr>
<tr>
<td>Renaming a DB Instance Using the Console</td>
<td>81</td>
</tr>
<tr>
<td>Rebooting a DB Instance</td>
<td>82</td>
</tr>
<tr>
<td>Rebooting a DB Instance Using the Console</td>
<td>82</td>
</tr>
<tr>
<td>Deleting a DB Instance</td>
<td>83</td>
</tr>
<tr>
<td>Deleting a DB Instance Using the Console</td>
<td>83</td>
</tr>
<tr>
<td>Encrypting Neptune Resources</td>
<td>86</td>
</tr>
<tr>
<td>Enabling Encryption</td>
<td>86</td>
</tr>
<tr>
<td>Neptune Limits</td>
<td>88</td>
</tr>
</tbody>
</table>
What Is Amazon Neptune?

Amazon Neptune is a fast, reliable, fully managed graph database service that makes it easy to build and run applications that work with highly connected datasets. The core of Neptune is a purpose-built, high-performance graph database engine that is optimized for storing billions of relationships and querying the graph with milliseconds latency. Neptune supports the popular graph query languages Apache TinkerPop Gremlin and W3C’s SPARQL, allowing you to easily build queries that efficiently navigate highly connected datasets. Neptune powers graph use cases such as recommendation engines, fraud detection, knowledge graphs, drug discovery, and network security.

Neptune is highly available, with read replicas, point-in-time recovery, continuous backup to Amazon S3, and replication across Availability Zones. Neptune provides data security features, with support for encryption at rest and in transit. Neptune is fully managed, so you no longer need to worry about database management tasks like hardware provisioning, software patching, setup, configuration, or backups.

To learn about using Amazon Neptune, we recommend that you start with the following sections:

- What Is a Graph Database? (p. 3)
- Amazon Neptune Quick Start (p. 10)
- Getting Started with Neptune (p. 13)

Supports Open Graph APIs

Amazon Neptune supports open graph APIs for both Gremlin and SPARQL, and it provides high performance for both of these graph models and their query languages. You can choose the Property Graph (PG) model and its open source query language, Apache TinkerPop Gremlin graph traversal language, or you can use the W3C standard Resource Description Framework (RDF) model and its standard SPARQL Query Language.

Highly Secure

Neptune provides multiple levels of security for your database, including network isolation using Amazon VPC, encryption at rest using keys that you create and control through AWS Key Management Service (AWS KMS), and encryption of data in transit using Transport Layer Security (TLS). On an encrypted Neptune instance, data in the underlying storage is encrypted, as are the automated backups, snapshots, and replicas in the same cluster.

Fully Managed

With Amazon Neptune, you don’t have to worry about database management tasks like hardware provisioning, software patching, setup, configuration, or backups.

You can use Neptune to create sophisticated, interactive graph applications that can query billions of relationships in milliseconds. SQL queries for highly connected data are complex and hard to tune for performance. Instead, Neptune allows you to use the popular graph query languages TinkerPop Gremlin and SPARQL to execute powerful queries that are easy to write and perform well on connected data. This significantly reduces code complexity and enables you to more quickly create applications that process relationships.
Neptune is designed to offer greater than 99.99 percent availability. It increases database performance and availability by tightly integrating the database engine with an SSD-backed virtualized storage layer that is built for database workloads. Neptune storage is fault-tolerant and self-healing, and disk failures are repaired in the background without loss of database availability. Neptune automatically detects database crashes and restarts without the need for crash recovery or rebuilding the database cache. If the entire instance fails, Neptune automatically fails over to one of up to 15 read replicas.

Amazon Neptune Features

Neptune provides the following basic features and capabilities.

Performance and Scalability

Amazon Neptune is a high-performance graph database service that is optimized for processing graph queries. Neptune supports up to 15 low-latency read replicas across three Availability Zones to scale read capacity and execute more than 100,000 graph queries per second. You can easily scale your database deployment up and down from smaller to larger instance types as your needs change.

High Availability and Durability

Neptune is highly available and durable and is designed to provide greater than 99.99 percent availability. It features fault-tolerant and self-healing storage built for the cloud that replicates six copies of your data across three Availability Zones. Neptune continuously backs up your data to Amazon S3 and transparently recovers from physical storage failures. For high availability, instance failover typically takes less than 30 seconds.

Support for Open Graph APIs

Neptune supports open graph APIs for both Gremlin and SPARQL, and it provides high performance for both of these graph models and their query languages. You can choose the PG model and its open source query language TinkerPop Gremlin, or the RDF model and its standard query language SPARQL.

Enhanced Data Security

Amazon Neptune provides multiple levels of security for your database, including network isolation using Amazon VPC, encryption at rest using keys that you create and control through AWS KMS, and encryption of data in transit using TLS. On an encrypted Neptune instance, data in the underlying storage is encrypted, as are the automated backups, snapshots, and replicas in the same cluster.

Fully Managed Service

You don't have to worry about database management tasks like hardware provisioning, software patching, setup, configuration, or backups. Neptune automatically and continuously monitors and backs up your database to Amazon S3, enabling granular point-in-time recovery.
What Is a Graph Database?

| Preview Release | Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page. |

Topics

- Graph Database Uses (p. 4)
- Graph Queries and Traversals (p. 7)

Graph databases like Amazon Neptune are purpose-built to store and navigate relationships. Graph databases have advantages over relational databases for certain use cases—including social networking, recommendation engines, and fraud detection—when you want to create relationships between data and quickly query these relationships. There are a number of challenges to building these types of applications using a relational database. It requires you to have multiple tables with multiple foreign keys. The SQL queries to navigate this data require nested queries and complex joins that quickly become unwieldy. And the queries don't perform well as your data size grows over time.

Neptune uses graph structures such as nodes (data entities), edges (relationships), and properties to represent and store data. The relationships are stored as first-order citizens of the data model. This condition allows data in nodes to be directly linked, dramatically improving the performance of queries that navigate relationships in the data. The interactive performance at scale in Neptune effectively enables a broad set of graph use cases.

A graph in a graph database can be traversed along specific edge types, or across the entire graph.

Graph databases can represent how entities relate by using actions, ownership, parentage, and so on. Whenever connections or relationships between entities are at the core of the data that you’re trying to model, a graph database is a natural choice. Therefore, graph databases are useful for modeling and querying social networks, business relationships, dependencies, shipping movements, and similar items.

You can use edges to show typed relationships between entities (also called vertices or nodes). Edges can describe parent-child relationships, actions, product recommendations, purchases, and so on. A relationship, or edge, is a connection between two vertices that always has a start node, end node, type, and direction.

An example of a common use case that is suited to a graph is social networking data. Amazon Neptune can quickly and easily process large sets of user profiles and interactions to build social networking applications. Neptune enables highly interactive graph queries with high throughput to bring social features into your applications. For example, suppose that you want to build a social feed into your application. You can use Neptune to provide results that prioritize showing your users the latest updates from their family, from friends whose updates they “Like,” and from friends who live close to them.

Following is an example of a social network graph.
This example models a group of friends and their hobbies as a graph. A simple traversal of this graph can tell you what Justin’s friends like.

**Graph Database Uses**

Graph databases are useful for connected, contextual, relationship-driven data. An example is modeling social media data, as shown in the previous section. Other examples include recommendation engines, driving directions (route finding), logistics, diagnostics, and scientific data analysis in fields like neuroscience.

**Fraud Detection**

Another use case for graph databases is detecting fraud. For example, you can track credit card purchases and purchase locations to detect uncharacteristic use. Detecting fraudulent accounts is another example.

With Amazon Neptune, you can use relationships to process financial and purchase transactions in near-real time to easily detect fraud patterns. Neptune provides a fully managed service to execute fast graph queries to detect that a potential purchaser is using the same email address and credit card as a known fraud case. If you are building a retail fraud detection application, Neptune can help you build
graph queries to easily detect relationship patterns like multiple people associated with a personal email address, or multiple people sharing the same IP address but residing in different physical addresses.

The following graph shows the relationship of three people and their identity-related information. Each person has an address, a bank account, and a social security number. However, we can see that Matt and Justin share the same social security number, which is irregular and indicates possible fraud by one or more of the connected people. A query to the graph database could help you discover these types of connections so that they can be reviewed.
Recommendation Engines

With Amazon Neptune, you can store relationships between information categories such as customer interests, friends, and purchase history in a graph. You can then quickly query it to make recommendations that are personalized and relevant. For example, you can use a highly available graph
database to make product recommendations to a user based on which products are purchased by others who follow the same sport and have similar purchase history. Or, you can identify people who have a friend in common, but don't yet know each other, and make a friendship recommendation.

**Knowledge Graphs**

Amazon Neptune helps you build knowledge graph applications. A knowledge graph lets you store information in a graph model and use graph queries to help your users navigate highly connected datasets more easily. Neptune supports open source and open standard APIs so that you can quickly use existing information resources to build your knowledge graphs and host them on a fully managed service. For example, if a user is interested in the *Mona Lisa* by Leonardo da Vinci, you can help them discover other works of art by the same artist or other works located in The Louvre. Using a knowledge graph, you can add topical information to product catalogs, build and query complex models of regulatory rules, or model general information, like Wikidata.

**Life Sciences**

Amazon Neptune helps you build applications that store and navigate information in the life sciences, and process sensitive data easily using encryption at rest. For example, you can use Neptune to store models of disease and gene interactions, and search for graph patterns within protein pathways to find other genes that may be associated with a disease. You can model chemical compounds as a graph and query for patterns in molecular structures. Neptune helps you integrate information to tackle challenges in healthcare and life sciences research. You can use Neptune to create and store patient relationships from medical records across different systems and topically organize research publications to find relevant information quickly.

**Network / IT Operations**

You can use Amazon Neptune to store a graph of your network and use graph queries to answer questions like how many hosts are running a specific application. Neptune can store and process billions of events to manage and secure your network. If you detect an event, you can use Neptune to quickly understand how it might affect your network by querying for a graph pattern using the attributes of the event. You can issue graph queries to Neptune to find other hosts or devices that may be compromised. For example, if you detect a malicious file on a host, Neptune can help you find the connections between the hosts that spread the malicious file and enable you to trace it to the original host that downloaded it.

**Graph Queries and Traversals**

Neptune supports two different graph query languages: Gremlin ([Apache TinkerPop3](https://tinkerpop.apache.org)) and SPARQL ([SPARQL 1.1](https://www.w3.org/TR/rdf-sparql-query/)).

- Gremlin is a graph traversal language and, as such, a query in Gremlin is a traversal made up of discrete steps. Each step follows an edge to a node.
- SPARQL is a declarative query language based on graph pattern-matching standardized by the W3C.

Given the following graph of people (nodes) and their relationships (edges), you can find out who the "friends of friends" of a particular person are—for example, the friends of Howard's friends.
Looking at the graph, you can see that Howard has one friend, Jack, and Jack has three friends: Annie, Harry, and Mac. This is a simple example with a simple graph, but these types of queries can scale in complexity, dataset size, and result size.

The following is a Gremlin traversal query that returns the names of the friends of Howard's friends.

```plaintext
g.V().has('name', 'Howard').out('friend').out('friend').values('name')
```
The following is a SPARQL query that returns the names of the friends of Howard's friends.

**Note**
Each part of any Resource Description Framework (RDF) triple has a URI associated with it. In this example, the URI prefix is intentionally short. For more information, see Accessing the Neptune Graph with SPARQL (p. 39).

```sparql
prefix : <#>

select ?names where {
  ?howard :name "Howard" .
}
```

For more examples of Gremlin and SPARQL queries, see Accessing a Neptune Graph (p. 19).
Amazon Neptune Quick Start

Prerequisites

Before you create an Amazon Neptune cluster, you need the following items in the US East (N. Virginia) Region.

- An Amazon Virtual Private Cloud (Amazon VPC). The default VPC will work. The Neptune console can create a VPC for you when you create a Neptune cluster.
- A Neptune DB subnet group with at least two subnets, with each subnet in a different Availability Zone. You can create a DB subnet group in the Neptune console at https://yukon.aws.amazon.com/neptune?region=us-east-1.

  Note
  Amazon Neptune is not supported in every Availability Zone. If you receive the console error "DB Subnet Group doesn't meet availability zone coverage requirement", try adding subnets in additional Availability Zones to the DB subnet group.
- An Amazon Elastic Compute Cloud (Amazon EC2) instance in the VPC.
  Important
  Access to the Neptune cluster from outside the VPC is disabled.
- A security group that allows SSH (port 22) access to the Amazon EC2 instance.
- A security group that allows TCP access to the Neptune port (the default is 8182) from the Amazon EC2 IP or its security group.

  Note
  This can be one single security group.
- An AWS Identity and Access Management (IAM) user with AmazonRDSFullAccess permissions. These permissions are required to use the Neptune Preview console and create a Neptune cluster. For information about adding these permissions, see AWS Managed (Predefined) Policies.
- (Loading only) An Amazon Simple Storage Service (Amazon S3) bucket in the US East (N. Virginia) Region.
- (Loading only) An Amazon S3 VPC endpoint. For more information see, Amazon S3 VPC Endpoint (p. 57).

For detailed instructions and information about creating these items, see Getting Started with Neptune (p. 13).

Creating a Neptune Cluster

Accessing the Neptune Graph

1. Connect to your Amazon EC2 instance via SSH.
2. Query the endpoint for either Gremlin or SPARQL.

   **Note**
   The first access to a Neptune DB instance sets the query engine mode to either Gremlin or SPARQL. When you access either the Gremlin or SPARQL endpoint on a Neptune DB instance, the query engine is set.
   If the first access to your Neptune DB instance is a bulk load request, the csv format sets the query engine to Gremlin. The ntriples, nquads, rdfxml, or turtle formats set the query engine to SPARQL.

   **For Gremlin:**
   To query the Gremlin graph, type the following command, replacing `your-neptune-endpoint` with the **Cluster endpoint** from the previous section:

   ```bash
   curl -X POST -d '{"gremlin":"g.V()"}' http://your-neptune-endpoint:8182/gremlin
   ```
   
The graph is empty, so the result value has no data. The response looks like the following:

   ```json
   {"requestId":"43aae48d-5807-40e7-87bd-c92c2df99e9","status":
   {"message":null,"code":200,"attributes":null},"result":
   {"data":null,"meta":null}}
   ```

   **For SPARQL:**
   To query the SPARQL endpoint, type the following command, replacing `your-neptune-endpoint` with the **Cluster endpoint** from the previous section:

   ```bash
   ```
   
The graph is empty, so the result value has no data. The response looks like the following:

   ```xml
   <sparql xmlns='http://www.w3.org/2005/sparql-results#'>
   <head>
   <variable name='s'/>
   <variable name='p'/>
   <variable name='o'/>
   </head>
   <results/>
   </sparql>
   ```

API Version 2017-11-29
For detailed information about connecting to the Neptune graph, see Accessing a Neptune Graph (p. 19).

For general information about Neptune, see What Is Amazon Neptune? (p. 1).

For information about loading data into Neptune, see Loading Data into Neptune (p. 48).
Getting Started with Neptune

This section provides details on the requirements and prerequisites for Amazon Neptune and shows you how to use it to create a Neptune DB instance. For a less detailed overview, see the Amazon Neptune Quick Start (p. 10).

Topics
- Setting Up Amazon Neptune (p. 13)
- Launching a Neptune DB Cluster (p. 16)

Setting Up Amazon Neptune

Before you create a Neptune DB instance, you must have an Amazon Virtual Private Cloud (VPC). If you want to access your Neptune DB instance from outside the VPC, you must also have a security group for the VPC with rules that allow you to connect to the Neptune DB instance.

You also need an IAM user with AmazonRDSFullAccess permissions. This is required to use the Neptune Beta console and create a Neptune cluster. For information about adding these permissions, see AWS Managed (Predefined) Policies.

Neptune VPC Requirements

If you created your AWS account after 2013-12-04, then you have a default VPC in each AWS Region. If you aren't sure whether you have a default VPC, see the Detecting Whether You Have a Default VPC section in the Amazon VPC User Guide.

For more information about the default VPC, see Default VPC and Default Subnets in the Amazon VPC User Guide.

If you have a default VPC, you can create a VPC security group to allow an Amazon EC2 instance to connect to the Neptune DB instance from within the VPC. Access from the internet is allowed only to the EC2 instance. The EC2 instance is allowed access to the graph database.
There are many possible ways to configure a VPC or multiple VPCs. For information about creating your own VPCs, see the [Amazon VPC User Guide](https://docs.aws.amazon.com/vpc/).

An Amazon Neptune DB cluster can only be created in an Amazon VPC that has at least two subnets in at least two Availability Zones. By distributing your cluster instances across at least two Availability Zones, you help ensure that there are instances available in your DB cluster in the unlikely event of an Availability Zone failure. The cluster volume for your Neptune DB cluster always spans three Availability Zones to provide durable storage with less possibility of data loss.

If you're using the Amazon Neptune console to create your Neptune DB cluster, you can have Neptune automatically create a VPC for you. Alternatively, you can use an existing VPC or create a new VPC for your Neptune DB cluster. Your VPC must have at least two subnets in order for you to use it with an Amazon Neptune DB cluster.

**Note**

You can communicate with an Amazon EC2 instance that is not in a VPC and a Neptune DB cluster using ClassicLink.

If you don't have a default VPC, and you have not created a VPC, you can have Neptune automatically create a VPC for you when you create a Neptune DB cluster using the console. Neptune can also create a VPC security group and a DB subnet group for you.

Otherwise, you must do the following:

- Create a VPC with at least two subnets in at least two Availability Zones.
- Specify a VPC security group that authorizes connections to your Neptune DB cluster. You can do this in the Amazon VPC console at [https://console.aws.amazon.com/vpc/](https://console.aws.amazon.com/vpc/).
• Specify a Neptune DB subnet group with at least two subnets with each subnet in a different Availability Zone. You can create a DB subnet group in the Neptune console at https://yukon.aws.amazon.com/neptune?region=us-east-1.

   **Note**
   Amazon Neptune is not supported in every Availability Zone. If you receive the console error **DB Subnet Group doesn't meet availability zone coverage requirement**, try adding subnets in additional Availability Zones to the DB subnet group.

The following section walks you through setting up a security group for your default VPC, as shown in the preceding diagram.

**Creating a Security Group to Provide Access to the Neptune DB Instance in the VPC**

Your Neptune DB instance is launched in a VPC. Security groups provide access to the Neptune DB instance in the VPC. They act as a firewall for the associated Neptune DB instance, controlling both inbound and outbound traffic at the instance level. Neptune DB instances are created by default with a firewall and a default security group that prevents access to the Neptune DB instance. You must add rules to a security group that enable you to connect to your DB instance.

The security group you need to create is a **VPC security group**. Neptune DB instances in a VPC require that you add rules to a VPC security group to allow access to the instance.

The following procedure shows you how to add a custom TCP rule that specifies the port range and IP addresses that the EC2 instance uses to access the database. You can use the VPC security group assigned to the EC2 instance rather than the IP address.

**To create a VPC security group for Neptune**

1. Sign in to the AWS Management Console and open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the upper-right corner of the console, choose the AWS Region in which you want to create the VPC security group and the Neptune DB instance. In the list of Amazon VPC resources for that Region, it should show that you have at least one VPC and several subnets. If it does not, you don't have a default VPC in that Region.
3. In the navigation pane, choose **Security Groups**.
4. Choose **Create Security Group**.
5. In the **Create Security Group** window, type the **Name tag**, **Group name**, and **Description** of your security group. Choose the **VPC** that you want to create your Neptune DB instance in. Choose **Yes, Create**.
6. The VPC security group that you created should still be selected. The details pane at the bottom of the console window displays the details for the security group, and tabs for working with inbound and outbound rules. Choose the **Inbound Rules** tab.
7. On the **Inbound Rules** tab, choose **Edit**. In the **Type** list, choose **Custom TCP Rule**.
8. In the **PortRange** text box, type **8182**, the default port value for a Neptune DB instance. Then type the IP address range (CIDR value) from where you will be accessing the instance, or choose a security group name in the **Source** text box.
9. If you need to add more IP addresses or different port ranges, choose **Add another rule**.
10. When you have finished, choose **Save**.

You will use the VPC security group you just created as the security group for your DB instance when you create it.
Finally, a quick note about VPC subnets: If you use a default VPC, a default subnet group spanning all of the VPC's subnets is already created for you. When you use the Launch a Neptune DB instance wizard to create a DB instance, you can choose the default VPC and use default for the DB Subnet Group.

After you complete the setup requirements, you can use your settings and the security group you created to launch a Neptune DB instance.

Launching a Neptune DB Cluster

**Preview Release**  Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

The following procedures describe how to use the AWS Management Console to launch an Amazon Neptune DB cluster and create a Neptune Replica.

**Launch a Neptune DB Cluster Using the Console**

Before you can access the Neptune Beta console, you need to have an IAM user with AmazonRDSFullAccess permissions. This is required to use the Neptune Beta console and create a Neptune cluster. For information about adding these permissions, see **AWS Managed (Predefined) Policies**.

**To launch a Neptune DB cluster using the console**

2. Choose Launch Instance to start the Launch DB Instance wizard.
3. On the Specify DB details page, you can customize the settings for your Neptune DB cluster. The following table shows the advanced settings for a DB cluster.

<table>
<thead>
<tr>
<th>For this option...</th>
<th>Do this</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DB Instance Class info</strong></td>
<td>Choose a DB instance class that defines the processing and memory requirements for each instance in the DB cluster.</td>
</tr>
<tr>
<td><strong>DB Instance Identifier</strong></td>
<td>Type a name for the primary instance in your DB cluster. This identifier is used in the endpoint address for the primary instance of your DB cluster. The DB instance identifier has the following constraints:</td>
</tr>
<tr>
<td></td>
<td>- It must contain from 1 to 63 alphanumeric characters or hyphens.</td>
</tr>
<tr>
<td></td>
<td>- Its first character must be a letter.</td>
</tr>
<tr>
<td></td>
<td>- It cannot end with a hyphen or contain two consecutive hyphens.</td>
</tr>
<tr>
<td></td>
<td>- It must be unique for all DB instances per AWS account, per AWS Region.</td>
</tr>
</tbody>
</table>
4. On the **Configure Advanced Settings** page, you can customize additional settings for your Neptune DB cluster. The following table shows the advanced settings for a DB cluster.

<table>
<thead>
<tr>
<th>For this option...</th>
<th>Do this</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPC</td>
<td>Choose the VPC that will host the DB cluster. Choose <strong>Create a New VPC</strong> to have Neptune create a VPC for you. You need to create an Amazon EC2 instance in this same VPC to access the Neptune instance. For more information, see Setting Up Amazon Neptune (p. 13).</td>
</tr>
<tr>
<td>Subnet Group</td>
<td>Choose the Neptune DB subnet group to use for the DB cluster. If your VPC does not have any subnet groups, Neptune creates a DB subnet group for you. For more information, see Setting Up Amazon Neptune (p. 13).</td>
</tr>
<tr>
<td>Availability Zone</td>
<td>Specify a particular Availability Zone, or choose <strong>No preference</strong> to have Neptune choose one for you.</td>
</tr>
<tr>
<td>VPC Security Group(s)</td>
<td>Choose one or more VPC security groups to secure network access to the DB cluster. Choose <strong>Create a New VPC Security Group</strong> to have Neptune create a VPC security group for you. For more information, see Setting Up Amazon Neptune (p. 13).</td>
</tr>
<tr>
<td>DB Cluster Identifier</td>
<td>The identifier for your DB cluster. If you don't specify this value, Neptune creates one based on the DB instance identifier.</td>
</tr>
<tr>
<td>Database Port</td>
<td>The port for all HTTP and WebSockets connections. Neptune DB clusters use 8182 as the default.</td>
</tr>
<tr>
<td>Enable Encryption</td>
<td>Choose <strong>Yes</strong> to enable encryption at rest for this DB cluster. For more information, see Encrypting Neptune Resources (p. 86).</td>
</tr>
<tr>
<td>Failover Priority</td>
<td>Choose the priority tier. If there is contention within a tier, the replica that is the same size as the primary instance is selected.</td>
</tr>
<tr>
<td>Backup Retention Period</td>
<td>Choose the length of time, from 1 to 35 days, that Neptune will retain backup copies of the database. Backup copies can be used for point-in-time restores (PITR) of your database down to the second.</td>
</tr>
<tr>
<td>Auto Minor Version Upgrade</td>
<td>Choose <strong>Yes</strong> if you want to enable your Neptune DB cluster to receive minor Neptune DB Engine version upgrades automatically when they become available. The <strong>Auto Minor Version Upgrade</strong> option only applies to upgrades to Neptune minor engine versions for your Amazon Neptune DB cluster. It doesn't apply to regular patches applied to maintain system stability.</td>
</tr>
<tr>
<td>Maintenance Window</td>
<td>Choose the weekly time range during which system maintenance can occur.</td>
</tr>
</tbody>
</table>

5. Choose **Launch DB Instance** to launch your Neptune DB instance, and then choose **Close** to close the wizard.
On the Amazon Neptune console, the new DB cluster appears in the list of DB clusters. The DB cluster has a status of **creating** until it is created and ready for use. When the state changes to **available**, you can connect to the primary instance for your DB cluster. Depending on the DB instance class and store allocated, it can take several minutes for the new instances to be available.

To view the newly created cluster, choose the **Clusters** view in the Neptune console.

Note the **Cluster endpoint** value. You will need this to connect to your Neptune DB cluster.
Accessing a Neptune Graph

Amazon Neptune supports two different graph query languages: Gremlin (Apache TinkerPop3) and SPARQL (SPARQL 1.1). Instructions for accessing the Neptune graph on a running Neptune DB instance are divided into sections for Gremlin and SPARQL.

The first access to a Neptune DB instance sets the query engine mode to either Gremlin or SPARQL. When you access either the Gremlin or SPARQL endpoint on a Neptune DB instance, the query engine is set. If the first access to your Neptune DB instance is a bulk load request, the csv format sets the query engine to Gremlin, and the ntriples, nquads, rdfxml, or turtle format sets the query engine to SPARQL.

Gremlin

Gremlin is a graph traversal language, and as such, a query in Gremlin is a traversal made up of discrete steps. Each step follows an edge to a node.

To learn about connecting to Neptune with Gremlin, see Accessing the Neptune Graph with Gremlin (p. 21).

SPARQL

SPARQL is a declarative query language based on the graph pattern matching that is standardized by the W3C and described in the SPARQL 1.1 Query Language specification.

To learn about connecting to Neptune with SPARQL, see Accessing the Neptune Graph with SPARQL (p. 39).

Topics
- Finding the Endpoint for a Neptune Cluster (p. 19)
- Launching an Amazon EC2 Instance (p. 20)
- Accessing the Neptune Graph with Gremlin (p. 21)
- Accessing the Neptune Graph with SPARQL (p. 39)
- Secure Sockets Layer Settings for a Neptune Cluster (p. 47)

Finding the Endpoint for a Neptune Cluster

To run the examples in this guide, the endpoint for a Neptune cluster is required. The following sections show you how to get this information.

To find the endpoint for a Neptune cluster

2. Choose **Clusters**, and then choose the DB cluster from the list.
3. Choose the **Details** tab to show the DB cluster details. On the **Details** page, copy the value for the **Cluster endpoint**.

![Cluster endpoint information](image)

### Launching an Amazon EC2 Instance

**Preview Release** Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

Access to Neptune is limited to within the virtual private cloud (VPC) that the Neptune DB instance is in. The following steps launch an Amazon Elastic Compute Cloud (Amazon EC2) instance in your default Amazon VPC. If you created a Neptune DB instance in a different VPC, you should launch the EC2 instance in that VPC.

**To launch an EC2 instance**

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the upper-right corner of the console window, choose **US East (N. Virginia)** from the Region selector.
3. Choose **Launch Instance**, and do the following:
   a. Choose an Amazon Machine Image (AMI):
At the top of the list of AMIs, go to Amazon Linux AMI, and choose Select.

b. Choose an Instance Type:
   1. At the top of the list of instance types, choose t2.micro.
   2. Choose Next: Configure Instance Details.

c. Configure Instance Details:
   1. Go to Network, and choose your default VPC.
   2. Choose Next: Add Storage.

d. Add Storage:
   • Skip this step by choosing Next: Tag Instance.

e. Tag Instance:
   • Skip this step by choosing Next: Configure Security Group.

f. Configure Security Group:
   1. Choose Select an existing security group.
   2. In the list of security groups, choose default. This is the default security group for your VPC.

g. Review Instance Launch:
   • Choose Launch.

4. In the Select an existing key pair or create a new key pair window, do one of the following:
   • If you don't have an Amazon EC2 key pair, choose Create a new key pair and follow the instructions. You are asked to download a private key file (.pem file); you need this file when you log in to your Amazon EC2 instance.
   • If you already have an existing Amazon EC2 key pair, go to Select a key pair and choose your key pair from the list. You must already have the private key file (.pem file) available in order to log in to your Amazon EC2 instance.

5. When you have configured your key pair, choose Launch Instances.

6. Return to the Amazon EC2 console home page and choose the instance that you launched. In the lower pane, on the Description tab, find the Public DNS for your instance. For example: ec2-00-00-00-00.us-east-1.compute.amazonaws.com.

Make a note of this public DNS name, because you need it to connect to the instance.

Note
It takes a few minutes for your Amazon EC2 instance to become available. Before you continue, ensure that the Instance State is running and that all of its Status Checks have passed.

Accessing the Neptune Graph with Gremlin

Preview Release  Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

Amazon Neptune is compatible with Apache TinkerPop3 and Gremlin 3.3.0. This means that you can connect to a Neptune DB instance and use the Gremlin traversal language to query the graph.
A traversal in Gremlin is a series of chained steps. It starts at a vertex (or edge) and walks the graph by following the outgoing edges of each vertex and then the outgoing edges of those vertices. Each step is an operation in the traversal. For more information, see The Traversal in the TinkerPop3 documentation.

There are Gremlin language variants and support for Gremlin access in various programming languages. For more information, see On Gremlin Language Variants in the TinkerPop3 documentation.

This documentation describes how to access Neptune with the following variants and programming languages:

**Gremlin-Groovy**

The Gremlin Console and HTTP REST examples in this section use the Gremlin-Groovy variant.

**Gremlin-Java**

The Java sample is written with the official TinkerPop3 Java implementation and uses the Gremlin-Java variant.

**Gremlin-Python**

The Python sample is written with the official TinkerPop3 Python implementation and uses the Gremlin-Python variant.

The following sections walk you through how to use the Gremlin Console, REST over HTTP, and various programming languages to connect to a Neptune DB instance.

Before you begin, you must have the following:

- A Neptune DB instance. For information about creating a Neptune DB instance, see Getting Started with Neptune (p. 13).
- An Amazon EC2 instance in the same virtual private cloud (VPC) as your Neptune DB instance.

For more information about loading data into Neptune, including prerequisites, loading formats, and load parameters, see Loading Data into Neptune (p. 48).

**Topics**

- Neptune Gremlin Implementation Differences (p. 22)
- Loading the TinkerPop Modern Graph (p. 27)
- Using the Gremlin Console to Connect to a Neptune DB Instance (p. 29)
- Using the HTTP REST Endpoint to Connect to a Neptune DB Instance (p. 31)
- Using Java to Connect to a Neptune DB Instance (p. 31)
- Using Python to Connect to a Neptune DB Instance (p. 34)
- Using .NET to Connect to a Neptune DB Instance (p. 35)
- Using Node.js to Connect to a Neptune DB Instance (p. 37)
- Gremlin HTTP and WebSocket API (p. 38)
- Next Steps (p. 38)

**Neptune Gremlin Implementation Differences**

There are a few important differences between the Neptune implementation of Gremlin and the TinkerPop implementation.
**Pre-Bound Variables**

The traversal object \( g \) is Pre-bound. The graph object is not supported.

**Script Execution**

All queries must begin with \( g \).

Multiple queries can be issued separated by a semicolon.

**Sessions**

Neptune is sessionless. It does not support the console session argument. For a description of the difference, see the TinkerPop Session Reference.

**Transactions**

Neptune opens a new transaction at the beginning of each Gremlin traversal and closes the transaction upon the successful completion of the traversal. The transaction is rolled back when there is an error.

Manual transaction logic is not supported. Multiple statements separated by a semicolon are included in a single transaction.

**Vertex and Edge IDs**

Neptune Gremlin Vertex and Edge IDs must be of type String. If you don’t supply an ID when you add a vertex or an edge, a UUID is generated and converted to a string; for example, "48af8178-50ce-971a-fc41-8c9a954cea62".

**Note**

- This means that user-supplied IDs are supported, but they are optional in normal usage.
- However, the Neptune Load command requires that all IDs be specified using the ~id field in the Neptune CSV format.

**User Supplied IDs**

User supplied IDs are allowed in Neptune Gremlin with the following stipulations.

- Supplied IDs are optional.
- Only vertexes and edges are supported.
- Only type String is supported.

**Vertex Property IDs**

Vertex property IDs are generated automatically and can show up as positive or negative numbers when queried.

**Cardinality**

Neptune only supports set cardinality. This means that if you set a property value, it adds a new value to the property, but only if it does not already appear in the set of values. This is the Gremlin enumeration value of Cardinality.Set. Cardinality.List is not supported. For more information about property cardinality, see the Vertex topic in the Gremlin JavaDoc.

**Labels**

Neptune supports multiple labels for a vertex. When you create a label, you can specify multiple labels by separating them with ::. For example, \( g \).addV("Label1::Label2::Label3") adds a vertex
with three different labels. The `hasLabel` step matches this vertex with any of those three labels: `hasLabel("Label1")`, `hasLabel("Label2")`, and `hasLabel("Label3")`.

**Important**
The `::` delimiter is reserved for this use only. You cannot specify multiple labels in the `hasLabel` step. For example, `hasLabel("Label1::Label2")` does not match anything.

**Variables**
Neptune does not support Gremlin variables and does not support the `bindings` property.

**Serialization**
Neptune supports the following serializations based on the requested MIME type.

<table>
<thead>
<tr>
<th>MIME type</th>
<th>Serialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>application/vnd.gremlin-v1.0+gryo</td>
<td>GryoMessageSerializerV1d0</td>
</tr>
<tr>
<td>application/vnd.gremlin-v1.0+gryo-stringd</td>
<td>GryoMessageSerializerV1d0</td>
</tr>
<tr>
<td>application/vnd.gremlin-v3.0+gryo</td>
<td>GryoMessageSerializerV3d0</td>
</tr>
<tr>
<td>application/vnd.gremlin-v3.0+gryo-stringd</td>
<td>GryoMessageSerializerV3d0</td>
</tr>
<tr>
<td>application/vnd.gremlin-v1.0+json</td>
<td>GraphSONMessageSerializerGremlinV1d0</td>
</tr>
<tr>
<td>application/vnd.gremlin-v2.0+json</td>
<td>GraphSONMessageSerializerGremlinV2d0</td>
</tr>
<tr>
<td>application/json</td>
<td>GraphSONMessageSerializerV3d0</td>
</tr>
</tbody>
</table>

**Other Features**
The Neptune implementation of Gremlin does not expose the `graph` object, so the supported and unsupported `graph` features are described in the following section.

**Gremlin Graph Supported Features**
Here is a set of features as implemented by the Neptune Gremlin graph. These features are the same as would be returned by the `graph.features()` command.

<table>
<thead>
<tr>
<th>Graph Feature</th>
<th>Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transactions</td>
<td>true</td>
</tr>
<tr>
<td>ThreadedTransactions</td>
<td>false</td>
</tr>
<tr>
<td>Computer</td>
<td>false</td>
</tr>
<tr>
<td>Persistence</td>
<td>true</td>
</tr>
<tr>
<td>ConcurrentAccess</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable Feature</th>
<th>Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables</td>
<td>false</td>
</tr>
<tr>
<td>Feature</td>
<td>Value</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>SerializableValues</td>
<td>false</td>
</tr>
<tr>
<td>UniformListValues</td>
<td>false</td>
</tr>
<tr>
<td>BooleanArrayValues</td>
<td>false</td>
</tr>
<tr>
<td>DoubleArrayValues</td>
<td>false</td>
</tr>
<tr>
<td>IntegerArrayValues</td>
<td>false</td>
</tr>
<tr>
<td>StringArrayValues</td>
<td>false</td>
</tr>
<tr>
<td>BooleanValues</td>
<td>false</td>
</tr>
<tr>
<td>ByteValues</td>
<td>false</td>
</tr>
<tr>
<td>DoubleValues</td>
<td>false</td>
</tr>
<tr>
<td>FloatValues</td>
<td>false</td>
</tr>
<tr>
<td>IntegerValues</td>
<td>false</td>
</tr>
<tr>
<td>LongValues</td>
<td>false</td>
</tr>
<tr>
<td>MapValues</td>
<td>false</td>
</tr>
<tr>
<td>MixedListValues</td>
<td>false</td>
</tr>
<tr>
<td>StringValues</td>
<td>false</td>
</tr>
<tr>
<td>ByteArrayValues</td>
<td>false</td>
</tr>
<tr>
<td>FloatArrayValues</td>
<td>false</td>
</tr>
<tr>
<td>LongArrayValues</td>
<td>false</td>
</tr>
<tr>
<td>Vertex Feature</td>
<td>Enabled</td>
</tr>
<tr>
<td>MetaProperties</td>
<td>false</td>
</tr>
<tr>
<td>DuplicateMultiProperties</td>
<td>false</td>
</tr>
<tr>
<td>AddVertices</td>
<td>true</td>
</tr>
<tr>
<td>RemoveVertices</td>
<td>true</td>
</tr>
<tr>
<td>MultiProperties</td>
<td>true</td>
</tr>
<tr>
<td>UserSuppliedIds</td>
<td>true</td>
</tr>
<tr>
<td>AddProperty</td>
<td>true</td>
</tr>
<tr>
<td>RemoveProperty</td>
<td>true</td>
</tr>
<tr>
<td>NumericIds</td>
<td>false</td>
</tr>
<tr>
<td>StringIds</td>
<td>true</td>
</tr>
<tr>
<td>UuidIds</td>
<td>false</td>
</tr>
<tr>
<td>CustomIds</td>
<td>false</td>
</tr>
<tr>
<td>AnyIds</td>
<td>false</td>
</tr>
<tr>
<td>Property</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Vertex Property Feature</td>
<td>Enabled</td>
</tr>
<tr>
<td>UserSuppliedIds</td>
<td>false</td>
</tr>
<tr>
<td>AddProperty</td>
<td>true</td>
</tr>
<tr>
<td>RemoveProperty</td>
<td>true</td>
</tr>
<tr>
<td>NumericIds</td>
<td>true</td>
</tr>
<tr>
<td>StringIds</td>
<td>true</td>
</tr>
<tr>
<td>UuidIds</td>
<td>false</td>
</tr>
<tr>
<td>CustomIds</td>
<td>false</td>
</tr>
<tr>
<td>AnyIds</td>
<td>false</td>
</tr>
<tr>
<td>Properties</td>
<td>true</td>
</tr>
<tr>
<td>SerializableValues</td>
<td>false</td>
</tr>
<tr>
<td>UniformListValues</td>
<td>false</td>
</tr>
<tr>
<td>BooleanArrayValues</td>
<td>false</td>
</tr>
<tr>
<td>DoubleArrayValues</td>
<td>false</td>
</tr>
<tr>
<td>IntegerArrayValues</td>
<td>false</td>
</tr>
<tr>
<td>StringArrayValues</td>
<td>false</td>
</tr>
<tr>
<td>BooleanValues</td>
<td>true</td>
</tr>
<tr>
<td>ByteValues</td>
<td>true</td>
</tr>
<tr>
<td>DoubleValues</td>
<td>true</td>
</tr>
<tr>
<td>FloatValues</td>
<td>true</td>
</tr>
<tr>
<td>IntegerValues</td>
<td>true</td>
</tr>
<tr>
<td>LongValues</td>
<td>true</td>
</tr>
<tr>
<td>MapValues</td>
<td>false</td>
</tr>
<tr>
<td>MixedListValues</td>
<td>false</td>
</tr>
<tr>
<td>StringValues</td>
<td>true</td>
</tr>
<tr>
<td>ByteArrayValues</td>
<td>false</td>
</tr>
<tr>
<td>FloatArrayValues</td>
<td>false</td>
</tr>
<tr>
<td>LongArrayValues</td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge Feature</td>
<td>Enabled</td>
</tr>
<tr>
<td>AddEdges</td>
<td>true</td>
</tr>
<tr>
<td>RemoveEdges</td>
<td>true</td>
</tr>
<tr>
<td>Feature</td>
<td>Setting</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------</td>
</tr>
<tr>
<td>UserSuppliedIds</td>
<td>true</td>
</tr>
<tr>
<td>AddProperty</td>
<td>true</td>
</tr>
<tr>
<td>RemoveProperty</td>
<td>true</td>
</tr>
<tr>
<td>NumericIds</td>
<td>false</td>
</tr>
<tr>
<td>StringIds</td>
<td>true</td>
</tr>
<tr>
<td>UuidIds</td>
<td>false</td>
</tr>
<tr>
<td>CustomIds</td>
<td>false</td>
</tr>
<tr>
<td>AnyIds</td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Feature</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge Property Feature</td>
<td>Enabled</td>
</tr>
<tr>
<td>Properties</td>
<td>true</td>
</tr>
<tr>
<td>SerializableValues</td>
<td>false</td>
</tr>
<tr>
<td>UniformListValues</td>
<td>false</td>
</tr>
<tr>
<td>BooleanArrayValues</td>
<td>false</td>
</tr>
<tr>
<td>DoubleArrayValues</td>
<td>false</td>
</tr>
<tr>
<td>IntegerArrayValues</td>
<td>false</td>
</tr>
<tr>
<td>StringArrayValues</td>
<td>false</td>
</tr>
<tr>
<td>BooleanValues</td>
<td>true</td>
</tr>
<tr>
<td>ByteValues</td>
<td>true</td>
</tr>
<tr>
<td>DoubleValues</td>
<td>true</td>
</tr>
<tr>
<td>FloatValues</td>
<td>true</td>
</tr>
<tr>
<td>IntegerValues</td>
<td>true</td>
</tr>
<tr>
<td>LongValues</td>
<td>true</td>
</tr>
<tr>
<td>MapValues</td>
<td>false</td>
</tr>
<tr>
<td>MixedListValues</td>
<td>false</td>
</tr>
<tr>
<td>StringValues</td>
<td>true</td>
</tr>
<tr>
<td>ByteArrayValues</td>
<td>false</td>
</tr>
<tr>
<td>FloatArrayValues</td>
<td>false</td>
</tr>
<tr>
<td>LongArrayValues</td>
<td>false</td>
</tr>
</tbody>
</table>

Next Step: Loading the TinkerPop Modern Graph (p. 27)

Loading the TinkerPop Modern Graph

The following is a visual representation of the TinkerPop modern graph:
Important
Before you load data from Amazon S3, you must create an Amazon S3 VPC endpoint in your VPC. For information about creating an endpoint, see Amazon S3 VPC Endpoint (p. 57). For information about the limitations of VPC endpoints, Endpoints for S3.

Sample data is available in an Amazon S3 bucket.

Run the following command to load the TinkerPop modern graph from the S3 bucket. Replace the endpoint, access key, and secret key placeholders with the appropriate values.

Note
For information about finding the hostname of your Neptune DB instance, see Finding the Endpoint for a Neptune Cluster (p. 19).

```bash
curl -X POST \
-H 'Content-Type: application/json' \
http://your-neptune-endpoint:8182/loader -d ' \
{ 
  "source" : "s3://neptune-us-east-1/tinkerpopmodern/", 
  "format" : "csv", 
  "accessKey" : "access-key-id", 
  "secretKey" : "secret-key", 
```
Now that you have loaded some data, you can access the graph using the method of your choice.

- Using the Gremlin Console to Connect to a Neptune DB Instance (p. 29)
- Using the HTTP REST Endpoint to Connect to a Neptune DB Instance (p. 31)
- Using Java to Connect to a Neptune DB Instance (p. 31)
- Using Python to Connect to a Neptune DB Instance (p. 34)
- Using the Gremlin Console to Connect to a Neptune DB Instance (p. 29)

Using the Gremlin Console to Connect to a Neptune DB Instance

Preview Release  Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

The Gremlin Console allows you to experiment with TinkerPop graphs and queries in a REPL (read-eval-print loop) environment.

You can use the Gremlin Console to connect to a remote graph database. The following section walks you through the configuration of the Gremlin Console to connect remotely to a Neptune DB instance. These instructions must be followed from an Amazon EC2 instance in the same virtual private cloud (VPC) as your Neptune DB instance.

To connect to Neptune using the Gremlin Console

1. The Gremlin Console binaries require Java 8. Type the following to install Java 8 on your EC2 instance.

   ```bash
   sudo yum install java-1.8.0-devel
   ```

2. Type the following to set Java 8 as the default runtime on your EC2 instance.

   ```bash
   sudo /usr/sbin/alternatives --config java
   ```

   When prompted, enter the number for Java 8.

3. Download Gremlin Console (version 3.3.0+) from the Apache Tinkerpop3 website on to your EC2 instance.

4. Unzip the Gremlin Console zip file.

   ```bash
   unzip apache-tinkerpop-gremlin-console-3.3.0-bin.zip
   ```

5. Change directories into the unzipped folder.

   ```bash
   cd apache-tinkerpop-gremlin-console-3.3.0-bin
   ```

6. In the conf subdirectory of the extracted directory, create a file named neptune-remote.yaml with the following text. Replace `your-neptune-endpoint` with the hostname or IP address of your Neptune DB instance. The square brackets ([ ]) are required.
7. In a terminal, navigate to the Gremlin Console directory (apache-tinkerpop-gremlin-console-3.3.1-bin), and then type the following command to run the Gremlin Console.

```
bin/gremlin.sh
```

You should see the following output:

```
\,,,/
(o o)
-----oOOo-(3)-oOOo-----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
plugin activated: tinkerpop.tinkergraph
gremlin>
```

You are now at the `gremlin>` prompt. You will type the remaining steps at this prompt.

8. At the `gremlin>` prompt, type the following to connect to the Neptune DB instance.

```
:remote connect tinkerpop.server conf/neptune-remote.yaml
```

9. At the `gremlin>` prompt, type the following to switch to remote mode. This sends all Gremlin queries to the remote connection.

```
:remote console
```

10. Type the following to run a Gremlin query that returns all the vertices in the graph.

```
g.V().limit(1)
```

11. When you are finished, type the following to exit the Gremlin Console.

```
:exit
```

The preceding example returns the vertex in the graph by using the `g.V().limit(1)` traversal. To query for something else, replace the traversal with another Gremlin traversal.

**Note**

Use a semicolon (`;`) to separate each statement.

For more information about Amazon Neptune, see *Next Steps (p. 38)*.
Using the HTTP REST Endpoint to Connect to a Neptune DB Instance

**Preview Release**  Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

Neptune provides an HTTP endpoint for Gremlin queries. The REST interface is compatible with Gremlin version 3.3.0.

The following instructions walk you through connecting to the Gremlin endpoint using the `curl` command and HTTP. These instructions must be followed from an Amazon EC2 instance in the same virtual private cloud (VPC) as your Neptune DB instance.

The HTTP endpoint for Gremlin queries to a Neptune DB instance is `http://your-neptune:endpoint:8182/gremlin`.

**Note**
For information about finding the hostname of your Neptune DB instance, see Finding the Endpoint for a Neptune Cluster (p. 19).

**To connect to Neptune using the HTTP REST endpoint**

- The following example uses `curl` to submit a Gremlin query through HTTP POST. The query is submitted in JSON format in the body of the post as the `gremlin` property.

```bash
curl -X POST -d '{"gremlin":"g.V().limit(1)"}' http://your-neptune-endpoint:8182/gremlin
```

**Note**
Amazon Neptune does not support the `bindings` property.

You can also send queries through HTTP GET requests, but HTTP POST requests are recommended.

```bash
```

**Important**
The REST endpoint returns all results in a single JSON result set. If the result set is too large, this can cause an `OutOfMemoryError` exception on the Neptune DB instance.

For more information about the Gremlin REST interface, see Connecting via HTTP in the Apache TinkerPop3 documentation.

The preceding example returns the first vertex in the graph by using the `g.V().limit(1)` traversal. To query for something else, replace it with another Gremlin traversal.

For more information about Amazon Neptune, see Next Steps (p. 38).

Using Java to Connect to a Neptune DB Instance

**Preview Release**  Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.
The following section walks you through the running of a complete Java sample that connects to a Neptune DB instance and performs a Gremlin traversal.

These instructions must be followed from an Amazon EC2 instance in the same virtual private cloud (VPC) as your Neptune DB instance.

**To connect to Neptune using Java**

1. Install Apache Maven on your EC2 instance. First, type the following to add a repository with a Maven package.

```
```

Type the following to set the version number for the packages.

```
sudo sed -i s/$releasever/6/g /etc/yum.repos.d/epel-apache-maven.repo
```

Then you can use `yum` to install Maven.

```
sudo yum install -y apache-maven
```

2. The Gremlin libraries require Java 8. Type the following to install Java 8 on your EC2 instance.

```
sudo yum install java-1.8.0-devel
```

3. Type the following to set Java 8 as the default runtime on your EC2 instance.

```
sudo /usr/sbin/alternatives --config java
```

When prompted, type the number for Java 8 (2).

4. Type the following to set Java 8 as the default compiler on your EC2 instance.

```
sudo /usr/sbin/alternatives --config javac
```

When prompted, type the number for Java 8 (2).

5. Create a new directory named `gremlinjava`.

```
mkdir gremlinjava
cd gremlinjava
```

6. In the `gremlinjava` directory, create a `pom.xml` file, and then open it in a text editor.

```
nano pom.xml
```

7. Copy the following into the `pom.xml` file and save it.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.amazonaws</groupId>
  <artifactId>GremlinExample</artifactId>
  <version>1.0-SNAPSHOT</version>
</project>
```
<name>GremlinExample</name>
<url>http://maven.apache.org</url>
<dependencies>
<dependency>
<groupId>org.apache.tinkerpop</groupId>
<artifactId>gremlin-driver</artifactId>
<version>3.3.0</version>
</dependency>
</dependencies>
</project>

<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>2.0.2</version>
<configuration>
<source>1.8</source>
<target>1.8</target>
</configuration>
</plugin>
<plugin>
<groupId>org.codehaus.mojo</groupId>
<artifactId>exec-maven-plugin</artifactId>
<version>1.3</version>
<configuration>
<mainClass>com.amazonaws.App</mainClass>
<complianceLevel>1.8</complianceLevel>
</configuration>
</plugin>
</plugins>
</build>

Note
If you are modifying an existing Maven project, the required dependency is highlighted in
the preceding code.

8. Create subdirectories for the example source code (src/main/java/com/amazonaws/) by typing
the following at the command line:

```bash
mkdir -p src/main/java/com/amazonaws/
```

9. In the src/main/java/com/amazonaws/ directory, create a file named App.java, and then open
it in a text editor.

```bash
nano src/main/java/com/amazonaws/App.java
```

10. Copy the following into the App.java file. Replace your-neptune-endpoint with the address
of your Neptune DB instance. Note that you must NOT include the https:// prefix in the
addContactPoint method.

    ```java
    package com.amazonaws;
    import org.apache.tinkerpop.gremlin.driver.Cluster;
    import org.apache.tinkerpop.gremlin.driver.Client;
    import org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversalSource;
    import org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversal;
    import org.apache.tinkerpop.gremlin.structure.util.empty.EmptyGraph;
    import org.apache.tinkerpop.gremlin.driver.remote.DriverRemoteConnection;
    ```

    Note
    For information about finding the hostname of your Neptune DB instance, see Finding the
    Endpoint for a Neptune Cluster (p. 19).
public class App
{
    public static void main( String[] args )
    {
        Cluster.Builder builder = Cluster.build();
        builder.addContactPoint("your-neptune-endpoint");
        builder.port(8182);
        Cluster cluster = builder.create();
        GraphTraversalSource g = EmptyGraph.instance().traversal().withRemote(DriverRemoteConnection.using(cluster));
        GraphTraversal t = g.V().limit(2).valueMap();
        t.forEachRemaining(
            e -> System.out.println(e));
        cluster.close();
    }
}

11. Compile and run the sample using the following Maven command:

```java
mvn compile exec:java
```

The preceding example returns a map of the key and values of each property for the first two vertexes in the graph by using the `g.V().limit(2).valueMap()` traversal. To query for something else, replace it with another Gremlin traversal.

For more information about Amazon Neptune, see Next Steps (p. 38).

**Using Python to Connect to a Neptune DB Instance**

**Preview Release**  Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

The following section walks you through the running of a Python sample that connects to a Neptune DB instance and performs a Gremlin traversal.

These instructions must be followed from an Amazon EC2 instance in the same virtual private cloud (VPC) as your Neptune DB instance.

Before you begin, do the following:

- Download and install Python 2.7 or later from the Python.org website.
- Verify that you have `pip` installed. If you don't have `pip` or you're not sure, see Do I need to install pip? in the `pip` documentation.

**To connect to Neptune using Python**

1. Type the following to install the `gremlinpython` package:

```bash
pip install gremlinpython
```

API Version 2017-11-29
2. Create a file named `gremlinexample.py`, and then open it in a text editor.

3. Copy the following into the `gremlinexample.py` file. Replace `your-neptune-endpoint` with the address of your Neptune DB instance.

   For information about finding the address of your Neptune DB instance, see the Accessing a Neptune Graph (p. 19) section.

   ```python
   from __future__ import print_function  # Python 2/3 compatibility
   from gremlin_python import statics
   from gremlin_python.structure.graph import Graph
   from gremlin_python.process.graph_traversal import *
   from gremlin_python.process.strategies import *
   from gremlin_python.driver.driver_remote_connection import DriverRemoteConnection

   graph = Graph()
   g = graph.traversal().withRemote(DriverRemoteConnection('ws://your-neptune-endpoint:8182/gremlin', 'g'))
   print(g.V().limit(2).toList())
   ```

4. Type the following command to run the sample.

   ```bash
   python gremlinexample.py
   ```

   The Gremlin query at the end of this example returns the vertices `(g.V().limit(2))` in a list. This list is then printed with the standard Python `print` function.

   **Note**
   The final part of the Gremlin query, `toList()`, is required to submit the traversal to the server for evaluation. If you don't include that method or another equivalent method, the query is not submitted to the Neptune DB instance.

   The following methods submit the query to the Neptune DB instance:
   
   - `toList()`
   - `toSet()`
   - `next()`
   - `nextTraverser()`
   - `iterate()`

   The preceding example returns the first two vertices in the graph by using the `g.V().limit(2).toList()` traversal. To query for something else, replace it with another Gremlin traversal with one of the appropriate ending methods.

   For more information about Amazon Neptune, see Next Steps (p. 38).

---

**Using .NET to Connect to a Neptune DB Instance**

**Preview Release** Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.
The following section contains a code example written in C# that connects to a Neptune DB instance and performs a Gremlin traversal.

Connections to Amazon Neptune must be from an Amazon EC2 instance in the same virtual private cloud (VPC) as your Neptune DB instance. This sample code was tested on an Amazon EC2 instance running Ubuntu.

Before you begin, do the following:

- Install .NET on the Amazon EC2 instance. To get instructions for installing .NET on multiple operating systems, including Windows, Linux, and macOS, see Get Started with .NET.
- Install Gremlin.NET. For more information, see Gremlin.NET in the TinkerPop documentation.

To connect to Neptune using Gremlin.NET

1. Create a new .NET project.

   ```
   dotnet new console -o gremlinExample
   ```

2. Change directories into the new project directory.

   ```
   cd gremlinExample
   ```

3. Copy the following into the Program.cs file. Replace `your-neptune-endpoint` with the address of your Neptune DB instance.

   For information about finding the address of your Neptune DB instance, see the Accessing a Neptune Graph (p. 19) section.

   ```
   using System;
   using System.Threading.Tasks;
   using System.Collections.Generic;
   using Gremlin.Net;
   using Gremlin.Net.Driver;

   namespace gremlinExample
   {
   class Program
   {
   static void Main(string[] args)
   {
   try
   {
   var endpoint = "your-neptune-endpoint";
   // This uses the default Neptune and Gremlin port, 8182
   var gremlinServer = new GremlinServer(endpoint);
   var gremlinClient = new GremlinClient(gremlinServer);
   Program program = new Program();
   program.RunQueryAsync(gremlinClient).Wait();
   }
   catch (Exception e)
   {
   Console.WriteLine("(0)", e);
   }
   }
   
   private async Task RunQueryAsync(GremlinClient gremlinClient)
   {
   ```
The Gremlin query at the end of this example returns the count of a single vertex for testing purposes. It is then printed to the console.

**Note**  
The final part of the Gremlin query, `next()`, is required to submit the traversal to the server for evaluation. If you don't include that method or another equivalent method, the query is not submitted to the Neptune DB instance.

The following methods submit the query to the Neptune DB instance:

- `toList()`  
- `toSet()`  
- `next()`  
- `nextTraverser()`  
- `iterate()`

The preceding example returns a number by using the `g.V().limit(1).count().next()` traversal. To query for something else, replace it with another Gremlin traversal with one of the appropriate ending methods.

For more information about Amazon Neptune, see [Next Steps (p. 38)](#).

### Using Node.js to Connect to a Neptune DB Instance

The following section walks you through the running of a Node.js sample that connects to a Neptune DB instance and performs a Gremlin traversal.

These instructions must be followed from an Amazon EC2 instance in the same virtual private cloud (VPC) as your Neptune DB instance.

Before you begin, do the following:

- Verify that Node.js is installed. If it is not, download and install Node.js from the [Nodejs.org website](http://nodejs.org).

To connect to Neptune using Node.js

1. Type the following to install the `gremlin-javascript` package:

```bash
npm install gremlin #save
```

2. Create a file named `gremlinexample.js` and open it in a text editor.

3. Copy the following into the `gremlinexample.js` file. Replace `your-neptune-endpoint` with the address of your Neptune DB instance.
For information about finding the address of your Neptune DB instance, see the Accessing a Neptune Graph (p. 19) section.

gremlin = require('gremlin');
const client = gremlin.createClient(8182, "your-neptune-endpoint", { accept: "application/vnd.gremlin-v2.0+json" });
client.execute('g.V().limit(2)', (err, results) => {
  if (err) {
    return console.error(err)
  }
  console.log(results);
});

4. Type the following command to run the sample:

   node gremlinexample.js

The preceding example returns the first two vertices in the graph by using the `g.V().limit(2)` traversal. To query for something else, replace it with another Gremlin traversal.

For more information about Amazon Neptune, see Next Steps (p. 38).

**Gremlin HTTP and WebSocket API**

**Preview Release**  Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

Gremlin HTTP requests all use a single endpoint:  
http://your-neptune-endpoint:8182/gremlin

**Note**  Amazon Neptune does not support the bindings property.

For more information about connecting to the Gremlin endpoint, see Accessing the Neptune Graph with Gremlin (p. 21).

The Amazon Neptune Gremlin implementation has specific implementation details. For more information, see Neptune Gremlin Implementation Differences (p. 22).

For information about the Gremlin language and traversals, see The Traversal in the Apache TinkerPop documentation.

**Next Steps**

**Preview Release**  Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

These resources provide more information about Neptune and Gremlin traversals.
Accessing the Neptune Graph with SPARQL

SPARQL is a query language for the Resource Description Framework (RDF), which is a graph data format designed for the web. Amazon Neptune is compatible with SPARQL 1.1. This means that you can connect to a Neptune DB instance and query the graph using the query language described in the SPARQL 1.1 Query Language specification.

A query in SPARQL consists of a SELECT clause to specify the variables to return and a WHERE clause to specify which data to match in the graph. If you are unfamiliar with SPARQL queries, see Writing Simple Queries in the SPARQL 1.1 Query Language.

Important

Neptune does not support SPARQL UPDATE LOAD from URI. For small datasets, SPARQL UPDATE INSERT might be an option. If you need to load data from a file, see Loading Data into Neptune (p. 48).

Before you begin, you must have the following:

• A Neptune DB instance. For information about creating a Neptune DB instance, see Getting Started with Neptune (p. 13).
• An Amazon EC2 instance in the same virtual private cloud (VPC) as your Neptune DB instance.

Loading an Example Graph

This section walks you through loading an example graph in an RDF format from Amazon S3.

The following is a visual representation of the graph:
Important
Before you load data from Amazon S3, you must create an Amazon S3 VPC endpoint in your VPC. For information about creating an endpoint, see Amazon S3 VPC Endpoint (p. 57). For information about the limitations of VPC endpoints, Endpoints for S3.

Sample data is available in an Amazon S3 bucket.

Run the following command to load the graph from the S3 bucket. Replace the endpoint, access key, and secret key placeholders with the appropriate values.

```
Note
For information about finding the hostname of your Neptune DB instance, see Finding the Endpoint for a Neptune Cluster (p. 19).
```

```
curl -X POST \
-H 'Content-Type: application/json' \
http://your-neptune-endpoint:8182/loader -d ' 
{"source": "s3://neptune-us-east-1/moderngraph.ttl", 
"format": "turtle", 
"accessKey": "access-key-id", 
"secretKey": "secret-key", 
"region": "us-east-1", 
"failOnError": "FALSE"
}
```

Now that you have loaded some data, you can access the graph using the method of your choice.

Topics
- Using the RDF4J Console to Connect to a Neptune DB Instance (p. 41)
- Using the HTTP REST Endpoint to Connect to a Neptune DB Instance (p. 42)
Using the RDF4J Console to Connect to a Neptune DB Instance

The RDF4J Console allows you to experiment with RDF graphs and queries in a REPL (read-eval-print loop) environment.

You can add a remote graph database as a repository and query it from the RDF4J Console. This section walks you through the configuration of the RDF4J Console to connect remotely to a Neptune DB instance.

To connect to Neptune using the RDF4J Console

1. Download the RDF4J SDK from the Download page on the RDF4J website.
2. Unzip the RDF4J SDK zip file.
3. In a terminal, navigate to the RDF4J SDK directory, and then type the following command to run the RDF4J Console:
   ```bash
   bin/console.sh
   ```
   You should see output similar to the following:

   ```
   14:11:51.126 [main] DEBUG o.e.r.c.platform.PlatformFactory - os.name = linux
   14:11:51.130 [main] DEBUG o.e.r.c.platform.PlatformFactory - Detected Posix platform
   Connected to default data directory
   RDF4J Console 2.1.5
   2.1.5
   Type 'help' for help.
   >
   ```

   You are now at the > prompt. This is the general prompt for the RDF4J Console. You use this prompt for setting up repositories and other operations. A repository has its own prompt for running queries.

4. At the > prompt, type the following to create a SPARQL repository for your Neptune DB instance:
   ```bash
   create sparql
   ```

5. The RDF4J Console prompts you for values for the variables required to connect to the SPARQL endpoint.

   ```
   Please specify values for the following variables:
   
   Specify the following values:
   
<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPARQL query endpoint:</td>
<td><a href="http://your-neptune-endpoint:8182/sparql">http://your-neptune-endpoint:8182/sparql</a></td>
</tr>
</tbody>
</table>
   ```
SPARQL update endpoint: http://your-neptune-endpoint:8182/sparql
Local repository ID [endpoint@localhost]: neptune
Repository title [SPARQL endpoint repository @localhost]: Neptune DB instance

For information about finding the address of your Neptune DB instance, see the Accessing a Neptune Graph (p. 19) section.

If the operation is successful, you see the following message:

Repository created

6. At the > prompt, type the following to connect to the Neptune DB instance:

open neptune

If the operation is successful, you see the following message:

Opened repository 'neptune'

You are now at the neptune> prompt. At this prompt, you can run queries against the Neptune graph.

Note
Now that you have added the repository, the next time you run bin/console.sh, you can immediately run the open neptune command to connect to the Neptune DB instance.

7. At the neptune> prompt, type the following to run a SPARQL query that returns all the triples (subject-predicate-object) in the graph by using the ?s ?p ?o query with no constraints. To query for something else, replace the text after the sparql command with another SPARQL query.

sparql select ?s ?p ?o where {?s ?p ?o}

For more information about Amazon Neptune, see Next Steps (p. 46).

Using the HTTP REST Endpoint to Connect to a Neptune DB Instance

Preview Release Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

Neptune provides an HTTP endpoint for SPARQL queries. The REST interface is compatible with SPARQL version 1.1.

The following instructions walk you through connecting to the SPARQL endpoint using the curl command and HTTP. These instructions must be followed from an Amazon EC2 instance in the same virtual private cloud (VPC) as your Neptune DB instance.
The HTTP endpoint for SPARQL queries to a Neptune DB instance is \texttt{http://your-neptune-endpoint:8182/sparql}.

**Note**
For information about finding the hostname of your Neptune DB instance, see the Finding the Endpoint for a Neptune Cluster (p. 19) section.

**To connect to Neptune using the HTTP REST endpoint**

- The following example uses \texttt{curl} to submit a SPARQL query through HTTP POST.

\begin{verbatim}
\end{verbatim}

The preceding example returns all the triples (subject-predicate-object) in the graph by using the \texttt{?s \?p \?o} query with no constraints. To query for something else, replace it with another SPARQL query.

For more information about the Neptune SPARQL REST interface, see SPARQL HTTP API (p. 46). For more information about Amazon Neptune, see Next Steps (p. 46).

**Using Java to Connect to a Neptune DB Instance**

**Preview Release** Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

This section walks you through the running of a complete Java sample that connects to a Neptune DB instance and performs a SPARQL query.

These instructions must be followed from an Amazon EC2 instance in the same virtual private cloud (VPC) as your Neptune DB instance.

**To connect to Neptune using Java**

1. Install Apache Maven on your EC2 instance. First, type the following to add a repository with a Maven package:

\begin{verbatim}
\end{verbatim}

Type the following to set the version number for the packages:

\begin{verbatim}
sudo sed -i s/\$releasever/6/g /etc/yum.repos.d/epel-apache-maven.repo
\end{verbatim}

Then you can use \texttt{yum} to install Maven.

\begin{verbatim}
sudo yum install -y apache-maven
\end{verbatim}

2. This example was tested with Java 8 only. Type the following to install Java 8 on your EC2 instance:

\begin{verbatim}
sudo yum install java-1.8.0-devel
\end{verbatim}

3. Type the following to set Java 8 as the default runtime on your EC2 instance:
When prompted, type the number for Java 8.

4. Type the following to set Java 8 as the default compiler on your EC2 instance:

    sudo /usr/sbin/alternatives --config javac

When prompted, type the number for Java 8.

5. In a new directory, create a pom.xml file, and then open it in a text editor.

6. Copy the following into the pom.xml file and save it.

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.amazonaws</groupId>
    <artifactId>RDFExample</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>RDFExample</name>
    <url>http://maven.apache.org</url>
    <dependencies>
        <dependency>
            <groupId>org.eclipse.rdf4j</groupId>
            <artifactId>rdf4j-runtime</artifactId>
            <version>2.2</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>exec-maven-plugin</artifactId>
                <version>1.2.1</version>
                <configuration>
                    <mainClass>com.amazonaws.App</mainClass>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

**Note**
If you are modifying an existing Maven project, the required dependency is highlighted in the preceding code.

7. To create subdirectories for the example source code (src/main/java/com/amazonaws/), type the following at the command line:
8. In the src/main/java/com/amazonaws/ directory, create a file named App.java, and then open it in a text editor.

9. Copy the following into the App.java file. Replace your-neptune-endpoint with the address of your Neptune DB instance.

   **Note**
   For information about finding the hostname of your Neptune DB instance, see the Finding the Endpoint for a Neptune Cluster (p. 19) section.

```java
package com.amazonaws;

import org.eclipse.rdf4j.repository.Repository;
import org.eclipse.rdf4j.repository.http.HTTPRepository;
import org.eclipse.rdf4j.repository.sparql.SPARQLRepository;
import java.util.List;
import org.eclipse.rdf4j.RDF4JException;
import org.eclipse.rdf4j.repository.RepositoryConnection;
import org.eclipse.rdf4j.query.TupleQuery;
import org.eclipse.rdf4j.query.TupleQueryResult;
import org.eclipse.rdf4j.query.BindingSet;
import org.eclipse.rdf4j.query.QueryLanguage;
import org.eclipse.rdf4j.model.Value;

public class App {
    public static void main(String[] args) {
        String sparqlEndpoint = "http://your-neptune-endpoint:8182/sparql";
        Repository repo = new SPARQLRepository(sparqlEndpoint);
        repo.initialize();

        try (RepositoryConnection conn = repo.getConnection()) {
            try (TupleQueryResult result = conn.prepareTupleQuery(QueryLanguage.SPARQL, sparqlEndpoint)) {
                while (result.hasNext()) {
                    BindingSet bindingSet = result.next();
                    Value s = bindingSet.getValue("s");
                    Value p = bindingSet.getValue("p");
                    Value o = bindingSet.getValue("o");
                    System.out.print(s);
                    System.out.print("\t");
                    System.out.print(p);
                    System.out.print("\t");
                    System.out.println(o);
                }
            }
        }
    }
}
```

10. Use the following Maven command to compile and run the sample:
The preceding example returns all the triples (subject-predicate-object) in the graph by using the ?s ?p ?o query with no constraints. To query for something else, replace the query with another SPARQL query.

The iteration of the results in the example prints the value of each variable returned. The Value object is converted to a String and then printed. If you change the SELECT part of the query, you must modify the code.

For more information about Amazon Neptune, see Next Steps (p. 46).

**SPARQL HTTP API**

**Preview Release**  Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

SPARQL HTTP requests are accepted at the following endpoint: http://your-neptune-endpoint:8182/sparql

For more information about connecting to Amazon Neptune with SPARQL, see Accessing the Neptune Graph with SPARQL (p. 39).

For more information about the SPARQL protocol and query language, see the SPARQL 1.1 Protocol and the SPARQL 1.1 Query Language specification.

SPARQL UPDATE LOAD from URI only works with resources within the same VPC.

This includes Amazon S3 URLs in the us-east-1 Region with an Amazon S3 VPC endpoint created. For information about creating a VPC endpoint, see Amazon S3 VPC Endpoint (p. 57).

The Amazon S3 URL must be HTTPS, and any authentication must be included in the URL. For more information, see Authenticating Requests: Using Query Parameters.

If you need to load data from a file, we recommend using the Amazon Neptune loader API. For more information, see Loading Data into Neptune (p. 48).

**Note**

The Amazon Neptune loader API is non-ACID.

**Next Steps**

**Preview Release**  Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

These resources provide more information about Neptune and SPARQL queries.

- Loading Data into Neptune (p. 48).
- More about SPARQL queries and the Resource Description Framework (RDF):
  - SPARQL HTTP API (p. 46)
  - SPARQL 1.1 Query Language
Secure Sockets Layer Settings for a Neptune Cluster

Secure Sockets Layer (SSL) is disabled by default on Amazon Neptune clusters. This section walks you through how to enable SSL for Neptune.

The root SSL certificate for connecting to a Neptune DB instance is available for download at the following location: https://s3.amazonaws.com/rds-downloads/rds-ca-beta-2015-root.pem

You can then specify this certificate when you connect. For example, to use the curl command, specify the certificate with the `--cacert rds-ca-beta-2015-root.pem` parameter/value pair, and change the URL to begin with `https://`.

```
```

You can enable SSL on a Neptune cluster by changing the `neptune_enable_tls` parameter in the DB cluster parameter group.

When creating a cluster, you must specify a custom parameter group if you want to change any cluster parameters. You can’t modify the parameters in the default DB cluster parameter group (default.neptune1).

**Note**
If you change the `neptune_enable_tls` parameter, you must reboot all DB instances in the cluster.

You can create a Neptune cluster with SSL enabled by choosing a DB cluster parameter group that already has the `neptune_enable_tls` parameter set to 1.

**Warning**
If you enable SSL in a DB cluster parameter group, SSL is enabled for every Neptune cluster that uses that parameter group.

**To enable SSL for a Neptune cluster**

2. Choose Parameter groups in the navigation pane.
3. Follow the Name link for the DB cluster parameter group that you want to edit.
   (Optional) Choose Create Parameter Group to create a new cluster parameter group. Choose DB Cluster Parameter Group for the type, and create the new group. Then choose the Name of the new parameter group.
   **Important**
   This step is required if you have only the default DB cluster parameter group because the default DB cluster parameter group can’t be modified.
4. Set the value for `neptune_enable_tls` to 1.
5. Choose Save changes.
6. Reboot every Neptune DB instance in the Neptune cluster.
Loading Data into Neptune

**Preview Release**  Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

Amazon Neptune provides a process for loading data from external files directly into a Neptune DB instance. You can use this process instead of executing a large number of `INSERT` statements, `addVertex` and `addEdge` steps, or other API calls.

The Neptune **Loader** command is faster, has less overhead, is optimized for large datasets, and supports both RDF (Resource Description Framework) and Gremlin data.

The following diagram shows an overview of the load process:

As the diagram shows, there are four basic steps in the loading process:

1. Copy the data files to an Amazon Simple Storage Service (Amazon S3) bucket.
2. Create an IAM role with Read and List access to the bucket.
3. Create an Amazon S3 VPC endpoint.
4. Start the Neptune loader by sending a request via HTTP to the Neptune DB instance.
5. The Neptune DB instance assumes the IAM role to load the data from the bucket.

The following sections provide instructions for preparing and loading data into Neptune.

**Topics**
- Prerequisites: IAM Role and Amazon S3 Access (p. 49)
- Load Data Formats (p. 51)
Prerequisites: IAM Role and Amazon S3 Access

**Preview Release**  Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

Loading data from an Amazon S3 bucket requires an AWS Identity and Access Management (IAM) role that has access to the bucket. Amazon Neptune assumes this role in order to load the data.

The following sections show how to create an IAM policy and an IAM role, associate the two, and then attach the role to your Neptune cluster.

**Note**
These instructions require the user to have access to the IAM console and permissions to manage IAM roles and policies. For more information, see Permissions for Working in the AWS Management Console in the IAM User Guide.

The Amazon Neptune console requires the user to have following IAM permissions to attach the role to the Neptune cluster:

- `iam:GetAccountSummary` on resource: *
- `iam:ListAccountAliases` on resource: *

---

**Creating an IAM Policy to Allow Amazon S3 Read and List Access**

To create an IAM Policy to allow read access to an Amazon S3 bucket.

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Policy.
3. Choose Create policy.
4. Choose the JSON tab.
5. In the text area, add the following text, replacing `bucket-name` with the name of the S3 bucket that you want to load data from.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": ["s3:ListBucket"],
         "Resource": ["arn:aws:s3:::bucket-name"]
      },
      {
         "Effect": "Allow",
         "Action": ["s3:GetObject"],
         "Resource": ["arn:aws:s3:::bucket-name/*"]
      }
   ]
}
```
6. Complete the steps in Creating an IAM Role to Allow Amazon Neptune to Access AWS Services (p. 50).

Creating an IAM Role to Allow Amazon Neptune to Access AWS Services

After creating an IAM policy to allow Neptune to access AWS resources, you must create an IAM role and attach the IAM policy to the new IAM role.

Start with an Amazon Relational Database Service (Amazon RDS) role and modify it to work with Amazon Neptune.

To create an IAM role to allow Amazon Neptune to access AWS services

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Roles.
3. Choose Create role.
4. Under AWS service, choose RDS.
5. Under Select your use case, choose RDS.
6. Choose Next: Permissions.
7. Choose Next: Review.
8. Set Role Name to a name for your IAM role, for example: NeptuneLoadFromS3. You can also add an optional Role Description value.
9. Choose Create Role.
10. In the navigation pane, choose Roles.
11. In the Search field, type the name of the role you created, and choose the role when it appears in the list.
12. On the Permissions tab, detach the following default roles from the policy:

   • AmazonRDSDirectoryServiceAccess
   • RDSCloudHsmAuthorizationRole

To detach a role, choose the X associated with the role on the right, and then choose Detach.
13. On the Permissions tab, choose Attach policy.
14. On the Attach policy page, type the name of your policy in the Search field.
15. When it appears in the list, choose the policy that you defined in the previous section, for example: NeptuneLoadFromS3.
16. Choose Attach policy.
17. In the navigation pane, choose Roles.
18. In the Search field, type the name of the role you created, and choose the role when it appears in the list.
20. In the text field, paste the following trust policy.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
```
Adding the IAM Role to a Cluster

Use the console to add the IAM role to an Amazon Neptune cluster. This allows any Neptune DB instance in the cluster to assume the role and load from Amazon S3.

Note
The Amazon Neptune console requires the user to have the following IAM permissions to attach the role to the Neptune cluster:

iam:GetAccountSummary on resource: *
iam:ListAccountAliases on resource: *

To add an IAM role to an Amazon Neptune cluster

2. In the navigation pane, choose Clusters.
3. Choose the radio button next to the cluster you want to modify.
4. Under Actions, choose Manage IAM roles.
5. Choose the IAM role you created in the previous section.
6. Choose Done.

Next Steps
Now that you have granted access to the Amazon S3 bucket, you can prepare to load data. For information about supported formats, see Load Data Formats (p. 51).

Load Data Formats

Preview Release  Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

The Neptune Load API currently requires specific formats for incoming data. The following formats are available, listed with their identifiers for the Neptune loader API in parentheses.

- CSV format (csv) for property graph / Gremlin
Gremlin Load Data Format

Important
All files must be encoded in UTF-8 format. If a file is not in UTF format, Neptune tries to load it anyway as UTF-8 data.

If your data is not in a supported format, you must convert it before you load it into a Neptune DB instance.

Compression Support
Neptune supports compression of single files in gzip format. The file name must end in the .gz extension and must contain a single text file encoded in UTF-8 format. Multiple files can be loaded, but each one must be contained in a separate .gz file (or uncompressed text file). Archive files (for example, .tar, .tar.gz, and .tgz) are not supported.

The following sections describe the formats in more detail.

Topics
- Gremlin Load Data Format (p. 52)
- RDF Load Data Formats (p. 56)

Gremlin Load Data Format

To load Apache TinkerPop Gremlin data using the csv format, you must specify the vertices and the edges in separate files.

For each load command, the set of files to be loaded must be in the same folder in the Amazon S3 bucket, and you specify the folder name for the source parameter. The file names and extensions are not important.

To avoid vertices without corresponding edges, we recommend loading vertex files first and then the edge files in a subsequent loader command.

The Neptune csv format follows the RFC 4180 csv specification. For more information, see Common Format and MIME Type for CSV Files on the Internet Engineering Task Force (IETF) website.

Note
All files must be encoded in UTF-8 format.

Each file has a comma-separated header row. The header row consists of both system column headers and property column headers.

System Column Headers
The required and allowed system column headers are different for vertex files and edge files.

Each system column can appear only once in a header.
All labels are case-sensitive.

**Vertex headers**

- **~id** - Required
  An ID for the vertex.
- **~label**
  A label for the vertex. Multiple label values are allowed. Separate values with a semicolon (;) character.

**Edge headers**

- **~id** - Required
  An ID for the edge.
- **~from** - Required
  The vertex ID of the from vertex.
- **~to** - Required
  The vertex ID of the to vertex.
- **~label**
  A label for the edge. Edges can only have a single label.

**Property Column Headers**

You can specify a column for a property by using the following syntax. The type names are not case-sensitive.

```
propertyname:type
```

You can specify a column for an array type by adding [ ] to the type.

```
propertyname:type[
```

**Note**
Spaces are not allowed in the column headers, so property names cannot include spaces.

The following example shows the column header for a property named `age` of type `Int`.

```
age: Int
```

Every row in the file would be required to have an integer in that position or be left empty.

Arrays of strings are allowed, but strings in an array must not include the semicolon (;) character.

The following section lists all the available data types.

**Data Types**

This is a list of the allowed property types, with a description of each type.

**Bool (or Boolean)**
Indicates a Boolean field. Allowed values: 0, 1, false, true

Whole Number Types

Values outside of the defined ranges result in an error.

<table>
<thead>
<tr>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>-127 to 126</td>
</tr>
<tr>
<td>Short</td>
<td>-32768 to 32767</td>
</tr>
<tr>
<td>Int</td>
<td>-2^31 to 2^31-1</td>
</tr>
<tr>
<td>Long</td>
<td>-2^63 to 2^63-1</td>
</tr>
</tbody>
</table>

Decimal Number Types

Supports both decimal notation or scientific notation. Also allows symbols such as (+/-) INFINITY or NaN. INF is not supported.

<table>
<thead>
<tr>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Float</td>
<td>32-bit IEEE 754 floating point</td>
</tr>
<tr>
<td>Double</td>
<td>64-bit IEEE 754 floating point</td>
</tr>
</tbody>
</table>

String

Quotation marks are optional. Commas, newline, and carriage return characters are automatically escaped if they are included in a string surrounded by double quotation marks ("). Example: "Hello, World"

To include quotation marks in a quoted string, you can escape the quotation mark by using two in a row: Example: "$Hello ""World""

Arrays of strings are allowed, but strings in an array must not include the semicolon (;) character.

If you want to surround strings in an array with quotation marks, you must surround the whole array with one set of quotation marks. Example: "String one; String 2; String 3"

Date

Java date in ISO-8601 format. Supports the following formats: YYYY-MM-DD, YYYY-MM-DDTHH:mm, YYYY-MM-DDTHH:mm:ss, YYYY-MM-DDTHH:mm:ssZ

Row format

Delimiters

Fields in a row are separated by a comma. Records are separated by a newline or a newline followed by a carriage return.

Blank Fields

Blank fields are allowed for non-required columns (such as user-defined properties). A blank field still requires a comma separator. The example in the next section has a blank field in each example vertex.

Vertex IDs
- *id values must be unique for all vertixes in every vertex file. Multiple vertex rows with identical ~id values are applied to a single vertex in the graph.

**Edge IDs**

Additionally, -id values must be unique for all edges in every edge file. Multiple edge rows with identical -id values are applied to the single edge in the graph.

**Labels**

Labels are case-sensitive.

**String Values**

Quotation marks are optional. Commas, newline, and carriage return characters are automatically escaped if they are included in a string surrounded by double quotation marks (*).

**CSV Specification**

The Neptune csv format follows the RFC 4180 csv specification, including the following requirements.

- Both Unix and Windows style line endings are supported (\n or \r\n).
- Any field can be quoted (using double quotation marks).
- Fields containing a line-break, double-quote, or commas must be quoted. (If they are not, load aborts immediately.)
- A double quotation mark character (") in a field must be represented by two (double) quotation mark characters. For example, a string Hello "World" must be present as "Hello ""World""" in the data.
- Surrounding spaces between delimiters are ignored. If a row is present as value1, value2, they are stored as "value1" and "value2".
- Any other escape characters are stored verbatim. For example, "data1\tdata2" is stored as "data1\tdata2". No further escaping is needed as long as these characters are enclosed within quotation marks.
- Blank fields are allowed. A blank field is considered an empty value.
- Multiple values for a field are specified with a semicolon (;) between values.

For more information, see Common Format and MIME Type for CSV Files on the Internet Engineering Task Force (IETF) website.

**Example**

The following diagram shows an example of two vertices and an edge taken from the TinkerPop Modern Graph.
The following is the graph in Neptune CSV load format.

**Vertex file:**

```
~id, name: String, age: Int, lang: String, ~label
v1, "marko", 29, , person
v2, "lop", , "java", software
```

Tabular view of vertex file.

<table>
<thead>
<tr>
<th>~id</th>
<th>name: String</th>
<th>age: Int</th>
<th>lang: String</th>
<th>~label</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>&quot;marko&quot;</td>
<td>29</td>
<td></td>
<td>person</td>
</tr>
<tr>
<td>v2</td>
<td>&quot;lop&quot;</td>
<td></td>
<td>&quot;java&quot;</td>
<td>software</td>
</tr>
</tbody>
</table>

**Edge file:**

```
~id, ~from, ~to, ~label, weight: Double
e1, v1, v2, created, 0.4
```

Tabular view of edge file.

<table>
<thead>
<tr>
<th>~id</th>
<th>~from</th>
<th>~to</th>
<th>~label</th>
<th>weight: Double</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>v1</td>
<td>v2</td>
<td>created</td>
<td>0.4</td>
</tr>
</tbody>
</table>

**Next Steps**

Now that you know the loading formats, see Example: Loading Data into a Neptune DB Instance (p. 57).

**RDF Load Data Formats**

**Preview Release**  Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

To load Resource Description Framework (RDF) data, you can use one of the following standard formats as specified by the W3C:

- N-Triples (ntriples) from the specification at [https://www.w3.org/TR/n-triples/](https://www.w3.org/TR/n-triples/)
- N-Quads (nquads) from the specification at [https://www.w3.org/TR/n-quads/](https://www.w3.org/TR/n-quads/)
- RDF/XML (rdfxml) from the specification at [https://www.w3.org/TR/rdf-syntax-grammar/](https://www.w3.org/TR/rdf-syntax-grammar/)
- Turtle (turtle) from the specification at [https://www.w3.org/TR/turtle/](https://www.w3.org/TR/turtle/)

**Important**

All files must be encoded in UTF-8 format. For N-Quads and N-triples data that includes Unicode characters, \uxxxxx escape sequences are supported. However, Neptune does not support normalization. If a value is present that
Example: Loading Data

requires normalization, it will not match byte-to-byte during querying. For more information about normalization, see the Normalization page on Unicode.org.

Next Steps

Now that you know the loading formats, see Example: Loading Data into a Neptune DB Instance (p. 57).

Example: Loading Data into a Neptune DB Instance

Preview Release  Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

This example shows how to load data into Amazon Neptune. Unless stated otherwise, you must follow these steps from an Amazon Elastic Compute Cloud (Amazon EC2) instance in the same Amazon Virtual Private Cloud (VPC) as your Neptune DB instance.

Prerequisites

Before you begin, you must have the following:

• A Neptune DB instance.
  For information about launching a Neptune DB instance, see Getting Started with Neptune (p. 13).
• An Amazon Simple Storage Service (Amazon S3) bucket to put the data files in.
  You can use an existing bucket. If you don’t have an S3 bucket, see Create a Bucket in the Amazon S3 Getting Started Guide.
• An IAM role for the Neptune DB instance to assume that has an IAM policy that allows access to the data files in the S3 bucket. The policy must grant Read and List permissions.
  For information about creating a role with access to S3 and associating it with a Neptune cluster, see Prerequisites: IAM Role and Amazon S3 Access (p. 49).

• Note
  The Neptune Load API needs read access to the data files only. The IAM policy doesn’t need to allow write access or access to the entire bucket.
• An Amazon S3 VPC endpoint. For more information, see the following section.

Amazon S3 VPC Endpoint

The Neptune loader requires a VPC endpoint for Amazon S3.

To set up access for Amazon S3

1. Sign in to the AWS Management Console and open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the left navigation pane, choose Endpoints.
3. Choose Create Endpoint.
4. Choose the Service Name com.amazonaws.us-east-1.s3.
5. Choose the VPC that contains your Neptune DB instance.
6. Select the check box next to the route tables that are associated with the subnets related to your cluster. If you only have one route table, you must select that box.

7. Choose Create Endpoint.

For information about creating the endpoint, see VPC Endpoints in the Amazon VPC User Guide. For information about the limitations of VPC endpoints, VPC Endpoints for Amazon S3.

To load data into a Neptune DB instance

1. Copy the data files to an Amazon S3 bucket. The S3 bucket must be in the same AWS Region (us-east-1) as the cluster that loads the data.

   You can use the following AWS CLI command to copy the files to the bucket.

   ```bash
   aws s3 cp data-file-name s3://bucket-name/object-key-name
   ```

   **Note**
   This command does not need to be run from the Amazon EC2 instance.

2. From a command line window, type the following to run the Neptune loader, replacing the values for the endpoint, Amazon S3 path, format, and access keys.

   The `format` parameter can be any of the following values: csv (Gremlin), ntriples, nquads, turtle, and rdfxml (RDF). For information about the other parameters, see Loader Command (p. 59).

   ```json
   curl -X POST \
   -H 'Content-Type: application/json' \
   http://your-neptune-endpoint:8182/loader -d '   
   {   
   "source" : "s3://bucket-name/object-key-name",
   "format" : "format",
   "iamRoleArn" : "arn:aws:iam::account-id:role/role-name",
   "region" : "us-east-1",
   "failOnError" : "FALSE"
   }
   ```

   For information about finding the hostname of your Neptune DB instance, see the Finding the Endpoint for a Neptune Cluster (p. 19) section.

   For information about creating and associating an IAM role with a Neptune cluster, see Prerequisites: IAM Role and Amazon S3 Access (p. 49).

   **Note**
   The `SOURCE` parameter accepts an Amazon S3 URI that points to either a single file or a folder. If you specify a folder, Neptune loads every data file in the folder. The URI can be in any of the following formats.

   - s3://bucket_name/object-key-name
   - https://s3.amazonaws.com/bucket_name/object-key-name
3. The Neptune loader returns a job id that allows you to check the status or cancel the loading process; for example:

```json
{
    "status": "200 OK",
    "payload": {
        "loadId": "ef478d76-d9da-4d94-8ff1-08d9d4863aa5"
    }
}
```

4. Type the following to get the status of the load with the loadId from Step 3:

```bash
curl -G 'http://your-neptune-endpoint:8182/loader/ef478d76-d9da-4d94-8ff1-08d9d4863aa5'
```

If the status of the load lists an error, you can request more detailed status and a list of the errors. For more information and examples, see Loader Get Status (p. 63).

5. (Optional) Cancel the Load job.

Type the following to Delete the loader job with the job id from Step 3:

```bash
curl -X DELETE 'http://your-neptune-endpoint:8182/loader/ef478d76-d9da-4d94-8ff1-08d9d4863aa5'
```

The DELETE command returns the HTTP code 200 OK upon successful cancellation.

Data from files from the load job that has finished loading is not rolled back. The data remains in the Neptune DB instance.

---

### Neptune Loader API Reference

**Preview Release** Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

This section describes the **Loader** APIs for Amazon Neptune that are accessible from the HTTP endpoint of a Neptune DB instance.

**Topics**
- **Loader Command** (p. 59)
- **Loader Get Status** (p. 63)
- **Loader Cancel Job** (p. 68)

#### Loader Command

**Preview Release** Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

Loads data from an Amazon S3 bucket into a Neptune DB instance.
To load data, you must send an HTTP POST request to the `http://your-neptune-endpoint:8182/loader` endpoint. The parameters for the loader request can be sent in the POST body or as URL-encoded parameters.

**Important**

The MIME type must be `application/json`.

The S3 bucket must be in the same AWS Region as the cluster.

**Request Syntax**

```json
{
  "source" : "string",
  "format" : "string",
  "iamRoleArn" : "string",
  "mode" : "NEW|RESUME|AUTO",
  "region" : "us-east-1",
  "failOnError" : "string"
}
```

**Request Parameters**

**source**

An Amazon S3 URI.

The `source` parameter accepts an Amazon S3 URI that points to either a single file or a folder. If you specify a folder, Neptune loads every data file in the folder.

The URI can be in any of the following formats.

- `s3://bucket_name/object-key-name`
- `https://s3.amazonaws.com/bucket_name/object-key-name`
- `https://s3-us-east-1.amazonaws.com/bucket_name/object-key-name`

**format**

The format of the data. For more information about data formats for the Neptune Loader command, see [Loading Data into Neptune](#).

**Allowed values**: csv (Gremlin), ntriples,nquads, rdfxml, turtle (RDF)

**iamRoleArn**

The Amazon Resource Name (ARN) for an IAM role to be assumed by the Neptune DB instance for access to the S3 bucket. For information about creating a role with access to Amazon S3 and associating it with a Neptune cluster, see [Prerequisites: IAM Role and Amazon S3 Access](#).

**region**

The AWS Region of the S3 bucket (must be `us-east-1`).

**mode**

Load job mode.

**AUTO** mode determines whether there is failed load, and resumes that if possible for the load request. If a failed load is not found, a new load request is created.
**RESUME** mode determines whether there is failed load, and resumes that if possible for the load request. If a failed load is not found, the load is aborted.

**NEW** mode creates a new load request regardless of failed loads.

**Default:** AUTO

**Allowed values:** NEW, RESUME, AUTO.

**failOnError**

Flag to toggle a complete stop on an error. **Default:** TRUE

**Allowed values:** TRUE, FALSE

[deprecated] **accessKey**

The `iamRoleArn` parameter is recommended instead. For information about creating a role with access to Amazon S3 and associating it with a Neptune cluster, see Prerequisites: IAM Role and Amazon S3 Access (p. 49).

An access key ID of an IAM role with access to the S3 bucket and data files.

For more information, see Access keys (access key ID and secret access key).

[deprecated] **secretKey**

The `iamRoleArn` parameter is recommended instead. For information about creating a role with access to Amazon S3 and associating it with a Neptune cluster, see Prerequisites: IAM Role and Amazon S3 Access (p. 49).

For more information, see Access keys (access key ID and secret access key).

**Response Syntax**

```json
{
   "status" : "200 OK",
   "payload" : {
      "loadId" : "guid_as_string"
   }
}
```

**200 OK**

Successfully started load job returns a 200 code.

**Errors**

When an error occurs, a JSON object is returned in the BODY of the response. The `message` object contains a description of the error.

**Error 400**

Syntax errors return a 400 bad request error. The message describes the error.

**Error 500**

A valid request that cannot be processed returns a 500 internal server error. The message describes the error.
Loader Error Messages

The following are possible error messages from the loader with a description of the error.

Max concurrent load limit breached (HTTP 400)
You can only have 1 load job at a time.

Couldn't find the AWS credential for iam_role_arn (HTTP 400)
The credentials were not found. Verify the supplied credentials against the IAM console or AWS CLI output.

S3 bucket not found for source (HTTP 400)
The S3 bucket does not exist. Check the name of the bucket.

The source source-uri does not exist/not reachable (HTTP 400)
No matching files were found in the S3 bucket.

Unable to connect to S3 endpoint. Provided source = source-uri and region = aws-region (HTTP 400)
Unable to connect to Amazon S3. The AWS Region must be us-east-1. Ensure that you have a VPC endpoint. For information about creating a VPC endpoint, see Amazon S3 VPC Endpoint (p. 57).

Bucket is not in provided region (aws-region) (HTTP 400)
The bucket must be in the same AWS Region as your Neptune DB instance, us-east-1.

Unable to perform S3 list operation (HTTP 400)
The IAM user or role provided does not have List permissions on the bucket or the folder. Check the policy and/or the access control list (ACL) on the bucket.

Failed to start load because of unknown error from S3 (HTTP 500)
Amazon S3 returned an unknown error. Contact AWS Support.

Invalid S3 access key (HTTP 400)
Access key is invalid. Check the provided credentials.

Invalid S3 secret key (HTTP 400)
Secret key is invalid. Check the provided credentials.

Examples

Example Request
The following is a request sent via HTTP POST using the curl command. It loads a file in the Neptune CSV format. For more information, see Gremlin Load Data Format (p. 52).

curl -X POST \
   -H 'Content-Type: application/json' \
   http://your-neptune-endpoint:8182/loader -d ' \
   { "source" : "s3://bucket-name/object-key-name"},
Loader Get Status

Preview Release  Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

Gets the status of a loader job.

To get load status, you must send an HTTP GET request to the http://your-neptune-endpoint:8182/loader endpoint. To get the status for a particular load request, you must include the loadId as a URL parameter, or the loadId can be appended to the URL path.

Request Syntax

```
GET http://your-neptune-endpoint:8182/loader?loadId=loadId
GET http://your-neptune-endpoint:8182/loader/loadId
GET http://your-neptune-endpoint:8182/loader
```

Request Parameters

**loadId**

The ID of the load job. If you do not specify a loadId, a list of load IDs is returned.

details

Include details beyond overall status. Default: False

**Allowed values**: TRUE, FALSE

errors

Include the list of errors. The list of errors is paged. The page and errorsPerPage parameters allow you to page through all the errors. Default: False

**Allowed values**: TRUE, FALSE
The error page number. Only valid with the `errors` parameter set to TRUE. **Default:** 1

**Allowed values:** Positive integers

**errorsPerPage**

The number of errors per each page. Only valid with the `errors` parameter set to TRUE. **Default:** 10

**Allowed values:** Positive integers

**limit**

The number of load ids to list. Only valid when requesting a list of load IDs by sending a GET request with no `loadId` specified. **Default:** 100

**Allowed values:** Positive integers, 1 - 100

### Response Syntax

```json
{
  "status" : "200 OK",
  "payload" : {
    "feedCount" : [
      {
        "LOAD_FAILED" : int
      }
    ],
    "overallStatus" : {
      "datatypeMismatchErrors" : int,
      "fullUri" : "s3://bucket/key",
      "insertErrors" : int,
      "parsingErrors" : int,
      "retryNumber" : int,
      "runNumber" : int,
      "status" : "string",
      "totalDuplicates" : int,
      "totalRecords" : int,
      "totalTimeSpent" : float
    }
  }
}
```

**200 OK**

Successful status check returns a 200 code.

**Errors**

When an error occurs, a JSON object is returned in the **BODY** of the response. The `message` object contains a description of the error.

**Error 400**

An invalid `loadId` returns a 400 bad request error. The message describes the error.

**Error 500**

A valid request that cannot be processed returns a 500 internal server error. The message describes the error.
Loader Status Descriptions

The following are possible status responses from the loader with a description of the error.

**LOAD_NOT_STARTED**
Load has been recorded but not started.

**LOAD_IN_PROGRESS**
Load has started and is in progress.

**LOAD_COMPLETED**
Load has completed without any errors or errors within an acceptable threshold.

**LOAD_CANCELLED_BY_USER**
Load has been cancelled by user.

**LOAD_CANCELLED_DUE_TO_ERRORS**
Load has been cancelled by the system due to errors.

**LOAD_UNEXPECTED_ERROR**
Load failed with an unexpected error.

**LOAD_FAILED**
Load was rolled back because the error threshold was breached.

**LOAD_S3_READ_ERROR**
Feed failed due to intermittent or transient Amazon S3 connectivity issues. If any of the feeds receive this error, overall load status is set to LOAD_FAILED.

**LOAD_S3_ACCESS_DENIED_ERROR**
Access was denied to the S3 bucket. If any of the feeds receive this error, overall load status is set to LOAD_FAILED.

**LOAD_COMMITTED_W_WRITE_CONFLICTS**
Loaded data committed with unresolved write conflicts.

**LOAD_DATA_DEADLOCK**
Load was automatically rolled back due to deadlock.

Examples

**Example Request for Load Status**

The following is a request sent via HTTP GET using the `curl` command.

```
curl -X GET 'http://your-neptune-endpoint:8182/loader/0a237328-afd5-4574-a0bc-c29ce5f54802'
```

**Example Response**

```
{

```
Example Request for Load Ids

The following is a request sent via HTTP GET using the curl command.

```
curl -X GET 'http://your-neptune-endpoint:8182/loader?limit=3'
```

Example Response

```
{
  "status" : "200 OK",
  "payload" : {
    "loadIds" : [
      "a2c0ce44-a44b-4517-8cd4-1dc144a8e5b5",
      "09683a01-6f37-4774-bb1b-5620d87f1931",
      "58085eb8-ceb4-4029-a3dc-3840969826b9"
    ]
  }
}
```

Example Request for Detailed Load Status

The following is a request sent via HTTP GET using the curl command.

```
curl -X GET 'http://your-neptune-endpoint:8182/loader/0a237328-afd5-4574-a0bc-c29ce5f54802?details=true'
```

Example Response

```
{
  "status" : "200 OK",
  "payload" : {
    "failedFeeds" : [
      {
        "datatypeMismatchErrors" : 0,
        "fullUri" : "s3://bucket/key",
        "insertErrors" : 0,
        "parsingErrors" : 5,
        "retryNumber" : 0,
        "runNumber" : 1,
        "status" : "LOAD_FAILED",
        "totalDuplicates" : 0,
        "totalRecords" : 5,
        "totalTimeSpent" : 3.0
      }
    ]
  }
}
```
Example Request for Detailed Status with Load Errors

The following is a request sent via HTTP GET using the curl command.

curl -X GET 'http://your-neptune-endpoint:8182/loader/0a237328-afd5-4574-a0bc-c29ce5f54802?details=true&errors=true&page=1&errorsPerPage=3'

Example Response

```json
{
    "status": "200 OK",
    "payload": {
        "failedFeeds": [
            {
                "datatypeMismatchErrors": 0,
                "fullUri": "s3://bucket/key",
                "insertErrors": 0,
                "parsingErrors": 5,
                "retryNumber": 0,
                "runNumber": 1,
                "status": "LOAD_FAILED",
                "totalDuplicates": 0,
                "totalRecords": 5,
                "totalTimeSpent": 3.0
            }
        ],
        "feedCount": [
            {
                "LOAD_FAILED": 1
            }
        ],
        "overallStatus": {
            "datatypeMismatchErrors": 0,
            "fullUri": "s3://bucket/key",
            "insertErrors": 0,
            "parsingErrors": 5,
            "retryNumber": 0,
            "runNumber": 1,
            "status": "LOAD_FAILED",
            "totalDuplicates": 0,
            "totalRecords": 5,
            "totalTimeSpent": 3.0
        }
    }
}
```
Loader Cancel Job

Preview Release  Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

Cancels a load job.

To load data, you must send an HTTP DELETE request to the http://your-neptune-endpoint:8182/loader endpoint. The loadId can be appended to the /loader URL path, or included as a variable in the URL.

Request Syntax

DELETE http://your-neptune-endpoint:8182/loader?loadId=loadId

DELETE http://your-neptune-endpoint:8182/loader/loadId

Request Parameters

loadId

The ID of the load job.
Response Syntax

| no response body |

200 OK

Successfully deleted load job returns a 200 code.

Errors

When an error occurs, a JSON object is returned in the BODY of the response. The message object contains a description of the error.

Error 400

An invalid loadId returns a 400 bad request error. The message describes the error.

Error 500

A valid request that cannot be processed returns a 500 internal server error. The message describes the error.

Examples

Example Request

The following is a request sent via HTTP DELETE using the curl command.

curl -X DELETE 'http://your-neptune-endpoint:8182/loader/0a237328-afd5-4574-a0bc-c29ce5f54802'
Amazon Neptune DB Instance Lifecycle

Preview Release Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

The lifecycle of a DB instance includes creating, modifying, maintaining, performing backups, rebooting, and deleting the instance. This section provides information about these processes.

Topics
- Backing Up and Restoring Amazon Neptune DB Instances (p. 71)
- Amazon Neptune DB Parameter Groups (p. 76)
- Modifying a Neptune DB Instance and Using the Apply Immediately Parameter (p. 78)
- Renaming a DB Instance (p. 81)
- Rebooting a DB Instance (p. 82)
- Deleting a DB Instance (p. 83)
Back up and restore snapshots of a Neptune DB instance.

**Important**
Restoring snapshots is not supported at this time.

**Topics**
- Working with Backups (p. 71)
- Creating a Snapshot (p. 74)

**Working with Backups**

Amazon Neptune creates and saves automated backups of your DB instance. It creates a storage volume snapshot of your DB instance, backing up the entire DB instance and not just individual databases. Neptune creates automated backups during the backup window of your DB instance. It saves the backups according to the backup retention period that you specify. If necessary, you can recover your database to any point in time during the backup retention period.

Your DB instance must be in the **ACTIVE** state for automated backups to occur. If your database is in another state, for example **STORAGE_FULL**, automated backups don't occur.

You can also back up your DB instance manually by creating a DB snapshot. For more information about creating a DB snapshot, see **Creating a Snapshot** (p. 74).

You can copy both automatic and manual DB snapshots, and share manual DB snapshots.

**Backup Storage**

Your Neptune backup storage for each AWS Region is composed of the automated backups and manual DB snapshots for that Region. Your backup storage is equivalent to the sum of the database storage for all instances in that Region. Moving a DB snapshot to another Region increases the backup storage in the destination Region.

All automated backups are deleted when you delete a DB instance. After you delete a DB instance, the automated backups can't be recovered. If you choose to have Neptune create a final DB snapshot before it deletes your DB instance, you can use that to recover your DB instance.

Manual snapshots are not deleted.

**Backup Window**

Automated backups occur daily during the preferred backup window. If the backup requires more time than allotted to the backup window, the backup continues after the window ends, until it finishes. The backup window can't overlap with the weekly maintenance window for the DB instance.
During the automatic backup window, storage I/O might be suspended briefly while the backup process initializes (typically under a few seconds). You might experience elevated latencies for a few minutes during backups for Multi-AZ deployments.

If you don’t specify a preferred backup window when you create the DB instance, Neptune assigns a default 30-minute backup window. This window is selected at random from an eight-hour block of time per Region.

Neptune is currently available only in the US East (N. Virginia) Region. The default backup window for the US East (N. Virginia) Region is 03:00–11:00 UTC.

Backup Retention Period

You can set the backup retention period when you create a DB instance. If you don’t set the backup retention period, the default backup retention period is seven days if you create the DB instance using the AWS Management Console. For DB clusters, the default backup retention period is one day regardless of how the DB cluster is created.

After you create a DB instance, you can modify the backup retention period. You can set the backup retention period to between 1 and 35 days. You can also set the backup retention period to 0, which disables automated backups. Manual snapshot limits (100 per AWS Region) don’t apply to automated backups.

Important

An outage occurs if you change the backup retention period from 0 to a non-zero value or from a non-zero value to 0.

Disabling Automated Backups

In certain situations, you might want to disable automated backups temporarily; for example, while loading large amounts of data.

Important

We highly discourage disabling automated backups because it disables point-in-time recovery. Disabling automatic backups for a DB instance deletes all existing automated backups for the instance. If you disable and then re-enable automatic backups, you can only restore starting from the time you re-enabled automated backups.

In this example, you disable automated backups for a DB instance named mydbinstance by setting the backup retention parameter to 0.

Disabling Automated Backups Using the Console

Follow these steps to use the AWS Management Console to disable automated backups immediately for your DB instance.

To disable automated backups immediately

2. In the navigation pane, choose DB Instances, and then choose the DB instance that you want to modify.
3. Choose Instance Actions, and then choose Modify. The Modify DB Instance window appears.
4. For Backup Retention Period, choose 0.
5. Choose Apply Immediately.
6. Choose Continue.
7. On the confirmation page, choose Modify DB Instance to save your changes and disable automated backups.
Enabling Automated Backups

If your DB instance doesn't have automated backups enabled, you can enable them at any time. You enable automated backups by setting the backup retention period to a positive non-zero value. When automated backups are enabled, an outage occurs and a backup is immediately created.

In this example, you enable automated backups for a DB instance named `mydbinstance` by setting the backup retention period to a positive non-zero value (in this case, 3).

Enabling Automated Backups Using the Console

Use the AWS Management Console to enable automated backups immediately for your DB instance.

To enable automated backups immediately

2. In the navigation pane, choose DB Instances, and then choose the DB instance that you want to modify.
3. Choose Instance Actions, and then choose Modify. The Modify DB Instance page appears.
4. For Backup Retention Period, choose a positive non-zero value, for example, 3.
5. Choose Apply Immediately.
6. Choose Continue.
7. On the confirmation page, choose Modify DB Instance to save your changes and enable automated backups.
Creating a Snapshot

Amazon Neptune creates a storage volume snapshot of your DB instance, backing up the entire DB instance and not just individual databases. Creating this DB snapshot on a Single-AZ DB instance results in a brief I/O suspension that can last from a few seconds to a few minutes, depending on the size and class of your DB instance. Multi-AZ DB instances are not affected by this I/O suspension because the backup is taken on the standby.

When you create a DB snapshot, identify which DB instance you are going to back up, and then give your DB snapshot a name so that you can restore from it later. If you have IAM database authentication enabled, this setting is inherited from the source DB instance.

**Important**
Restoring snapshots is not supported at this time.

Creating a DB Snapshot Using the Console

Follow these steps to create a DB snapshot in the AWS Management Console.

**To create a DB snapshot**

2. In the navigation pane, choose **Instances**.
3. Choose **Instance Actions**, and then choose **Take Snapshot**.
4. The **Take DB Snapshot** dialog box appears.
5. In the **Snapshot name** box, type the name of the snapshot.
5. Choose **Take Snapshot**.
Amazon Neptune DB Parameter Groups

You manage your database configuration in Amazon Neptune by using parameters in a DB parameter group. DB parameter groups act as a container for engine configuration values that are applied to one or more DB instances.

There are two types of DB parameter groups: DB cluster parameter groups and DB parameter groups.

- DB cluster parameter groups apply to every instance in the cluster and generally have broader settings. An example is the `neptune_enable_tls` parameter that is used to toggle Secure Sockets Layer (SSL).
- DB parameter groups apply at the instance level and generally are associated with the Neptune graph engine, such as the `neptune_query_timeout` parameter.

A default DB parameter group is used if you create a DB instance without specifying a custom DB parameter group. You can't modify the parameter settings of a default DB parameter group. You must create your own DB parameter group to change parameter settings from their default value. Not all DB engine parameters can be changed in a custom DB parameter group.

Here are some important points you should know about working with parameters in a DB parameter group:

- When you change a static parameter and save the instance DB parameter group, the parameter change takes effect after you manually reboot the DB instance.
- When you change a static parameter and save the DB cluster parameter group, the parameter change takes effect after you manually reboot every DB instance in the cluster.
- Improperly setting parameters in a DB parameter group can have unintended adverse effects, including degraded performance and system instability. Always exercise caution when modifying database parameters, and back up your data before modifying a DB parameter group. Try out your parameter group setting changes on a test DB instance before applying those changes to a production DB instance.

Editing a DB Parameter Group

2. Choose **Parameter groups** in the navigation pane.
3. Follow the **Name** link for the DB parameter group that you want to edit.
   
   (Optional) Choose **Create Parameter Group** to create a new cluster parameter group and create the new group. Then choose the **Name** of the new parameter group.

   **Important**
   This is required if you only have the default DB cluster parameter group because the default DB cluster parameter group can't be modified.

4. Choose **Edit Parameters**.
5. Set the value for the parameters that you want to change.
6. Choose **Save changes**.
7. Reboot every Neptune DB instance in the Neptune cluster.

## Creating a DB Parameter Group

2. Choose **Parameter Groups** in the left navigation pane.
3. Choose **Create DB Parameter Group**.
   
   The **Create DB Parameter Group** screen appears.
4. In the **Type** list, choose **DB Parameter Group** or **DB Cluster Parameter Group**.
5. In the **DB Parameter Group** box, type the name of the new DB parameter group.
6. In the **Description** box, type a description for the new DB parameter group.
7. Choose **Yes, Create**.
Modifying a Neptune DB Instance and Using the Apply Immediately Parameter

**Preview Release** Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

Most modifications to an Amazon Neptune DB instance can be applied immediately or deferred until the next maintenance window. Some modifications, such as parameter group changes, require that you manually reboot your DB instance for the change to take effect.

**Important**
Some modifications result in an outage because Neptune must reboot your DB instance for the change to take effect. Review the impact to your database and applications before modifying your DB instance settings.

**Impact of the Apply Immediately Option**

When you modify a DB instance, you can apply the changes immediately. To apply changes immediately, you choose the **Apply Immediately** option in the AWS Management Console.

If you don’t choose to apply changes immediately, the changes are put into the pending modifications queue. During the next maintenance window, any pending changes in the queue are applied.

**Important**
If you choose to apply changes immediately, any changes in the pending modifications queue are also applied. If any of the pending modifications require downtime, choosing to apply changes immediately can cause unexpected downtime.

**Common Settings and Downtime Notes**

The following table contains details about which settings you can modify, when the changes can be applied, and whether the changes cause downtime for the DB instance.

**Important**
If you choose to apply changes immediately, any changes in the pending modifications queue are also applied. If any of the pending modifications require downtime, choosing to apply immediately can cause unexpected downtime.

<table>
<thead>
<tr>
<th>DB instance setting</th>
<th>When the change occurs</th>
<th>Downtime notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocated Storage</td>
<td>If <strong>Apply Immediately</strong> is set to true, the change occurs immediately. If <strong>Apply Immediately</strong> is set to false, the change occurs during the next maintenance window.</td>
<td>No downtime. Performance might be degraded during the change.</td>
</tr>
<tr>
<td>Auto Minor Version Upgrade</td>
<td>The change is applied asynchronously, as soon as possible. This setting ignores the <strong>Apply Immediately</strong> setting.</td>
<td>An outage occurs if a newer minor version is available, and Neptune has enabled automatic patching for that version.</td>
</tr>
<tr>
<td>DB instance setting</td>
<td>When the change occurs</td>
<td>Downtime notes</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Backup Retention Period</td>
<td>If <strong>Apply Immediately</strong> is set to true, the change occurs immediately.</td>
<td>An outage occurs if you change from 0 to a nonzero value, or from a nonzero value to 0.</td>
</tr>
<tr>
<td></td>
<td><strong>If Apply Immediately</strong> is set to false, and you change the setting from a nonzero value to another nonzero value, the change is applied asynchronously, as soon as possible. Otherwise, the change occurs during the next maintenance window.</td>
<td></td>
</tr>
<tr>
<td>Backup Window</td>
<td>The change is applied asynchronously, as soon as possible.</td>
<td></td>
</tr>
<tr>
<td>DB Instance Class</td>
<td>If <strong>Apply Immediately</strong> is set to true, the change occurs immediately.</td>
<td>An outage occurs during this change.</td>
</tr>
<tr>
<td></td>
<td><strong>If Apply Immediately</strong> is set to false, the change occurs during the next maintenance window.</td>
<td></td>
</tr>
<tr>
<td>DB Instance Identifier</td>
<td>If <strong>Apply Immediately</strong> is set to true, the change occurs immediately.</td>
<td>An outage occurs during this change. The DB instance is rebooted.</td>
</tr>
<tr>
<td></td>
<td><strong>If Apply Immediately</strong> is set to false, the change occurs during the next maintenance window.</td>
<td></td>
</tr>
<tr>
<td>DB Parameter Group</td>
<td><strong>Note:</strong> Parameter group can only be changed for an entire cluster.</td>
<td>An outage doesn't occur during this change. However, parameter changes only occur when you reboot the DB instance manually without failover.</td>
</tr>
<tr>
<td></td>
<td>The parameter group change occurs immediately. However, parameter changes only occur when you reboot the DB instance manually without failover.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For more information, see [Rebooting a DB Instance](p. 82).</td>
<td></td>
</tr>
<tr>
<td>DB instance setting</td>
<td>When the change occurs</td>
<td>Downtime notes</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Maintenance Window</td>
<td>The change occurs immediately. This setting ignores the <code>Apply Immediately</code> setting.</td>
<td>If there are one or more pending actions that cause an outage, and the maintenance window is changed to include the current time, those pending actions are applied immediately, and an outage occurs. If you set the window to the current time, there must be at least 30 minutes between the current time and end of the window to ensure that any pending changes are applied.</td>
</tr>
<tr>
<td>Security Group</td>
<td>The change is applied asynchronously, as soon as possible. This setting ignores the <code>Apply Immediately</code> setting.</td>
<td>–</td>
</tr>
</tbody>
</table>
Renaming a DB Instance

You can rename an Amazon Neptune DB instance by using the AWS Management Console. Renaming a DB instance can have far-reaching effects. The following is a list of things you should know before you rename a DB instance.

- When you rename a DB instance, the endpoint for the DB instance changes because the URL includes the name you assigned to the DB instance. You should always redirect traffic from the old URL to the new one.
- When you rename a DB instance, the old DNS name that was used by the DB instance is immediately deleted, but it can remain cached for a few minutes. The new DNS name for the renamed DB instance becomes effective after about 10 minutes. The renamed DB instance is not available until the new name becomes effective.
- You can't use an existing DB instance name when you are renaming an instance.
- All Read Replicas that are associated with a DB instance remain associated with that instance after it is renamed. For example, suppose that you have a DB instance that serves your production database, and the instance has several associated Read Replicas. If you rename the DB instance and then replace it in the production environment with a DB snapshot, the DB instance that you renamed still has the Read Replicas associated with it.
- Metrics and events that are associated with the name of a DB instance are maintained if you reuse a DB instance name. For example, if you promote a Read Replica and rename it to be the name of the previous master, the events and metrics that were associated with the master are then associated with the renamed instance.
- DB instance tags remain with the DB instance, regardless of renaming.
- DB snapshots are retained for a renamed DB instance.

Renaming a DB Instance Using the Console

Follow these steps to use the AWS Management Console to rename your Neptune DB instance.

**To rename a DB instance**

2. In the navigation pane, choose **DB Instances**.
3. Select the check box next to the DB instance that you want to rename.
4. In the **Instance Actions** drop-down menu, choose **Modify**.
5. Type a new name in the **DB Instance Identifier** text box. Select **Apply Immediately**, and then choose **Continue**.
6. Choose **Modify DB Instance** to complete the change.
Rebooting a DB Instance

In some cases, if you modify an Amazon Neptune DB instance, change the DB parameter group that is associated with the instance, or change a static DB parameter in a parameter group that the instance uses, you must reboot the instance for the changes to take effect.

Rebooting a DB instance restarts the database engine service. A reboot also applies to any changes to the associated DB parameter group that were pending. Rebooting a DB instance results in a momentary outage of the instance, during which the DB instance status is set to rebooting. If the Neptune instance is configured for Multi-AZ, the reboot might be conducted through a failover. A Neptune event is created when the reboot is completed.

If your DB instance is a Multi-AZ deployment, you can force a failover from one Availability Zone to another when you choose the Reboot option. When you force a failover of your DB instance, Neptune automatically switches to a standby replica in another Availability Zone and updates the DNS record for the DB instance to point to the standby DB instance. As a result, you must clean up and re-establish any existing connections to your DB instance.

**Reboot with failover** is beneficial when you want to simulate a failure of a DB instance for testing or restore operations to the original Availability Zone after a failover occurs. For more information, see High Availability (Multi-AZ). When you reboot a DB cluster, it fails over to the standby replica. Rebooting a Neptune replica does not initiate a failover.

The time required to reboot is a function of the crash recovery process. To improve the reboot time, we recommend that you reduce database activities as much as possible during the reboot process to reduce rollback activity for in-transit transactions.

In the console, the Reboot option may be disabled if the DB instance is not in the Available state. This can be due to several reasons, such as an in-progress backup, a customer-requested modification, or a maintenance-window action.

**Note**

Rebooting the primary instance of an Amazon Neptune DB cluster also automatically reboots the Neptune replicas for that DB cluster.

Rebooting a DB Instance Using the Console

Follow these steps to reboot a DB instance in the AWS Management Console.

**To reboot a DB instance**

2. In the navigation pane, choose Instances.
3. Select the check box of the DB instance that you want to reboot.
4. Choose Instance Actions, and then choose Reboot from the drop-down menu.
5. To force a failover from one AZ to another, select Reboot with failover? in the Reboot DB Instance dialog box.
6. Choose Yes, Reboot. To cancel the reboot, choose Cancel instead.
Deleting a DB Instance

You can delete an Amazon Neptune DB instance in any state and at any time. To delete a DB instance, you must specify the name of the instance and specify if you want to have a final DB snapshot taken of the instance. If the DB instance that you're deleting has a status of Creating, you can't have a final DB snapshot taken. If the DB instance is in a failure state with a status of failed, incompatible-restore, or incompatible-network, you can only delete the instance when the SkipFinalSnapshot parameter is set to true.

**Important**
If you choose not to create a final DB snapshot, you can't later restore the DB instance to its final state. When you delete a DB instance, all automated backups are deleted and cannot be recovered. Manual DB snapshots of the instance are not deleted.

When you delete all instances in a cluster, the cluster is deleted, too.

If the DB instance that you want to delete has a Read Replica, you should either promote the Read Replica or delete it.

In the following examples, you delete a DB instance both with and without a final DB snapshot.

### Deleting a DB Instance with No Final Snapshot

If you want to quickly delete a DB instance, you can skip creating a final DB snapshot. When you delete a DB instance, all automated backups are deleted and cannot be recovered. Manual snapshots are not deleted.

### Deleting a DB Instance Using the Console

Follow these steps to use the AWS Management Console to delete a Neptune DB instance without a final DB snapshot.

**To delete a DB instance with no final DB snapshot**

2. In the DB Instances list, select the check box next to the DB instance that you want to delete.
3. Choose Instance Actions, and then choose Delete from the menu.
4. Choose No in the Create final Snapshot? drop-down list.
5. Choose Yes, Delete.

### Deleting a DB Instance with a Final Snapshot

If you want to be able to restore a deleted DB instance at a later time, you can create a final DB snapshot. All automated backups are also deleted and cannot be recovered. Manual snapshots are not deleted.

### Deleting a DB Instance Using the Console

Follow these steps to use the AWS Management Console to delete a Neptune DB instance with a final DB snapshot.
To delete a DB instance with a final DB snapshot

2. In the DB Instances list, select the check box next to the DB Instance that you want to delete.
3. Choose Instance Actions, and then choose Delete from the menu.
5. In the Final Snapshot name box, type the name of your final DB snapshot.
6. Choose Yes, Delete.
Amazon Neptune User Guide

Enabling Encryption

Encrypting Neptune Resources

**Preview Release**  Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

Amazon Neptune encrypted instances use the AES-256 encryption algorithm to encrypt your data on the server that hosts your Neptune instance. After your data is encrypted, Neptune handles authentication of access and decryption of your data transparently with a minimal impact on performance. You don't need to modify your database client applications to use encryption.

Neptune encrypted instances provide an additional layer of data protection by securing your data from unauthorized access to the underlying storage. You can use Neptune encryption to increase data protection of your applications that are deployed in the cloud, and to fulfill compliance requirements for data-at-rest encryption.

To manage the keys used for encrypting and decrypting your Neptune resources, you use AWS Key Management Service (AWS KMS). AWS KMS combines secure, highly available hardware and software to provide a key management system scaled for the cloud. Using AWS KMS, you can create encryption keys and define the policies that control how these keys can be used. AWS KMS supports AWS CloudTrail, so you can audit key usage to verify that keys are being used appropriately. Your AWS KMS keys can be used in combination with Neptune and supported AWS services such as Amazon Simple Storage Service (Amazon S3), Amazon Elastic Block Store (Amazon EBS), and Amazon Redshift. For a list of services that support AWS KMS, see Supported Services in the AWS Key Management Service Developer Guide.

All logs, backups, and snapshots are encrypted for a Neptune encrypted instance.

### Enabling Encryption for a Neptune DB Instance

To enable encryption for a new Neptune DB instance, choose Yes in the Enable encryption section on the Neptune console. For information about creating a Neptune DB instance, see Getting Started with Neptune (p. 13).

When you create an encrypted Neptune DB instance, you can also supply the AWS KMS key identifier for your encryption key. If you don't specify an AWS KMS key identifier, Neptune uses your default encryption key for your new Neptune DB instance. AWS KMS creates your default encryption key for Neptune for your AWS account. Your AWS account has a different default encryption key for each AWS Region.

After you create an encrypted Neptune DB instance, you can't change the encryption key for that instance. So, be sure to determine your encryption key requirements before you create your encrypted Neptune DB instance.

You can use the Amazon Resource Name (ARN) of a key from another account to encrypt a Neptune DB instance. If you create a Neptune DB instance with the same AWS account that owns the AWS KMS encryption key that's used to encrypt that new Neptune DB instance, the AWS KMS key ID that you pass can be the AWS KMS key alias instead of the key's ARN.

**Important**

If Neptune loses access to the encryption key for a Neptune DB instance—for example, when Neptune access to a key is revoked—the encrypted DB instance is placed into a terminal state and can only be restored from a backup. We strongly recommend that you always enable
backups for encrypted Neptune DB instances to guard against the loss of encrypted data in your databases.
Amazon Neptune Limits

**Preview Release**  Amazon Neptune Preview is available only to whitelisted customers. To request access to Neptune, see the information on the Amazon Neptune Preview page.

**Instance Limit**

Amazon Neptune has a limit of three instances per account.

You can request an increase on this limit. For more information, see https://aws.amazon.com/support.

**Account Limits**

The following are per-account limits.

<table>
<thead>
<tr>
<th>Item</th>
<th>Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clusters</td>
<td>20</td>
</tr>
<tr>
<td>DB Subnet Groups</td>
<td>50</td>
</tr>
<tr>
<td>DB Snapshots</td>
<td>100</td>
</tr>
<tr>
<td>DB Security Groups (Per VPC)</td>
<td>25</td>
</tr>
</tbody>
</table>

You can request an increase on some limits. For more information, see https://aws.amazon.com/support.

**Console Access Required**

Creating and modifying Amazon Neptune instances and clusters must be done through the AWS Management Console.

**VPC Required**

Amazon Neptune is a virtual private cloud (VPC)-only service. Additionally, instances do not allow access from outside the VPC.

**Availability Zones and DB Subnet Groups**

Amazon Neptune requires a DB subnet group for each cluster that has subnets in at least two supported Availability Zones. We recommend using three or more subnets in different Availability Zones.

Amazon Neptune is not supported in every Availability Zone. If you receive the console error **DB Subnet Group doesn't meet availability zone coverage requirement**, try adding subnets in additional Availability Zones to the DB subnet group.

**Gremlin Implementation**

The Amazon Neptune Gremlin implementation has specific implementation details. For more information, see Neptune Gremlin Implementation Differences (p. 22).

**SPARQL UPDATE LOAD**
SPARQL UPDATE LOAD from URI only works with resources within the same VPC. This includes Amazon S3 URLs in the us-east-1 Region with an Amazon S3 VPC endpoint created. For information about creating a VPC endpoint, see Amazon S3 VPC Endpoint (p. 57).

The Amazon S3 URL must be HTTPS, and any authentication must be included in the URL. For more information, see Authenticating Requests: Using Query Parameters.

If you need to load data from a file, we recommend using the Amazon Neptune loader API. For more information, see Loading Data into Neptune (p. 48).

**Note**
The Amazon Neptune loader API is non-ACID.

**Authentication and Access**

IAM authentication and access control are not supported for Gremlin, SPARQL, or cluster/instance level.

The Amazon Neptune console requires AmazonRDSFullAccess permissions. You can restrict access to IAM users by revoking this access.

Amazon Neptune does not support user name/password–based access control.